UNIPAC - UNIVERSIDADE PRESIDENTE ANTÔNIO CARLOS
FACULDADE DE CIÊNCIA DA COMPUTAÇÃO E COMUNICAÇÃO SOCIAL

CURSO DE CIÊNCIA DA COMPUTAÇÃO

L E A N D R O C É S A R M E N D E S

SERVIÇOS *WEB* UTILIZANDO JAVA

BARBACENA
DEZEMBRO DE 2004

LEANDRO CÉSAR MENDES

SERVIÇOS *WEB* UTILIZANDO JAVA

Trabalho de conclusão de curso apresentado à Universidade Presidente Antônio Carlos como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

ORIENTADOR: Prof. Luis Augusto Mattos Mendes

BARBACENA
DEZEMBRO DE 2004

LEANDRO CÉSAR MENDES

SERVIÇOS *WEB* UTILIZANDO JAVA

Este trabalho de conclusão de curso foi julgado adequado à obtenção do grau de Bacharel em Ciência da Computação e aprovado em sua forma final pelo Curso de Ciência da Computação da Universidade Presidente Antônio Carlos.

Aprovada em _____/_____/_____

BANCA EXAMINADORA

_____

Prof. Luis Augusto Matos Mendes (Orientador)
Universidade Presidente Antônio Carlos

_____

Prof. Ms Emerson Rodrigo Alves Tavares (Membro Examinador)
Universidade Presidente Antônio Carlos

_____

Prof. Eduardo Macedo Bhering (Membro Examinador)
Universidade Presidente Antônio Carlos

Dedico este trabalho aos meus familiares, em especial a minha mãe, Conceição, pelo apoio e compreensão durante os momentos difíceis onde pensei em até desistir e pela iniciativa conjunta com a minha irmã que me pagaram um curso de Informática qual foi o que me fez optar pelo Curso de Ciência da Computação; a minha irmã, Luciene que, além da iniciativa conjunta com minha mãe, durante todo o curso me incentivou e contribui financeiramente para que eu estudasse; ao meu Pai que sempre se prestou disposto a me acordar pela manhã com um café e com um carinho enorme para que eu pudesse trabalhar e assim correr atrás de meus objetivos. Dedico a minha namorada, Sônia, que teve muita paciência quando nos finais de semanas que com ela não pude estar, pois, estava escrevendo um trabalho ou estudando para uma prova e pelo o amor e respeito dedicados. E aos meus amigos Paulo, Fausto, Zé Lúcio, Pietro, Jaime, Sérgio e Patrick.

Agradeço aos meus amigos: Antônio (Toninho) que me deu a oportunidade de ingressar nas atividades da informática, ao Geraldo Brigido que contribui diretamente com o tema deste trabalho, ao meu orientador Prof. Luis Augusto que se mostrou bastante interessado no trabalho e que me proporcionou um crescimento profissional muito grande.
Agradeço ao Felipe pelas dicas de Java que foram de um grande valor para este trabalho e Agradeço a Deus por ter tanta gente boa a meu redor e por ter me dado uma capacidade técnica e habilidades para desempenhar meu trabalho de forma correta e digna.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Um sistema distribuído envolve tecnologias diversas que empregadas de forma correta, pode trazer aumentos significativos ao processamento de informações e a implementação de novas soluções. A utilização de Serviços *Web* como forma de se implementar tais sistemas facilita em muitos aspectos a implementação e a compreensão de um cenário heterogêneo e em um ambiente também heterogêneo que é a *Internet*/*Intranet*.

Os avanços computacionais permitiram a integração entre sistemas onde surgiram arquiteturas que são divididas em camadas, como *Client/Server* onde uma gama maior de processamento é realizada no lado do servidor, ficando assim um volume menor de processamento para o lado cliente. Esse avanço nos permite hoje, a criação de sistemas ainda mais complexos, como os sistemas distribuídos. Tais sistemas são particionados em pequenos fragmentos que darão origem a um único sistema.

Atualmente o cenário empresarial requer uma interoperabilidade entre aplicações internas com aplicações externas a seu domínio, nesses casos muitas empresas têm receios quanto a esta interoperabilidade, uma vez que se faz necessário deixar uma “porta” aberta para uma comunicação entre as aplicações, além disso, as tecnologias existentes até pouco tempo atrás não permitiam um intercâmbio cem por cento, uma vez que estas dependiam de plataformas e linguagens de programação. Os Serviços *Web* surgiram para resolver este problema, pois, são totalmente independentes de plataforma ou de linguagens de programação.

Este trabalho visa aproveitar os recursos oferecidos pela linguagem Java e ferramentas totalmente implementadas nessa para a implementação de um Serviço *Web* denominado **WEBCALC,** capaz de fornecer serviços relacionados a cálculos legais (Imposto de Renda e Imposto Nacional de Seguridade Social) aplicados a cálculos de folha de pagamento.

Durante as explanações teóricas informações suficientes sobre os cenários dos Serviços *Web* e tecnologias Java que podem ser utilizadas serão passadas.

O trabalho está organizado de forma que uma linha de raciocínio sobre o cenário, necessário para se implementar os Serviços *Web*, possa ser criada de forma clara e objetiva. Ficando o mesmo dividido assim:

No Capítulo 2 são apresentadas informações sobre a *Internet* e sua arquitetura para que se possamos visualizar com clareza o ambiente onde são empregados, com maior freqüência, os Serviços *Web*. São também apresentados alguns dos protocolos que são usados na arquitetura.

O Capítulo 3 fornece dados necessários para se compreender os Serviços *Web*, como podemos utilizá-los, os protocolos empregados, de fundamental compreensão para se implementar os Serviços *Web*, assim como questões teóricas sobre segurança que podem e devem ser aplicadas ao ambiente.

O Capitulo 4 traz informações sobre segurança. Formas e padrões baseados em XML que são utilizados para implementar segurança para Serviços *Web*.

No Capitulo 5 são mostradas algumas tecnologias e ferramentas Java que serão utilizadas na implementação do WEBCALC e que podem ser utilizadas em qualquer aplicação distribuída.

No Capítulo 6 a implementação do serviço WEBCALC, explicando passo-a-passo a sua implementação e o cenário onde este poderá ser utilizado. Esta implementação, completamente em Java, mostrará também como utilizar o AXIS (*Apache Extensible Interaction System*) juntamente com o Tomcat para publicação e disponibilização dos Serviços *Web*.

O Capítulo 7 traz as conclusões do trabalho e sugestões para trabalhos futuros.

2 INTERNET

O propósito maior de se estudar Serviços *Web* é compreender uma das tecnologias de sistemas distribuídos que ainda é muito recente. No intuito de se fazer uma boa justificativa para o uso dessas tecnologias julgou-se necessário escrever em um trabalho sobre Serviços *Web* um pouco sobre o funcionamento da *Internet*, para que se possa ter uma visão geral do ambiente de trabalho utilizado pelos Serviços *Web*.

Então o que vem a ser *Internet*? *Internet* é uma rede de computadores mundialmente interligados, isso nos diz que qualquer computador ligado a *Internet* pode ter acesso aos demais computadores, também ligados. A *Internet* não pertence a nenhum governo ou empresa. Surge nos Estados Unidos, na década de 60, na época da Guerra Fria, a partir de uma rede de informações militares que interliga centros de comando e de pesquisa bélica. Para atender à necessidade militar de proteger os sistemas de defesa do país no caso de um ataque nuclear, a rede não tem um "centro" que poderia servir de alvo principal ao inimigo. Nos anos 70 começa a ser utilizada pela comunidade acadêmica mundial, e em 1975, são feitas as primeiras ligações internacionais. Nesse período, os computadores conectados não passavam de 200. Em 1997, segundo a *Direct Marketing Association* (DMA) e a *Price Waterhouse*, empresas norte americanas de consultoria em *marketing*, o número de usuários chega a 60 milhões em todo o mundo. De acordo com as projeções de crescimento, alcançando 300 milhões no ano de 2000.

A *Internet* tem revolucionado a comunicação mundial ao permitir, por exemplo, a troca de *e-mail* (correio eletrônico). Com a *Internet* surge a expressão *ciberespaço*, que significa o espaço virtual e sem fronteiras, no qual circulam milhares de informações veiculadas na rede.

2.1 Arquitetura *Web*

Podemos dizer que a *Web* trabalha com uma arquitetura Cliente/Servidor de duas ou três camadas. Entenda por camadas como sendo o papel de cada um dentro do cenário. A Figura 1 mostra melhor o que é esta arquitetura.

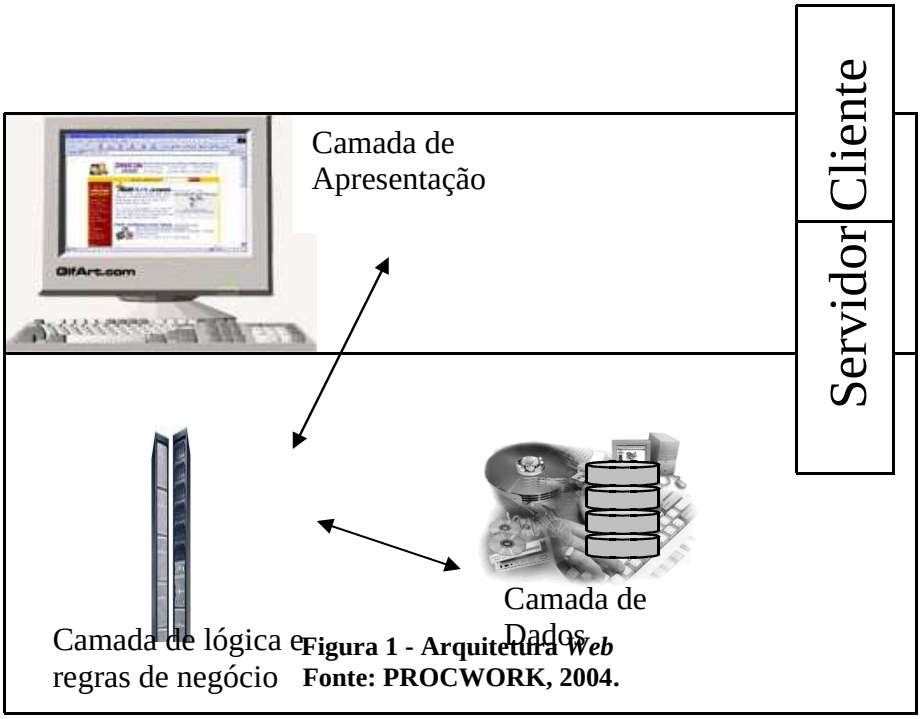


Figura 1 - Arquitetura *Web*
Fonte: PROCWORK, 2004.

Na Figura 1, o Cliente solicita um serviço ao Provedor de Acesso que por sua vez, verifica a disponibilidade de atender a requisição do Cliente e em caso positivo, uma resposta é retornada para o cliente. Em casos onde um acesso a uma base de dados se faz necessário, a Camada de regras do negócio é a responsável por fazê-lo. O cliente só vê o resultado final.

Na programação de *Web Sites* é comum o termo *Request/Response* ou Requisição/Resposta, que é a interação Cliente/Servidor. Em meio a este cenário os *Application Server* despontaram como soluções completas, de Desenvolvimento a Gerenciamento de Sistema. Os *Application Server* simplifica o desenvolvimento, facilita a integração, distribuição e

gerenciamento de “clientes leves” e sistemas de *Software* empresarial **distribuídos**. Os *Application Server* proporcionam soluções de negócio que integram ambientes empresariais heterogêneos, incluindo servidores *Web*, servidores de Banco de Dados, Legados, múltiplos clientes, linguagens e plataformas. Ao decorrer desta sessão estaremos focalizando os servidores *Web*.

Os servidores *Web* são programas que respondem chamadas através de conexões TCP/IP (*Transfer Control Protocol / Internet Protocol*) com arquivos que são, basicamente, gerados em HTML (*Hipertext Markup Language*). Esses *softwares* são instalados nas máquinas ditas como servidores, são máquinas com alto poder de processamento. Ao serem instalados, criam uma estrutura de diretórios e nestes são colocadas páginas *Web* que se deseja disponibilizar acesso. Quando executado os servidores *Web* abre uma porta de conexão normalmente a Porta 80, para que usuários internos no caso de uma *Intranet*, ou usuários externos no caso de *Internet* possam fazer seu acesso. Atualmente existem diversos *softwares* atuando como servidores *Web* disponíveis no mercado. Existem, tanto *softwares* de código aberto quanto *softwares* proprietários, o Apache é um exemplo de *Software* de código aberto e por outro lado temos o IIS (*Internet Information Server*) da Microsoft® , como exemplo de *Software* proprietário.

Um ponto importante a considerar sobre servidores *Web* é a configuração. Apesar de ser relativamente simples entender os princípios básicos desse tipo de *Software*, a configuração não é uma tarefa fácil. Na sessão 5.2.1 sobre o Tomcat será abordada uma forma simples de configurar um servidor *Web*.

2.1.1 Protocolos da arquitetura *Web*

O Intuito desta sessão não é o de esgotar e tão pouco dar detalhes técnicos aprofundados sobre protocolos, essa sessão tem como objetivo descrever alguns protocolos que podem ser usados no cenário dos Serviços *Web*, tentando assim, se fazer uma justificativa de seus usos.

A definição de protocolo é bem simples, trata-se de um padrão de comunicação entre computadores ou sistemas. Diante da necessidade de se trocar informações temos um processo de requisição e resposta. A transmissão se inicia com um pedido de leitura ou escrita de um arquivo, o qual serve também para pedir uma conexão. Se o servidor reconhece o serviço, a conexão é aceita e o arquivo e transmitido por um bloco de 512 bytes. Cada pacote contém um bloco de dados e deve ser reconhecido por um pacote de confirmação antes que outro pacote indique o término de uma transmissão.

Segundo a Procwork (2004) se um protocolo se extraviar na rede, o recebedor indicará *Timeout* e poderá retransmitir seu último pacote, que pode ser dados ou reconhecimento. Isso faz com que o transmissor do pacote perdido retransmita o mesmo pacote. As duas máquinas interligadas são, ao mesmo tempo, transmissoras e receptoras, uma envia reconhecimento e recebe dados. Muitos erros são gerados pelo término da conexão (podendo ser ocasionada por n motivos). Quando um desses erros é indicado um pacote de erro é enviado, este pacote não é reconhecido e nem transmitido, assim o outro terminal da conexão não deverá recebê-lo. Por tanto, os *Timeout* são usados para detectar tais terminais quando o pacote foi extraviado.

Para um melhor direcionamento dos estudos, as próximas sessões descrevem, de forma introdutória, os conceitos de cada protocolo.

2.1.1.1 HTTP (*Hipertext Transfer Protocol*)

O HTTP é um protocolo em nível de aplicação para sistemas de hipermídia colaborativos e distribuídos. De forma generalizada o protocolo HTTP pode ser usado para realizar diversas tarefas além de sua aplicação mais comum que é a troca de documentos hipermídia, ele pode ser usado para Servidores de Nomes e Sistemas Distribuídos para o gerenciamento de objetos. Uma característica muito interessante do protocolo HTTP é sua flexibilidade na aceitação de dados de diferentes formatos e origens. Por essa característica o HTTP tem sido usado na *Web* desde 1990 (GETTYS 2004). A primeira versão desse protocolo

(HTTP/0.9) era um protocolo simples para a troca de dados através da *Internet*. Hoje as capacidades deste estão muito avançadas e com grande credibilidade perante as empresas.

Atualmente os Sistemas de Informação requerem maiores funcionalidades que uma simples visualização de dados e pesquisas. Na especificação do Protocolo HTTP/1.1, definida por Gettys (2004), é dito que esse disponhe de um conjunto de métodos e um cabeçalho que indica o propósito de uma requisição. É construído um caminho de referência provido por um URI (*Universal Resource Indentifier*) indicando um local no servidor *Web* ou um nome URN (*Universal Resource Name*) para indicar os recursos com os quais os métodos do conjunto estão habilitados. Embora a especificação HTTP detalhe alguns métodos como PUT e DELETE somente dois métodos, segundo Weissinger (1999), são aceitos por todos os servidores: os Métodos *GET* e *POST*, que são comparados abaixo, na Tabela 1.

Tabela 1- Comparação entre os métodos *Post* e *Get*

Métodos http	Descrição
GET	Adiciona corpo da mensagem na URL, separando por ponto de interrogação. Aplicação: usada para pequenas consultas
POST	O corpo da mensagem é enviado como um fluxo de dados. Aplicação: usado para grandes consultas.

No caso de surgir uma confusão no entendimento da diferença entre os dois principais métodos do HTTP adote o seguinte: *GET* pode ser utilizado para encontrar qualquer documento, e *POST* não pode. Por outro lado tanto *GET* quanto *POST* podem ser utilizados para passar dados para o objeto especificado na URL. Quando *GET* é utilizado com esse propósito, os dados são incluídos na URL como uma seqüência de argumentos, dessa forma se você pretende usar estes parâmetros deve analisar a string URL. Quando *POST* é utilizado os dados são incluídos no corpo da mensagem. Sendo assim, quando *POST* e *GET* são usados para enviar dados eles se diferem pelo método utilizado para transmitir os dados.

Exemplo 2. 1 - Requisição HTTP de um browser (uma imagem)

```
GET /imagems/WEBCALC.gif HTTP/1.1
User-Agent: Netscape 6.0[en] (Windows2000; I)
Cookies: query=uiop;asdfg=hjkl; num=123
```

Exemplo 2. 2 - Exemplo de resposta HTTP

```
HTTP 1.1 200 OK
Server: Apache 1.02
Date: Friday, August 13, 2004 03:12:56 GMT-03
Content-type: image/gif
Content-length: 23779
```

2.1.1.2 TCP/IP (*Transfer Control Protocol / Internet Protocol*)

A plataforma TCP/IP surgiu através dos trabalhos do DARPA (*Defense Advanced Research Projects Agency*) dos Estados Unidos, em meados da década de 70, constituindo a ARPANET, que mais tarde se desmembrou em ARPANET, para pesquisa, e MILNET, para instituições militares. Para encorajar os pesquisadores universitários a adotar o TCP/IP, o DARPA fez uma implementação de baixo custo, integrando-o ao UNIX da Universidade de *Berkley* já em uso em todas as universidades americanas. Além disso, teve-se o cuidado de definir aplicações de rede similares às já conhecidas em Unix, como `rusers` e `rcp`.

Mais tarde a NSF (*National Science Foundation*) estimulou o seu crescimento criando a NSFNET, que ligava centros de supercomputação espalhados por todo o país, numa rede de longa distância, também com os protocolos TCP/IP. Existe um grupo chamado IAB (*Internet Activities Board*) que coordena os esforços de pesquisa na área, através de vários grupos de trabalho.

A documentação dos trabalhos, propostas para novos protocolos ou alteração de outros já existentes é feita através de artigos conhecidos como RFCs (*Request for Comments*). Propostas ainda em estudos são chamadas de IEN (*Internet Engineering Notes*) ou *Internet Drafts*.Tanto as RFCs quanto as IENs são numeradas seqüencialmente e em ordem cronológica. São distribuídas pelo SRI-NIC, órgão que executa várias tarefas administrativas na *Internet*.

Figura 2 - Arquitetura TCP/IP
Fonte: IETF, 1981

A Figura 2, mostra a arquitetura TCP/IP. De acordo com a IETF (1981), uma arquitetura de rede é definida pelas camadas ou níveis que a compõem, pela interface entre essas camadas e pelas regras de comunicação entre camadas de mesmo nível em máquinas distintas, regras estas conhecidas como protocolo. O objetivo da divisão em camadas é permitir a modularização do *Software*, permitindo que as alterações sejam localizadas e transparentes aos outros níveis não afetados. Os módulos de *Software* de protocolo em cada máquina podem ser representados como camadas empilhadas. Cada camada cuida de uma parte do problema. Existem duas regras importantes para o entendimento da divisão do *Software* de rede em camadas:

- A camada inferior fornece serviços à camada superior.
- O protocolo de nível N no nó destino tem que receber o mesmo objeto enviado pelo protocolo de nível N no nó origem.

2.1.1.3 SMTP (*Simple Mail Transfer Protocol*)

O SMTP é o protocolo usado no processo de correio eletrônico na *Internet*. Um usuário, ao enviar um e-mail, solicita ao sistema de correio eletrônico que a entregue ao destinatário. Ao receber a mensagem do usuário, o sistema de correio eletrônico armazena uma

cópia em seu *Spool* (área do dispositivo de armazenamento), junto com o horário do envio e a identificação do remetente e do destinatário.

A transferência da mensagem é feita por um sistema em *background*, permitindo que o usuário remetente, após entregar a mensagem ao sistema de correio eletrônico, possa executar outras aplicações. O processo de transferência de mensagens, executando em *background*, mapeia o nome da máquina de destino em seu endereço IP, e tenta estabelecer uma conexão TCP com o servidor de correio eletrônico da máquina de destino. Note que a forma de transferência atua como cliente do servidor do *e-mail*.

Se a conexão for estabelecida com sucesso, o cliente envia uma cópia da mensagem para o servidor que guarda em seu *Spool*. Ao receber essa confirmação do recebimento e armazenamento, o cliente retira a cópia que mantinha em seu *Spool* local. Se a mensagem, por algum motivo qualquer, não for transmitida com sucesso, o cliente anota o horário da tentativa e suspende sua emissão. O cliente verifica se existem mensagens a serem enviadas na área de *Spool* e tenta transmiti-las. Se uma mensagem não tiver sido enviada por um período de tempo determinado, o serviço de correio eletrônico re-envia a mensagem ao remetente, avisando que não foi possível transmiti-la.

Normalmente o usuário se conecta à rede com seu sistema de correio eletrônico configurado para verificar se existem mensagens na caixa postal . Se existirem, o sistema de correio eletrônico emite um aviso para o usuário que, quando achar novamente conveniente ativará o módulo de interface para receber sua correspondência.

Segundo IETF (2001), uma mensagem SMTP divide-se em duas partes: cabeçalho e corpo, separados por uma lista em branco. No cabeçalho são especificadas as informações necessárias para a transferência da mensagem. O cabeçalho é composto por linhas, que contém uma palavra chave seguida de um valor. Por exemplo, identificação do remetente (palavra chave "To:" seguida do seu endereço), identificação do destinatário, assunto da mensagem propriamente dita. O formato do texto é livre e as mensagens são transferidas no formato texto. Os usuários do sistema de correio eletrônico são localizados através de um par de identificadores. Um deles especifica o nome da máquina de destino e o outro identifica a caixa postal do usuário. Um usuário pode emitir simultaneamente várias cópias de uma mensagem, para diferentes destinatários, usando o conceito de lista de distribuição (um nome que identifica um grupo de usuários). O formato dos endereços SMTP é: nome_local@nome_do_provedor, onde

nome_do_provedor identifica o domínio ao qual a máquina de destino pertence (esse endereço deve identificar um grupo de máquinas gerenciado por um servidor de correio eletrônico); o nome_local identifica a caixa postal do destinatário.

O SMTP especifica como o sistema de correio eletrônico transfere mensagens de uma máquina para outra. O módulo, interface com usuários e a forma como as mensagens são armazenadas, não são definidos pelo SMTP. O sistema de correio eletrônico pode também ser utilizado por processos de aplicação para transmitir mensagens contendo além de textos, qualquer tipo de arquivos, o que torna o SMTP um protocolo com grandes qualidades para a utilização com Serviços *Web*.

2.1.1.4 FTP (*File Tranfer Protocol*)

Segundo Postel (1985) os objetivos do FTP são:

1. Promover o compartilhamento de arquivos;
2. Encorajar o uso de computadores remotos;
3. Proteger um usuário de variações em sistemas de armazenamento de arquivos;
4. Transferir dados de forma rápida e eficiente;

O FTP é normalmente implementado por aplicações, sendo pouco utilizado por usuários simples (humanos).

Durante uma transmissão de dados em sessão de FTP é de responsabilidade do usuário enviar comandos e identificar destinatários para a transmissão, que é estabelecida como uma conexão TCP (*Tranfer Control Protocol*). Esses comandos podem ser executados por uma ferramenta conhecida por *Telnet*. Alguns dos comandos mais comuns definidos por Postel (1985) estão listados abaixo.

user - identifica o usuário que está tentando acessar um servidor de FTP. É obrigatório em muitos casos;

pass - senha do usuário, deve ser informado imediatamente após o comando user;

acct - utilizado para contas de usuário, alguns servidores podem adotar um processo de identificação por meio de contas ;

cwd - usado para mudar de diretório dentro de um servidor sem ter que se logar novamente;

cdup - comando para se mover para o diretório raiz com relação ao que se encontra o usuário

rein - comando para terminar uma sessão de usuário, terminando assim todas as atividades do usuário;

stor - acrônimo de *store* é usado para gravar um arquivo no servidor FTP;

allo - este comando pode ser requerido por alguns servidores para que seja, previamente, alocado um espaço de memória para a conexão em questão e para os arquivos desta sessão;

help - executa uma lista com as opções e como essas funcionam.

Embora, vários comandos foram mostrados, existem vários outros com finalidades diversas e além do mais, existem aplicações com interface gráfica com o usuário, que facilita bastante o trabalho, o próprio *Explorer* do *Windows* pode ser usado para acessar um *site* de FTP;

Após a definição de alguns dos principais protocolos envolvidos na arquitetura *Web* pode-se partir para o estudo dos Serviços *Web*. Dentro do capítulo sobre Serviços *Web*, Capítulo 3, temos mais uma sessão destinada a protocolos, sessão 3.1. Porém, sobre os protocolos envolvidos na arquitetura dos Serviços *Web*.

O que é nítido neste momento é que para qualquer trabalho envolvendo a *Internet* ou interoperabilidade de sistemas e sistemas distribuídos, um estudo sobre protocolos faz-se necessário.

O próximo capítulo é sobre Serviços *Web*. Serão abordados assuntos como arquitetura de Serviços *Web*, cenários comuns onde se pode utilizar Serviços *Web*, bem como os protocolos envolvidos na arquitetura.

3 SERVIÇOS *WEB*

Serviços *Web* em seu significado geral são serviços oferecidos por uma aplicação à outra através da *Internet* ou através de uma rede interna de uma empresa. Os clientes destes serviços podem agregá-los em suas aplicações finais habilitando transações de negócio ou criando novos Serviços *Web* (JENDROCK, 2003).

Muitos podem pensar, então, em se tratar de mais uma proposta de desenvolvimento de *Software* através de componentes distribuídos como CORBA ou COM+, porém não é bem esta a proposta de um Serviço *Web*. A principal característica dos Serviços *Web* é um pouco diferente, pois tem como objetivo principal o fornecimento de *Software* como serviço. Nesta concepção o termo Orientação a Serviços é utilizado a se referir à arquitetura de desenvolvimento de Serviços *Web*.

Segundo Hendricks (2002) em um cenário típico uma aplicação de negócio envia uma requisição para um serviço através de uma requisição SOAP (*Simple Object Access Protocol* – Protocolo de Acesso Simples a Objetos) sobre um protocolo de aplicação, podendo ser este HTTP, SMTP, FTP dentre outros. O serviço recebe essa requisição processa-a e retorna uma resposta. Normalmente o que se vê na *Internet* é que em sua maioria as respostas são feitas através de páginas de respostas, contendo os dados formatados e disponibilizados de forma amigável para os usuários. O conceito fundamental dos Serviços *Web* é simples – os Serviços *Web* nos permite compor as RPC (*Remote Procedure Call* - Chamada de Procedimentos Remotos) que permitirá a iteroperabilidade entre o serviço e seu cliente.

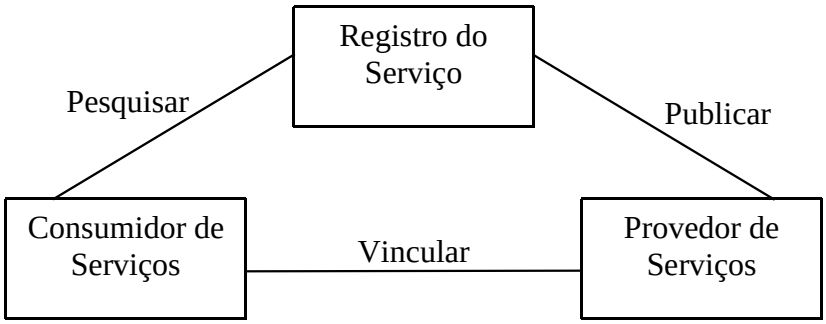


Figura 3 - Modelo comum de serviços *Web*
Fonte: HENDRICKS, 2002, p.10

Pode-se identificar, na Figura 3, três tipos de Papéis em um ambiente de Serviços *Web* típico, bem como as Operações que eles executam para fazer os Serviços *Web* funcionarem. Por Papéis, entende-se como os diferentes tipos de entidades; as Operações representam as funções executadas por essas entidades.

Os Papéis mostrados no diagrama são definidos segundo Hendricks (2002) da seguinte forma:

- **Provedores de Serviços** – O provedor de serviços é a entidade que cria o Serviço *Web*. Geralmente, o provedor de serviços apresenta alguma funcionalidade comercial em uma empresa como um Serviço *Web*, para que seja chamado por alguma outra empresa. Como exemplo: uma loja de locação de veículos on-line que deseja disponibilizar seu serviço locação on-line em forma de Serviços *Web*. O provedor de serviços precisa fazer duas coisas para ter acesso a todo o potencial em um Serviço *Web*. Primeiro precisa **descrever** o serviço *Web* em um formato padrão, que seja compreensível por qualquer empresa que possa usar este serviço. Em segundo lugar, para alcançar um grande público, o provedor de serviços deve **publicar** os detalhes sobre seu Serviço *Web* em um registro central que esteja publicamente disponível para todos os interessados.
- **Consumidor de Serviços** – Tratará como consumidor de serviços o "cliente" de serviços. O consumidor de serviços pode conhecer a funcionalidade de um Serviço *Web* a partir da descrição disponibilizada pelo Provedor de Serviços. Para recuperar os detalhes, o Consumidor de Serviços realiza uma **pesquisa** sobre o registro onde o Provedor de Serviços publicou sua descrição do Serviço *Web*, desta maneira o Consumidor de Serviços recupera através da descrição do Serviço *Web*, o mecanismo para **vínculo** com o mesmo podendo então chamar esse Serviço *Web*.

- **Registro do Serviço** – Um registro de serviço é a localização central onde o Provedor de Serviço pode relacionar seus serviços. Quando um Provedor de Serviços pretende disponibilizar seus Serviços *Web* para um público amplo o mesmo efetua um registro do serviço, desta forma os consumidores, potenciais, desse serviço podem facilmente encontrá-los. Tipicamente informações como detalhes sobre o Provedor de Serviços, Serviços *Web* por ele publicados e inclusive detalhes técnicos, são armazenados no Registro de Serviços.

Conforme apresentado acima, na Figura 3, é descrito um diagrama básico de um Serviço *Web* que citou além das entidades, operações que são fundamentais para o funcionamento dos Serviços *Web* – "localização", "vínculo" e "publicação". Uma comunicação entre aplicações, sem considerar a linguagem na qual foi desenvolvida ou para qual plataforma a aplicação foi planejada, precisa ser realizada. Para que isto seja possível há a necessidade dos padrões de cada uma dessas operações e de uma maneira padrão do Provedor de Serviços descrever seus Serviços *Web* independente da linguagem em que eles foram escritos. A seguir são apresentados os padrões, atualmente utilizados com os Serviços *Web*. Na Sessão 3.1.1, desse Capítulo informações detalhadas sobre os padrões e protocolos utilizados pelos Serviços *Web* podem ser encontradas.

3.1 Arquitetura dos Serviços *Web*

Para um entendimento sobre a arquitetura dos Serviços *Web* será apresentada uma "pilha básica de Serviços *Web*" definida por Hendricks (2002). Antes de descrever essa "pilha" é necessário que as operações apresentadas na seção anterior sejam relacionadas com padrões práticos e tecnologias utilizadas. Cada operação é relacionada com um padrão.

- No papel Provedor de Serviços se faz necessário uma forma padronizada para descrever os seus serviços. Esse padrão é conhecido com WSDL – *Web Services Description Language,* que utiliza o formato XML - *Extensible Markup Language*, para descrever Serviços *Web* em forma de um documento conhecido como

Data XML. Basicamente este documento descreve um conjunto de interfaces de um Serviço *Web* ou portas. Christensen (2001) define que em um documento WSDL tenha-se **mensagens** que são dados compartilhados pela rede e os ***"port types"*** que são os métodos ou operações descritas pelo Serviço *Web*. Os detalhes sobre o padrão WSDL podem ser encontrados na Sessão 3.1.1.2, desse capítulo.

- Analisando o diagrama da Figura 3 nota-se que duas operações incidem sobre o papel Registro de Serviços, publicar e pesquisar, estas são implementadas através do UDDI – *Universal Description, Discover and Integration*. A parte *Description* (descrição) é relacionada com a publicação realizada pelos Provedores de Serviços que requer que uma descrição detalhada de seus serviços sejam feitas para que a parte *Discover* (descoberta) possa estar disponíveis para os consumidores encontrarem serviços que, eventualmente, venham a atender suas necessidades. Detalhes sobre UDDI podem ser encontrados mais adiante, na Sessão 3.1.1.4.
- No que diz respeito ao vínculo um protocolo padrão de aplicações SOAP (*Simple Objetct Access Protocol*), um mecanismo superficial do XML usado para trocar informações entre aplicações, independente de sistema operacional, linguagem de programação ou modelo de objetos utilizados. Serão mostradas, mais adiante, neste capítulo, as características do SOAP.

De posse dessas informações pode-se então definir a "pilha básica de Serviços *Web*".

Tabela 2 - Pilha básica para Serviços *Web*

Conceitual **Protocolos**
Publicação / Descoberta de Serviços
UDDI
Descrição do Serviço
WSDL
Troca de mensagens XML
SOAP
Rede de Transporte
HTTP, FTP, SMTP, HTTPs sobre o protocolo TCP/IP

Fonte: HENDRICKS, 2002, p.11.

Na Tabela 2 está representado o conceitual (lado esquerdo) e o os protocolos (lado direito), práticos utilizados na arquitetura dos Serviços *Web*.

A camada mais baixa da pilha é a camada de **Rede de Transporte**, essa é a responsável pela disponibilização dos Serviços *Web* permitindo que estes estejam acessíveis

utilizando alguns protocolos de mercado já conhecido dos profissionais e usuários de *Internet* e de redes como o HTTP, FTP, SMTP dentre outros que fazem uso do protocolo TCP/IP.

A camada imediatamente superior **Troca de mensagens XML**, trata de definir um padrão para a troca de mensagens entre as aplicações. O protocolo SOAP é utilizado nesta camada para a troca de mensagens que é feita utilizando o padrão XML. A função básica desse protocolo é transmitir o XML via HTTP, ou um outro protocolo de necessidade / preferência, sem tentar definir um modelo de programação. Algumas literaturas mostram a arquitetura de Serviços *Web* de forma que seja possível entender que o protocolo SOAP seja vinculado ao protocolo de transporte, o que não é verdade. Segundo Hendricks (2002), o SOAP é independente do protocolo de transporte. É possível deduzir que o SOAP equivale a um XML. Detalhes da implementação do SOAP serão abordados na Sessão 3.1.1.3

Subindo a análise com relação à pilha, a próxima camada é a que define uma **Descrição dos Serviços *Web***. Aqui se encontra um documento XML que é o WSDL, esse documento fornece um mecanismo ao provedor de serviços para que seja possível descrever detalhes dos Serviços *Web* inclusive dados técnicos da funcionalidade do serviço. De acordo com Hendricks (2002) a WSDL é para os Serviços *Web* o que o CORBA IDL é para o CORBA ou a Microsoft MIDL é para o COM.

Bem falta saber agora como é possível encontrar ou publicar um Serviço *Web* para que este seja facilmente localizado por um potencial consumidor. Isso é feito pela camada do topo da pilha **Publicação e descoberta do Serviço** o propósito desta camada da pilha é permitir que, ao invés de disponibilizar os serviços individualmente para cada consumidor, eles possam ser agrupados em um registro central e desta forma ser “mais amplo”. Dentro da WSDL são descritos detalhes dos Serviços *Web* como as diferentes operações possíveis, tipos de dados, protocolos e vínculos. A especificação UDDI que fornece uma estrutura comercial usada para descrever um determinado negócio. Pode-se, para facilitar o entendimento, comparar um registro central com uma entidade certificadora onde atualmente são encontrados Certificados de Segurança. Estes Certificados são “expostos” de forma que para os clientes de determinadas empresas possam encontrar lá dados sobre a Política de segurança e detalhes dessas. É discutido em detalhe um modelo deste Registro Central na Sessão 3.1.1.4.

Abaixo é mostrada uma integração entre as camadas da pilha que foi demonstrada acima.

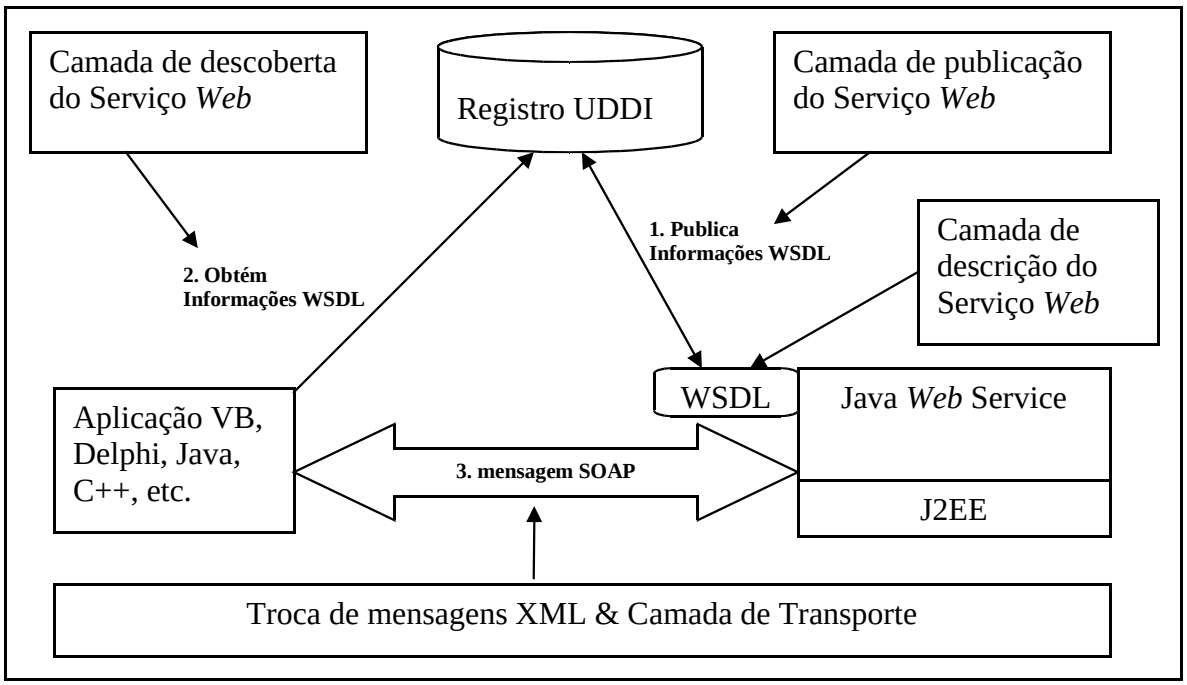


Figura 4 - Arquitetura para um Java *Web Service* (Seriço *Web* Java)
Fonte: HENDRICKS, 2002, p. 16.

Na Figura 4, três etapas são apresentadas:

1 – Publicar informações WSDL – o provedor de serviços tem o papel de codificar o documento WSDL, em primeira iniciativa. Para a codificação do documento WSDL o desenvolvedor tem várias alternativas para fazê-lo: Se o ambiente escolhido como servidor for o apache TOMCAT com o pacote para Serviços *Web* AXIS (*Apache Extensible Interaction System*), o próprio AXIS fornece uma ferramenta para a codificação da WSDL, com base no código Java. O desenvolvedor pode optar por escrever o documento usando um editor de XML, o que não é muito interessante, pois, poderia deixar algum detalhe técnico sem ser informado e que pode ser essencial para o serviço. Uma alternativa seria usar o wscompile, um aplicativo do pacote JWSDP – *Java Web Service Developer Package*.

Após a codificação é necessário publicar o serviço junto a Registro UDDI central;

2 – Obter informações WSDL – a aplicação do consumidor do Serviço *Web* pode descobrir os serviços que seja do seu interesse.

3 – Mensagens SOAP – O consumidor do serviço utiliza o documento WSDL para implementar o componente necessário para o envio de mensagens esperadas pelo serviço. A troca

de mensagens, como já visto, é feita pelos protocolos XML e SOAP pelo HTTP ou SMTP, FTP dentre outros.

3.1.1 Protocolos para Serviços *Web*

Na Sessão 2.1.1 foi abordado o assunto Protocolos para arquitetura *Web*. Nesta Sessão o assunto é, também, voltado para protocolos, porém, protocolos diretamente ligados com a arquitetura dos Serviços *Web*. Ao contrário da Sessão sobre protocolos para arquitetura *Web*, se faz necessário passar o máximo de detalhes sobre os protocolos aqui apresentados, porem sem o intuito de se fazer um "tutorial" sobre cada um deles e de esgotar o assunto por completo.

3.1.1.1 XML (*Extensible Markup Language*)

Esta sessão cobre os princípios básicos do XML. É o ponto de partida para iniciar a parte prática no desenvolvimento de Serviços *Web*. Algumas características do XML serão apresentadas de forma que se possa compreender o porque o XML é uma ótima opção para o armazenamento e interoperabilidade dentro do nosso cenário.

XML é uma linguagem de marcação baseada em texto que rapidamente se tornou padrão para o compartilhamento de dados através da *Internet*, assim como em HTML os dados são identificados pelo uso de "*tags*" (identificadores que aparecem em um texto XML entre < "menor" e > "maior"), estas *tags*, em conjuntos, são chamadas de marcação, o que dá origem ao M (*Markup*) da sigla XML e HTML. A principal diferença entre XML e HTML é que as *tags* do XML são utilizadas para identificar dados e no HTML são comumente utilizadas para formatação de texto, basicamente. O Exemplo 3.1, abaixo, ilustra um código XML simples de ser compreendido.

Exemplo 3. 1 - Código Básico XML

```
<?xml version=”1.0”?>
<mensagem>
<para>destinatario@provedor.com.br</para>
      <de>remetente@provedor.com.br</de>
      <objetivo> Exemplo de código XML</objetivo>
      <texto>
            Deixe-me mostrar o quanto o XML pode ser
            útil em Serviços Web.
      </texto>
</mensagem>
```

No Exemplo 3.1 é demonstrado um possível XML para transmissão de dados como é feito pelo e-mail. Note que a primeira linha presente no exemplo é o que a Microsoft (2001) define como mecanismo *Prolog* que identificará para seus interpretadores que se trata de um arquivo XML. Nas demais linhas do exemplo uma “Mensagem” é definida com as características necessárias como: destinatário, remetente, título e conteúdo da mensagem, desta forma um interpretador XML pode seguir um padrão de execução e formatar propriedades dentro de um código de programação com o objetivo de se enviar um e-mail, por exemplo.

3.1.1.1.1 *Tags* e Atributos

As *tags* do XML podem, também, conter atributos, ou seja, informações adicionais incluída como parte das *tags*. O exemplo abaixo é uma adaptação do exemplo anterior definindo algumas *tags* como sendo atributos.

Exemplo 3. 2 - Uso de atributos em XML

```
<?xml version=”1.0”?>
<mensagem
      para=”destinatário@provedor.com.br”
      de=”remetente@provedor.com”
```

```
      objetivo=" Exemplo de usando atributos">
      <texto>
            O código desta forma fica mais compacto e
            em certos casos fazem mais sentido.
      </texto>
</mensagem>
```

A sintaxe para se definir um atributo é similar à forma adotada em HTML o nome do atributo é seguido por um sinal de igualdade (=) e o conteúdo é colocado entre aspas duplas. Uma característica muito importante do XML é a simplicidade de se mapear os dados em classes de dados dentro dos programas que os executam.

Onde o XML pode ser utilizado? Bem, atualmente a XML pode ser usada, basicamente, em qualquer desenvolvimento que requer o compartilhamento de informações.

São listadas abaixo algumas das possibilidades de uso do XML:

- Processamento de dados onde um programa preparado para interpretar XML possa ser utilizado para recuperar tais dados;
- Documentos específicos para definição de interfaces. Muito utilizado no uso de Serviços *Web*;
- *Arqchiving:* Um processamento de *backup* de documentos onde se faz necessário o armazenamento da versão do documento, por exemplo.

Em XML é muito comum à adoção de padronização para nomes de *tags* e definição de tipos de dados, essa padronização é realizada por um arquivo também codificado em XML que denominamos DTD (*Data Type Definition*) quando dizemos que um determinado tipo de dados está definido em DTD então fornecemos um *namespace* (uma espécie de identificador único para uma *tag* dentro de um documento XML). Um exemplo da definição de um *namespace* é apresentado abaixo:

Exemplo 3. 3 - Uso de *namespace* por DTD em XML

```
<ir:taxa xmlns:ir="http://localhost:8080/WEBCALC/ir">
      15.00
</ir:taxa>
<inss:taxa xmlns:inss="http://localhost:8080/WEBCALC/inss">
      8.00
</inss:taxa>
```

O *namespace* é muito útil dentro do XML, pois possibilita padronizar um documento com base em outro e evita assim um conflito entre nomes das *tags.*

Durante todo o texto, a partir de agora, você irá encontrar diversos trechos que utilizam XML para descrever seus funcionamentos. O XML é o ponto chave para o perfeito funcionamento dos Serviços *Web*, pois todos os padrões envolvidos são baseados na XML. Algumas modificações com relação ao Exemplo 3.1 serão apresentadas à medida que o assunto é avançando com os protocolos envolvidos. Por hora será considerado que o XML é como uma forma de estruturar dados.

3.1.1.2 WSDL (*Web Services Description Language*)

O Protocolo WSDL é para os Serviços *Web* o que o IDL (*Interface Definition Language*) é para o CORBA e COM, ou seja, é um descritor da interface externa do Serviço *Web*, porém o WSDL vai um pouco mais além, pois fornece também a localização ou a extremidade do serviço. O principal objetivo do WSDL é especificar um formato XML que descreve os serviços externos, ou interfaces, de uma aplicação, nesse caso os Serviços *Web*, isso se dá, pois, como já visto Serviços *Web* tem o objetivo de possibilitar chamadas de seus métodos de forma remota e de forma que independa da linguagem que está sendo usada.

Implementado com base em XML a WSDL descreve tecnicamente detalhes do Serviço *Web* de forma que o Consumidor do Serviço não encontre dificuldades em executá-lo de forma correta. Esta Sessão trata da especificação de um documento WSDL que é essencial para o funcionamento da arquitetura utilizada para Serviços *Web*, de forma a fornecer detalhes teóricos e técnicos sobre como se implementar um arquivo WSDL, bem como alguns exemplos de ferramentas CASE (*Computer Aided System Engeneering*) que os implementam de forma simples e automática.

3.1.1.2.1 Estrutura de um documento WSDL

A estrutura apresentada nessa sessão é baseada em uma proposta padronizada pelo W3C (*World Wide Web Consortium*) definida por Christensen (2001) e será demonstrada de forma que os interessados possam escrever um arquivo WSDL de forma simples e clara.

Para iniciar com a especificação da estrutura, faz-se necessário conhecer o que pode ser tratado pelos documentos WSDL, como por exemplo, tipos de dados, parâmetros de entrada e de saída, relacionamento entre parâmetros de entrada e de saída, como são definidos as operações e os Tipos de Porta[1] e os protocolos a serem usados para acessar os métodos de um objeto.

Tipos de dados(Tipos): Os tipos de dados aceitos pelo WSDL são descritos por Paul (2001) e aqui serão apresentados alguns dos tipos primitivos de dados aceitos que são:

string (conjunto de caracteres alfanuméricos);

boolean (valores lógicos *true/false*);

decimal (número real com precisão arbitrária, por exemplo: -1.23; 1235.4);

float (tipo de dados com ponto flutuante com uma precisão de 32 bits – representado por: m x 2^c, onde m corresponde a um inteiro cujo valor absoluto é 2^24 e c é um inteiro entre -149 e 104 – exemplos: -1E6, 4.12, 5.32e-2);

double (semelhante ao float, porém com precisão dupla de 64 bits – representado por m x 2^c, onde m corresponde a um inteiro cujo valor absoluto é 2^53 e c é um inteiro entre -1075 e 970 – exemplo: -1E4, 12.78e-2);

duration (tipo de dados que representa a variação de tempo representado por seis coordenadas que representam: ano, mês, dia, horas, minutos e segundos, respectivamente – exemplo: P1Y2M3DT10H30M que representa 1 ano, 2 meses, 3 dias, 10 horas e 30 minutos);

dateTime (tipo de dados que corresponde a um tempo específico como por exemplo 1999-05-31T13:20:00 – formato UTC (*Universal Time Cordinate*) que indica 13:20 de 31 de maio de 1999);

time (tipo de dados para representar um dada hora no formato hh:mm:ss. Exemplo: 05:43:00);

[1] Entenda por Tipo de Porta um conjunto de operações ou métodos que o Serviço *Web* oferece.

date (tipo de dados para representar datas no formato yyyy-mm-dd, exemplo: 1999-09-28);

anyURI (tipo de dados utilizado para descrever uma URI (*Universal Resource Identifier*) podendo ser absoluto ou relativo, exemplo: ../imagens/WEBCALC.gif);

QName (*Qualified Name* – representa nomes qualificados de um documento XML é representado por duas partes: *namespace, local part* onde namespace é do tipo anyURI e *local part* é do tipo string. Este tipo de dados é um pouco mais complexo pois se refere ao reconhecimento de sintaxe do XML é muito usado por interpretadores XML);

Os tipos de dados para o padrão WSDL são os mesmos usados e aceitos pelo XML uma vez que WSDL é baseado em XML. Outros tipos são definidos pelo W3C e não foram abordados aqui por achar não pertinentes ao entendimento do uso da WSDL, maiores detalhes sobre os tipos de dados em Paul (2001). Uma ressalva sobre os tipos apresentados acima é que estes são primitivos, porém, o XML aceita também tipo de dados definidos pelo usuário, em Java pode-se notar este tratamento quando uma variável, que representa um objeto criado para fins da aplicação, é utilizada. Porém não é do escopo deste trabalho tratar tipo de dados aceitos em XML.

Mensagens: entende-se por mensagens todo o parâmetro de chamada de um serviço podendo ser ele de entrada ou de saída. Todo o interfaciamento é feito por troca de mensagens.

Operações: O relacionamento entre os parâmetros de entrada e saída é denominado de operações. Essas nada mais são que a assinatura de um método de um determinado objeto.

Tipos de Porta: O Agrupamento lógico de operações que podem constituir métodos de um objeto é denominado Tipo de Porta.

Protocolos: Para acessar os métodos de um objeto são necessários alguns protocolos. As únicas opções para se fazer isto são SOAP, HTTP e MIME (*Multpupose Internet Mail Extensions*) este último é conhecido por vínculo de objetos (HENDRICKS, 2002).

Após essas definições pode-se mostrar a estrutura de um documento WSDL e uma forma que esse poderá ser escrito. A Figura 5 mostra um framework para um documento WSDL, cada um dos componentes será explicado e detalhado mais adiante.

```
<definitions>
      <types>
      [esquma XML que descreve a utilização
      de tipos de dados]
      </types>
      <message>
      [descrição das mensagens de entrada e
      saída]
      </message>
      <portType>
      <operation>
                  [referencia para cada mensagem
                  de entrada e de saída]
            </operation>
      </portType>
      < binding >
      [descrição dos protocolos de rede para
            invocação]
      </ binding >
      <service>
      [referencia para a localização do
      serviço]
      </service>
</definitions>
```

Figura 5 - Estrutura básica para documentos WSDL

Fonte: HENDRICKS, 2002 p.103.

Componente definitions: o elemento *definitions* tem como conteúdo básico os *namespaces* usados no documento WSDL. Ele pode assumir, ainda alguns atributos como *name* que pode ser usado para indicar, dentro do WSDL, o nome do documento. O atributo *targetNamespace* que recebe com parâmetro um tipo anyURI para indicar o caminho utilizado para este documento WSDL.

Componente *types*: o conteúdo das mensagens tratadas por um Serviço *Web* pode utilizar-se de tipos de dados. Estes tipos são baseados em um modelo de dados específico podendo usar um esquema para defini-los o elemento *types* é usado, então, para definir os tipos de dados usados. Programadores estão acostumados com a declaração de tipos de dados, e utilizam em grande escala, então nada mais justo que na estrutura do WSDL tenhamos uma área específica para os tipos. Segundo Deitel (2001), em Java utiliza-se tipos de dados e objetos para manipular as informações de uma aplicação, em um sistema distribuído normalmente estes objetos são transportados por uma rede em formato binário, são **serializados** e somente assim são transportados. Nesse cenário ambos os lados são capazes de serializar e desserializar estes

objetos. Temos para Serviços *Web* um conceito semelhante, porém a serialização dos dados é feita via XML de forma que os serviços se tornem independentes de plataforma ou linguagens. Esse elemento tem a finalidade de informar para o cliente Consumidor do Serviço os tipos de dados usados pelo mesmo.

Componente *message*: O componente *message* associa um tipo definido com a mensagem compartilhada em uma determinada operação, mais adiante no Exemplo 3.4 você entenderá melhor o que isso quer dizer e como é simples. Por hora pense na mensagem como uma assinatura, detalhada, de um dado método de um determinado objeto.

Componente *portType*: "O Componente *portType* é o que define um conjunto lógico das operações de um serviço, neste componente o objetivo é obter as mensagens conforme declaradas pelos componentes <*message*> em uma seqüência útil" (HENDRICKS, 2002. p.102). Neste ponto uma questão interessante surge: quando as pesquisas sobre Serviços *Web*, para este Trabalho de Conclusão de Curso, se iniciaram várias informações, teóricas e técnicas foram adquiridas, algumas delas foram difíceis de entender seu funcionamento. Esse componente, por exemplo, é um deles. Porém, no início das pesquisas, a versão da especificação do WSDL era a 1.1 que será adotada neste trabalho, porém uma nova especificação já está sendo padronizada pelo W3C que é a versão 2.0, e nessa temos uma melhor definição para *portType* que foi substituída por ***Interface***, sendo mais fácil o seu entendimento. Os detalhes técnicos são os mesmos, a finalidade é a mesma, o que se deve frisar é que uma mudança de termo pode muitas vezes nos ajudar a compreender melhor, determinados assuntos. O Componente *portType* como dito anteriormente agrupa de forma lógica um conjunto de operações que nada mais são que elementos do componente *portType* que são referenciados dentro da WSDL como <*operation*>. Existem quatro tipos diferentes de operações definidas por Hendricks (2002):

- **Operação unidirecional**: define uma mensagem que é enviada do Consumidor para um serviço, sem que ocorra nenhuma resposta desse serviço.
- **Solicitação/Resposta**: foi dito no Capítulo 2 que é muito comum em um cenário baseado na *Web* que um cliente solicite um serviço e obtenha uma resposta de um provedor da *Internet*, neste ponto temos esta funcionalidade aplicada nos Serviços *Web*, onde um Consumidor envia uma solicitação de serviço a um Serviço *Web* qualquer e este então retorna com uma resposta

a solicitação efetuada. Pode-se perceber com esse tipo de mensagem o funcionamento das RPC's (*Remote Procedure Call*). Estes são os tipos mais comuns de operações.

- **Pedido/Resposta**: essa não é uma forma muito comum de utilização das operações, pois, ocorre ao contrário do que freqüentemente se nota, uma vez que o Serviço *Web* envia um pedido para o Consumidor o que resulta em uma mensagem que é enviada do Consumidor para o serviço.
- **Notificação**: esse tipo de operação é comum e ocorre do Serviço *Web* para o Consumidor podendo, por exemplo, ser uma confirmação a uma assinatura de um serviço.

Componente *binding*: O Componente *binding* é o que define de forma concreta um formato de mensagem e protocolos que podem ser usados para definir os *endpoints* (interfaces disponibilizadas pelo Serviço *Web*). Esse componente define detalhes da implementação necessários para acessar o serviço. É a parte que cuida, por exemplo, dos protocolos usados para acessar os métodos de um determinado objeto podendo ser SOAP, HTTP ou MIME conforme dito anteriormente.

Componente Service: Descreve um conjunto de *endpoints* assim como o nome e caminho do Serviço *Web*, é a parte final da estrutura que até então definiu detalhes das interfaces e como executá-las neste componente é onde se encontram as informações necessárias para identificar para onde o Consumidor deverá encaminhar uma solicitação.

Para esclarecer os detalhes acima apresentados um exemplo composto por duas partes: Exemplo 3.4 a parte que mostra um documento WSDL gerado pelo AXIS que descreve um Serviço *Web* chamado CalculadoraService que implementa as funções de soma, subtração multiplicação e divisão de uma classe Java chamada Calculadora; a segunda parte Exemplo 3.5 mostrará o código Java correspondente a classe que deu origem ao arquivo WSDL. A primeira parte do exemplo será explicada de forma interativa onde será apresentado um trecho do exemplo procedido de uma explicação sumária.

Exemplo 3. 4 - Documento WSDL "CalculadoraService"

```
<?xml version="1.0" encoding="ISO-8859-1"?>
      <wsdl:definitions
```

```
targetNamespace=
"http://localhost:8080/axis/Calculadora.jws"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://localhost:8080/axis/Calculadora.jws"
xmlns:intf="http://localhost:8080/axis/Calculadora.jws"
xmlns:soapenc=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://www.w3.org/1999/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

O trecho acima define os *namespaces* referente aos padrões que podem ser tratados dentro da WSDL, pode-se notar que toda *tag* precedida por “wsdl:” terá seu padrão definido pelo esquema XML http://schemas.xmlsoap.org/wsdl/. Tendo isso definido pode-se então descrever como será a troca de mensagens especificando cada método do código Java, que será mostrado na segunda parte deste exemplo.

```
<wsdl:message name="multiplicarResponse">
      <wsdl:part name="multiplicarReturn"
                  type="xsd:int" />
</wsdl:message>
<wsdl:message name="somarResponse">
      <wsdl:part name="somarReturn" type="xsd:int" />
</wsdl:message>
<wsdl:message name="subtrairRequest">
      <wsdl:part name="numA" type="xsd:int" />
      <wsdl:part name="numB" type="xsd:int" />
      </wsdl:message>
<wsdl:message name="somarRequest">
      <wsdl:part name="numA" type="xsd:int" />
      <wsdl:part name="numB" type="xsd:int" />
</wsdl:message>
<wsdl:message name="subtrairResponse">
      <wsdl:part name="subtrairReturn" type="xsd:int" />
</wsdl:message>
<wsdl:message name="dividirRequest">
```

```
      <wsdl:part name="numA" type="xsd:int" />
      <wsdl:part name="numB" type="xsd:int" />
</wsdl:message>
<wsdl:message name="multiplicarRequest">
      <wsdl:part name="numA" type="xsd:int" />
      <wsdl:part name="numB" type="xsd:int" />
</wsdl:message>
<wsdl:message name="dividirResponse">
      <wsdl:part name="dividirReturn" type="xsd:int" />
</wsdl:message>
```

No trecho acima são definidas as mensagens de requisição e resposta a uma solicitação. Para cada uma dessas têm-se a definição dos tipos de entrada e saída. O método *somar()* por exemplo, recebe dois inteiros e possui um retorno também do tipo inteiro. Isto é caracterizado pelas mensagens *somarRequest* e *somarResponse.*

```
<wsdl:portType name="Calculadora">
      <wsdl:operation name="dividir"
                  parameterOrder="numA numB">
            <wsdl:input message="impl:dividirRequest"
            name="dividirRequest" />
            <wsdl:output message="impl:dividirResponse"
            name="dividirResponse" />
      </wsdl:operation>
      <wsdl:operation name="multiplicar"
                  parameterOrder="numA numB">
            <wsdl:input message=
                  "impl:multiplicarRequest"
            name="multiplicarRequest" />
            <wsdl:output message=
                  "impl:multiplicarResponse"
            name="multiplicarResponse" />
      </wsdl:operation>
      <wsdl:operation name="somar"
                  parameterOrder="numA numB">
            <wsdl:input message="impl:somarRequest"
            name="somarRequest" />
```

```
                <wsdl:output message="impl:somarResponse"
                name="somarResponse" />
            </wsdl:operation>
            <wsdl:operation name="subtrair"
                        parameterOrder="numA numB">
                <wsdl:input message="impl:subtrairRequest"
                name="subtrairRequest" />
                <wsdl:output message="impl:subtrairResponse"
                name="subtrairResponse" />
            </wsdl:operation>
        </wsdl:portType>
```

Como mencionado na definição da estrutura, um *portType* define um agrupamento lógico das operações ou métodos que estarão disponíveis pelo serviço. Aqui o nível de detalhamento é um pouco maior do que o mostrado nas definições das mensagens. Pode-se notar que neste ponto é definida até a ordem em que são esperados os parâmetros.

```
            <wsdl:binding name="CalculadoraSoapBinding"
            type="impl:Calculadora">
                <wsdlsoap:binding style="rpc"
                transport="http://schemas.xmlsoap.org/soap/http"
/>
                <wsdl:operation name="dividir">
                    <wsdlsoap:operation soapAction="" />
                    <wsdl:input name="dividirRequest">
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/"
                        namespace="http://DefaultNamespace"
use="encoded" />
                    </wsdl:input>
                    <wsdl:output name="dividirResponse">
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/" namespace=
                        "http://localhost:8080/axis/Calculad
ora.jws" use="encoded"/>
```

```
            </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="multiplicar">
            <wsdlsoap:operation soapAction="" />
            <wsdl:input name="multiplicarRequest">
                  <wsdlsoap:body
                  encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/"
namespace="http://defaultnamespace" use="encoded" />
            </wsdl:input>
            <wsdl:output name="multiplicarResponse">
                  <wsdlsoap:body
                  encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/"
namespace="http://localhost:8080/axis/Calculadora.jws"
use="encoded" />
            </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="somar">
            <wsdlsoap:operation soapAction="" />
            <wsdl:input name="somarRequest">
                  <wsdlsoap:body
                  encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/"
                  namespace="http://DefaultNamespace"
use="encoded" />
            </wsdl:input>
            <wsdl:output name="somarResponse">
                  <wsdlsoap:body
                  encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/"
namespace="http://localhost:8080/axis/Calculadora.jws"
use="encoded" />
            </wsdl:output>
      </wsdl:operation>
      <wsdl:operation name="subtrair">
            <wsdlsoap:operation soapAction="" />
            <wsdl:input name="subtrairRequest">
```

```
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/"
                        namespace="http://DefaultNamespace"
use="encoded" />
                  </wsdl:input>
                  <wsdl:output name="subtrairResponse">
                        <wsdlsoap:body
                        encodingStyle="http://schemas.xmlsoa
p.org/soap/encoding/"
                        namespace="http://localhost:8080/axi
s/Calculadora.jws" use="encoded" />
                  </wsdl:output>
            </wsdl:operation>
      </wsdl:binding>
```

O que o componente *biding* está acrescentando à nossa descrição é qual protocolo será usado para executar as operações dos nossos objetos reais, neste caso a WSDL foi definida usando um mecanismo de RPC que funcionará sobre o protocolo SOAP que será, esse, detalhado na Sessão 3.1.1.3. Para finalizar a descrição da WSDL fornece o nome do Serviço *Web* e sua localização para que o Consumidor possa facilmente referenciá-lo.

```
      <wsdl:service name="CalculadoraService">
            <wsdl:port binding="impl:CalculadoraSoapBinding"
                        name="Calculadora">
                  <wsdlsoap:address location=
                  "http://localhost:8080/axis/Calculadora.jws"
                  />
            </wsdl:port>
      </wsdl:service>
</wsdl:definitions>
```

O Consumidor para o serviço CalculadoraService poderá, se quiser, montar um *stub*, ou seja, um esqueleto que automaticamente pode ser gerado utilizando o pacote WSDL2J que acompanha o kit de desenvolvimento JWSDP (*Java Web Service Development Pack*), representando as assinaturas dos métodos que poderão ser implementados dentro da aplicação

cliente. Em outras palavras cria uma interface baseada nas definições da WSDL. Na segunda parte do exemplo, conforme já foi citado, será apresentada a classe Java que deu origem ao documento WSDL apresentado no Exemplo 3.4 acima.

Exemplo 3. 5 - Classe Java "Calculadora"

```
public class Calculadora {
	public int somar (int numA, int numB){
		return numA + numB;
	}
	public int subtrair (int numA, int numB){
		return numA - numB;
	}
	public int multiplicar (int numA, int numB){
		return numA * numB;
	}
	public int dividir (int numA, int numB){
		if (numB != 0)
			return numA / numB;
		return 0;
	}
}
```

A classe apresentada no Exemplo 3.5, acima, é uma classe simples composta por quatro métodos que efetua uma das quatro operações matemáticas com base em dois argumentos (`numA e numB`).

É possível perceber que falar sobre a WSDL constituiria um trabalho a parte, como o objetivo desta Sessão é prestar alguns esclarecimentos necessários para se implementar um Serviço *Web*, utilizando a tecnologia Java ter, então, uma base para escrever os primeiros Serviços *Web*, porém, após a codificação de um serviço deve-se ter em mente onde e como torná-lo disponível para um possível Consumidor.

O propósito maior foi o de iniciar com a montagem da arquitetura apresentada na Sessão 3.1 que deverá ser implementada por qualquer um que queira desenvolver um Serviço *Web*.

As Sessões seguintes trazem informações sobre o protocolo SOAP que mostrará detalhes do protocolo que trata da troca de mensagens que é utilizado pelos Serviços *Web* e o protocolo UDDI que descreverá como são feitas a publicação e descoberta de um serviço pela *Web*.

3.1.1.3 SOAP (*Simple Object Access Protocol*)

Definido por Gudgin (2004) como sendo uma forma de prover uma definição para informações baseado no padrão XML que pode ser usado para o compartilhamento entre *hosts* em sistema distribuído e centralizado, o SOAP é um protocolo de computação superficial que não está vinculado a nenhuma plataforma de *Hardware* ou *Software*. A especificação do SOAP define um *framework* para a troca de mensagens em um ambiente distribuído, além de convenções para a transmissão de chamadas e respostas de procedimentos remotos (RPC). O SOAP se difere de outros protocolos utilizados em ambientes distribuídos como CORBA, o RMI e o DCOM, por usar o protocolo XML para se comunicar e não o formato binário. O SOAP não é o primeiro protocolo de computação distribuído, contudo é o primeiro a receber um grande apoio, por parte da indústria. Isso torna o SOAP uma opção vantajosa para se utilizar em um ambiente desta natureza. O SOAP está em sua versão 1.2.

A especificação SOAP consiste em duas partes: encapsulamento de mensagens e de RPC (*Remote Procedura Call*)

Para que se possa iniciar com as definições sobre esse protocolo é interessante mostrar como é o formato da mensagem SOAP, este formato é apresentado na Figura 6.

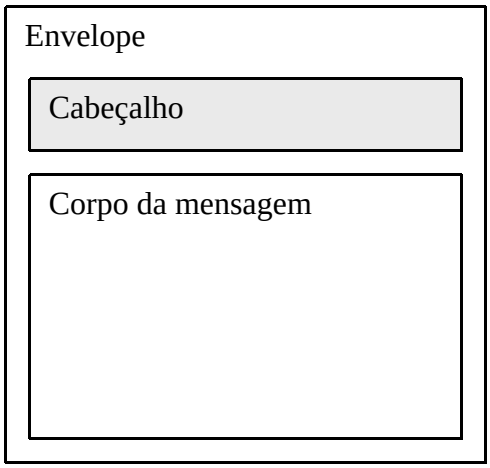


Figura 6 - Representação gráfica de uma mensagem SOAP
Fonte: HENDRICKS, 2002, p. 43.

O formato da mensagem é bem simples. Um *framework* é definido como Envelope que possui os componentes Cabeçalho e Corpo da Mensagem. O Cabeçalho é opcional, e o Corpo da Mensagem constitui a carga útil a ser transportada. O SOAP é considerado protocolo superficial, pois contém menos recursos que outros protocolos de computação distribuída, o que torna o protocolo menos complexo.

A parte da especificação relativa ao encapsulamento da RPC permite que o SOAP forneça uma abstração na camada do protocolo. Esta abstração fornece sistemas remotos com acesso a objetos através de plataformas, sistemas operacionais e ambientes que seriam, de outra maneira, incompatíveis.

"É possível uma camada SOAP entre objetos Java e CORBA. Essa camada cria um sistema distribuído comum aos dois objetos, sem que seja necessário se preocupar com a peculiaridade de cada protocolo de computação distribuída" (HENDRICKS, 2002, p.35).

3.1.1.3.1 Vantagens do SOAP

O SOAP Oferece várias vantagens em relação a outros protocolos de sistemas distribuídos, entre elas:

O SOAP pode atravessar *firewalls*, com facilidade - devido a sua capacidade de poder ser utilizado sobre o protocolo HTTP[2];

Os dados do SOAP são estruturados usando XML;

O SOAP pode ser usado, potencialmente, em combinação com vários protocolos de transporte, como HTTP, SMTP e JMS;

O SOAP mapeia satisfatoriamente para o padrão de solicitação/resposta do HTTP

O SOAP é razoavelmente superficial como um protocolo;

Existe suporte para SOAP, por parte de vários fornecedores, incluindo Microsoft, IBM e SUN;

Dentre as vantagens apresentadas acima uma delas pode também ser considerada como desvantagem. A capacidade de transpor os *firewalls* com facilidade o que pode ser um problema em questão de segurança, onde aplicações não autorizadas poderiam acessar uma aplicação que fizesse uso do SOAP.

3.1.1.3.2 Desvantagens do SOAP

Como visto anteriormente o SOAP oferece muitas vantagens, porém para contra-balancear ele também possui muitas desvantagens, no entanto essas desvantagens tendem a ser corrigidas e os pontos onde o SOAP ainda peca, aperfeiçoados.

Apesar de o SOAP ter um amplo suporte, ainda existem problemas de incompatibilidades entre diferentes implementações do SOAP.

Os mecanismos de segurança ainda são imaturos, o SOAP não define um mecanismo para autenticação de uma mensagem antes que ela seja processada. Também não define um mecanismo para a criptografia do conteúdo de uma mensagem SOAP.

O SOAP é um protocolo sem confirmação, ou seja, não há uma confirmação de que a mensagem foi entregue;

[2] Esta vantagem está diretamente ligada com o fato de que os *firewalls*, apesar de poderem ser configurados para bloqueio, não conhecem previamente a origem de uma possível tentativa de acesso, deixando assim que o mesmo seja feito por meio do protocolo HTTP.

As desvantagens apresentadas acima podem ser resolvidas utilizando outros protocolos como, por exemplo, o HTTPs para ajudar com relação à segurança; na questão da criptografia podemos utilizar mecanismos já existentes para criptografar o XML. No que diz respeito à confirmação talvez poderia ser feita através do TCP/IP que é um protocolo que possui confirmação e alguns outros aspectos que podem ser auxiliados por outras tecnologias.

Os assuntos sobre segurança são abrangidos em maiores detalhes no Capítulo 4.

3.1.1.3.3 Estrutura do SOAP

Apresentadas algumas das vantagens e desvantagens do SOAP pode-se então verificar com o que se parece o SOAP, a Figura 7 abaixo mostra isso.

```
<SOAP-ENV: Envelope xmlns: SOAP-ENV=
      “http://schemas.xmlsoap.org/soap/envelope”
      <SOAP-ENV:encodingStyle=
      ”http://schema.xmlsoap.org/soap/encoding”>
      <SOAP-ENV: Header>
      <!-- Opcional -->
      </SOAP-ENV: Header>
      <SOAP-ENV: Body>
      // Chamadas ou respostas RPC
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 7 - Exemplo da estrutura do SOAP
Fonte: HENDRICKS, 2002, p. 38.

Envelope SOAP: o Envelope SOAP é o elemento de nível mais elevado de uma mensagem SOAP. O envelope SOAP contém um cabeçalho, como dito anteriormente, opcional e um Corpo obrigatório. Como visto na Figura 7 a *tag* que o identifica é a `<Envelope>`. Observem que todas as *tags* possuem o pré-fixo `SOAP-ENV` que é padronizado no esquema XML, isso é feito para que os nomes das *tags*, específicas do SOAP, não se confundam com outras possíveis *tags*, conforme mencionado na Sessão 3.1.1.1 sobre o protocolo XML.

Cabeçalho SOAP: o cabeçalho SOAP é um mecanismo que permite a transmissão de informação dentro da mensagem que não tenha uma ligação técnica direta com a aplicação, esse componente não é obrigatório. Se for usado, o componente *Header* pode conter alguns atributos como, por exemplo: *actor* e *mustUnderstand* que podem respectivamente direcionar um item do cabeçalho para um destinatário específico e indicar se um entrada de cabeçalho é obrigatória ou não. O atributo *actor* recebe um URI que identifica o seu destinatário já o atributo *mustUnderstand* recebem os valores 1 ou 0, o valor 1 indicando que o receptor do cabeçalho deve saber processar o elemento, caso contrário o processamento da mensagem falhará

Corpo SOAP: no corpo SOAP é que se encontra toda a informação que deve ser recebida pelo receptor da mensagem. Na prática a maioria dos conteúdos consiste tipicamente em chamadas e repostas RPC e relatório de erro.

O SOAP também possui um elemento extra que para o tratamento de exceções, que podem ocorrer durante a troca de mensagens, este elemento é o `<Fault>`, esse só pode ocorrer uma única vez dentro de uma mensagem SOAP. Os sub-elementos `<faultcode>`, `<faultstring>`, `<faultactor>` e `<detail>` são utilizados para melhor descrever a exceção, porém somente `<faultcode>` é obrigatório e foi criado para prover uma maneira fácil de identificar uma falha. Em caso de ser usado um tratamento *Fault*, esse deve estar compreendido dentro do Corpo da mensagem, ou seja, entre as *tags* `<Body>` e `</Body>`.

Como dito anteriormente o SOAP pode utilizar vários protocolos de transporte dentre eles o HTTP que é mais amplamente usado. Segundo Gudgin (2004) o SOAP tem uma flexibilidade em se adaptar ao modelo de requisição/resposta do HTTP o que permite que uma solicitação SOAP seja transportada como uma solicitação HTTP e que uma resposta SOAP podem ser transportadas como uma resposta HTTP. A isto damos o nome de vínculo.

Até este momento já se sabe como codificar um documento WSDL e detalhes sobre o funcionamento do SOAP que foram baseadas em sua versão 1.1, porém, vale frisar que já está sendo implementada a versão 1.2.

No próximo capítulo será mostrado como é possível publicar um Serviço *Web* na *Internet* de forma que possa ser encontrado facilmente por qualquer interessado, serão apresentados detalhes de como funciona o UDDI (*Universal Description, Discovery and Integration*) ou Registro Central de Serviços *Web*.

3.1.1.4 UDDI (*Universal Description, Discovery and Integration*)

Bryan (2002), define que UDDI é o nome de um grupo de registros baseado na *Web* que tem por finalidade expor informações sobre um negócio de um determinada empresa e interfaces técnicas ou (API's).

O termo UDDI é usado para representar um servidor de registros para os nossos Serviços *Web*, é também conhecido com diretório de serviços. Qualquer pessoa que queira disponibilizar seus Serviços *Web* deverá registrá-lo em diretório UDDI. É considerado o "ponto chave" para a tecnologia dos Serviços *Web*, pois é através dele que se pode registrar e descobrir os Serviços *Web*. É um padrão também baseado em XML, o que permite uma interoperabilidade muito maior, sendo possível utilizar-se deste mecanismo por qualquer linguagem ou plataforma. No decorrer desse capítulo serão abordados alguns dos tópicos relacionado com o registro e a descoberta de Serviços *Web*, por meio do UDDI, neste ponto pode-se adotar duas estratégias para abordar o assunto, sendo: 1- características essenciais para que se possa trabalhar com o diretório de serviços ou 2 - detalhamento técnico utilizado para a implementação de um sistema de registro UDDI.

Uma escolha foi falar um pouco das duas sendo que os detalhes técnicos para se utilizar o registro faz-se necessário, e saber como funciona de fato os sistemas de registro UDDI ajudam no entendimento maior sobre o assunto Serviços *Web*.

Conforme Hendricks (2002) o UDDI consiste em quatro especificações principais:

Especificação da estrutura de dados: o objetivo desta especificação é o de definir uma interface que é avaliada de acordo com os termos de licença, definidas pelo grupo OASIS (*Organization for Structured Information Standards*) que tem o objetivo de padronizar as definições de estruturas de dados utilizadas dentro dos registros UDDI.

Essa especificação descreve que tipo de estrutura de dados é armazenado no UDDI, essa se baseia, como as outras tecnologias de Serviços *Web*, em XML o que permite então o que todos esperam, que é uma maior neutralidade com relação a linguagens de programação e plataformas. Essa estrutura de dados pode ser descrita por esquema de XML (XML *Schema*) que é publicado em forma de um documento separado, disponível no *site* de registro UDDI.

Um dos requisitos frisados pela especificação do UDDI é que as mensagens enviadas para o registro devem utilizar a codificação UTF-8, o que indica que os documentos XML gerados devem ter em sua primeira linha a seguinte definição para o documento.

```
<?xml version=”1.0” enconding=”UTF-8” ?>
```

Com isso garantimos que o mecanismo *Prolog* do XML irá codificar o arquivo no padrão definido, a especificação frisa também que o cabeçalho do HTML deve utilizar o tipo de conteúdo (*tag Content-Type*) conforme abaixo:

```
Content-Type: text/xml; charset= “utf-8”
```

De acordo com a especificação não se pode usar outro padrão de codificação, nem mesmo as variações do UTF-8 (UTF_8 e UTF8)

Especificação das API’s do programador: são definidas duas API’s que devem ser utilizadas pelos programadores dos Serviços *Web* e Consumidores; estas API’s serão utilizadas para que seja possível a Publicação e a Descoberta dos serviços sendo que as funções de publicação serão utilizadas pelo programador do Serviço *Web* para criar e atualizar entradas existentes nos registros, por outro lado, as funções de descoberta são somente de leitura, ou seja, tem como objetivo a pesquisa por serviço e normalmente são utilizadas pelos que implementam o consumidor para o serviço.

Segundo Hendricks (2002) as API’s independem da linguagem de programação uma vez que são utilizadas por intermédio de XML que descrevem em praticamente 100% o conteúdo do registro no UDDI. O Acesso aos serviços existentes se faz por meio do SOAP sobre o HTTP para as funções de pesquisas e no caso das funções de publicação podem ser utilizadas sobre o HTTPS, por exemplo. Se observada cuidadosamente esta especificação, pode-se notar que nela mesma são utilizados os conceitos básicos dos Serviços *Web*, já que existe uma comunicação entre aplicações para a descoberta e publicação de novos serviços. Mais adiante, nesta Sessão, um exemplo de como se comunicar com o registro UDDI será mostrado.

Especificação de Replicação: Dentro dos registros UDDI podemos ter diversos serviços publicados, a especificação de replicação só se faz necessária para que seja possível sincronizar a comunicação entre estes serviços, o que não é da competência de quem está

publicando ou pesquisando por um serviço e sim por quem implementa o registro UDDI, neste caso esta especificação se aplica para quem estiver implementando o seu próprio registro.

Especificação do Operador: a exemplo da especificação anterior, esta também é uma especificação para quem está implementando seu próprio registro UDDI, tem por objetivo definir políticas de segurança e de gerenciamento de dados. Alguns dos termos que podem ser definidos, por exemplo, são: deve solicitar um ID de usuário sempre que um registro ou alteração for efetuado; pelo lado do gerenciamento de dados podem ser definidas regras que diz que uma determinada entidade pode apenas publicar um número limitado de serviços para uma dada conta. E algumas outras definições que dizem respeito à política de segurança, podem ser implementadas.

Essas quatro especificações, embora não esgotem o assunto sobre as definições básicas sobre registros UDDI, nos da uma boa idéia do que tem que ser feito para publicar e pesquisar por Serviços *Web*. Quando se pesquisa por novos serviços, deve ter em mente o que realmente se espera do serviço, isso deve estar claro, porém uma coisa que ainda não foi mencionada é como interpretar as informações retornadas pelos registros, a próxima sessão descreve alguns dos detalhes técnicos do UDDI para que o programador de um serviço ou de um cliente saiba interpretar e utilizar o retorno de um UDDI e também padronizar o seu Serviço *Web* para o registro em específico.

3.1.1.4.1 Como usar os registros UDDI (para publicação e descoberta)

Quando se pensa em sistemas distribuídos, uma série de fatores parece ser um complicador para fazê-lo. Para Serviços *Web* temos muitas coisas que facilitam o desenvolvimento e aplicação da arquitetura proposta pela Sessão 3.1, um deles são os padrões de comunicação baseados em XML. Os dados UDDI são estruturados de uma forma bem amigável. Pense em um indexador por categoria assim como um catálogo telefônico. Então categorizar convencionalmente as categorias de um registro UDDI como sendo:

Gerais: Contém informações diversas como nome, endereço, número de telefone e outros dados específicos sobre entidades que decidiram publicar informações sobre si. Um ponto interessante e propício para o momento é dizer que os registros UDDI não têm como finalidade apenas servir como um registro de Serviços *Web*, tem também a finalidade de registrar dados sobre negócios diversos.

Classes de negócio: contém listas de dados de acordo com as categorias de negócio. Isso facilita, por exemplo, a busca ou registro de serviços que dizem respeito a vendas ou serviços de hotéis.

Detalhes por negócio: com uma maior aplicabilidade para quem implementa um serviço, pois são usadas para indicar serviços oferecidos por cada negócio, incluindo as informações técnicas que servirão para a interação entre Provedores de Serviços e seus Consumidores.

Tanto para publicação quanto para a descoberta de serviços esta estruturação deve ser usada, pois se quisermos publicar um Serviço *Web* sobre um dado negócio ou da uma dada empresa, dados como o nome da empresa, endereço, mercado de atuação e detalhes técnicos do serviço, devem ser publicados. A seguir será mostrado que os padrões UDDI são feitos por documentos XML que vão definir os componentes dentro daquele registro específico

3.1.1.4.2 Componentes para um registro UDDI

Para que se possa construir um registro ou para consultá-lo, devemos definir a estrutura do mesmo. Esta sessão faz-se presente para apresentar alguns dos componentes que definem essa estrutura de acordo com a especificação OASIS sobre o UDDI (BRYAN, 2002).

Componente tModel: para que dois ou mais componentes de um *Software* sejam compatíveis entre si o bastante para poder trocar dados com a finalidade de alcançar resultados desejados, eles devem compartilhar de metas e projeto em comum. Cada registro de serviço no UDDI está baseado neste conceito. Em um ambiente, simples, onde dois componentes de *Software* precisam se comunicar e de forma que cada um destes componentes tenha propósitos

comuns basta, então, definir uma interface comum entre os dois e o problema estará resolvido. Porém no cenário dos Serviços *Web*, os Provedores de serviço precisam de um modo de publicar suas especificações e versões destas para os serviços anunciados. Para se fazer isso, existem os meta-dados[3] que será tratado por *tModel*. Até aqui parece que já foi possível perceber, mas na Sessão 3.1.1.2 foi mostrado que a WSDL descreve a Interface para um Serviço *Web*, por tanto se seu serviço foi descrito por um documento WSDL, este será armazenado no UDDI como um *tModel*. O Exemplo 3.6, abaixo, mostra a definição de um *tModel* no padrão definido por Bryan (2002).

Exemplo 3. 6 - Componente tModel do UDDI

```
<tModel authorizedName= “CalcDemo”
   operator= “CALCULADORA/Service/uddi”
   tModelKey=”UUID:7C91BE10-EA7F-11D5-A6D4-9EAB24E10238”>

      <name>Calculadora tModel</name>

   <overviewDoc>
      <overviewURL>
            http://localhost:8080/axis/calculadora.wsdl
      </overviewURL>
   </overviewDoc>
</tModel>
```

Nesse exemplo uma demonstração de um meta-dados é mostrada, onde é definido um nome de autorização (`authorizedName`), qual é o registro operador (`operator`) e observe também que temos uma chave que identifica de forma única o Serviço *Web*, Essa é criada por um algoritmo que é definido no momento da implementação do registro UDDI e tem como *tag* correspondente a `tModelKey`. Em seguida o nome do meta-dados e uma URL que mostra onde está localizada a descrição do serviço.

Componentes *identifierBag* e *categoryBag*: Estes componentes muitas vezes fazem parte do `tModel`, e são usados para solucionar um problema que diz respeito a localização de serviços dentro do UDDI, Esses definem, respectivamente, Identificadores e Categorias de um

[3] Os meta-dados definem padrões para que um serviço seja publicado. Trata-se de uma estrutura básica utilizada pelo UDDI para a publicação e descrição de um Serviço *Web*

Serviço *Web*, segundo Hendricks (2002) porém, não são de fatos componentes UDDI. Essa referência fala sobre a versão 2.0 do UDDI já na especificação de versão 2.04, como foi possível perceber este é um componente de melhoria que já é citado na especificação e possuem as seguintes definições: ***identifierBag*** - procuras podem ser executadas através de um identificador, porém isso é feito por uma chave de referência para o identificador e destacada pela *tag* `<keyedReference>`; ***categoryBag*** - buscas podem ser realizadas por categorias de serviços que também não deixa de ser um identificador e por isso é auxiliado pela mesma *tag* `<keyedReference>`.

Os criadores do UDDI acharam interessante criar algumas taxonomias predefinidas que facilitassem essa divisão por categorias. Foram então criados quatro *namespaces*, cada um deles representados por um *tModel* que podemos utilizar para classificar nossos dados:

- UNSPSC – *Universal Standard na Products Classifications* – padrão para diferentes produtos;
- NAICS – *North American Industry Classification System* – padrão que define um sistema de classificação para negócios, com base na industria onde eles operam;
- ISSO 3166 – *Geographic Classification System* – padrão que descreve diferentes regiões e localizações ao redor do mundo;
- Outros – Taxonomia para palavras-chave de tipo geral.

O próximo componente apresentado é utilizado para se publicar um negócio em um registro UDDI, para que posteriormente os serviços possam ser vinculados ao negócio já registrado.

Componente *businessEntity*: quando pensamos que uma empresa possa publicar vários serviços, então pensamos também em uma forma de agrupar os serviços de um mesma empresa de forma que clientes desta encontre mais facilmente os seus serviços. Para que se possa então publicar o negócio que agrupará os serviços de uma empresa faz-se necessário descrever o componente *businessEntity*. Esse componente é a entrada chave no registro UDDI que contém informações sobre o negócio como nome, endereço, número de telefone e possivelmente sua categoria e identificação conforme mostrado, pelos componentes *identifierBag* e *categoryBag*.

Uma entrada *businessEntity* é mostrado abaixo no Exemplo 3.7

Exemplo 3. 7 - Uso do componente *businessEntity* - UDDI

```
<businessEntity businessKey =
      ”4C890B30-EB6D-11D5-A947-97BA2552F30”>
      <discoveryURLs>
         <discoveryURL useType=”businessEntity”>
            http://www.ibm.com/services/uddi/uddiget?
            businesskey=318C6450-EB6B11D5-A947-97BA25522F30
         </discoveryURL>
      </discoveryURLs>
      <name>Leandro César Mendes</name>
      <description>Calculadora simples</description>
      <contacts>
            <contact>
                  <personName>Leandro Mendes</personName>
                  <email>leoemail@prover.com</email>
                  <address>
                        <addressLine>Rua XXX</addressLine>
                        <addressLine>135, Centro</addressLine>
                        <addressLine>Congonhas</addressLine>
                        <addressLine>MG-Brasil</addressLine>
                  </address>
            </contact>
      <contacts>
      <categoryBag>
            <keyedReference keyName=”Calculos”
                  keyValue=”541611”
                  tModelKey=
                  ”UUID:7C91BE10-EA7F-11D5-A6D4-9EAB24E10238”>
      </categoryBag>
</businessEntity>
```

No exemplo acima é apresentado um modelo de entrada de *businessEntity* que definiu um negócio que possa então vincular os serviços futuramente criados. Com o negócio

publicado já é possível pesquisar pela categoria CALCULOS e encontrar a empresa como um possível Provedor de Serviços no resultado retornado.

Componente *businessService*: Quando se escreve serviços provavelmente estes serão registrados. Com um negócio já registrado pode-se então fazer o registro dos serviços de mesma categoria. O componente *businessService* é o responsável por esse registro, o componente possui basicamente todos os elementos definidos no UDDI, contém o nome de um serviço, uma descrição opcional, um <categoryBag> além de uma lista de elementos <bindingTemplate>. Além destes elementos, o componente businessService também possui os atributos businessKey e serviceKey que referenciaram o serviço e o negócio relacionado, por meio de uma chave que é gerada no momento em que a inclusão for feita no registro. O elemento < bindingTemplate > é usado para descrever informações específicas de um serviço e é opcional pois podíamos querer que o serviço tivesse apenas sua identificação e descrição sem maiores detalhes.

Os componentes apresentados acima são definidos pela especificação OASIS de acordo com Bryan (2002) que traz muitos detalhes sobre as APIs de publicação e descoberta do UDDI.

Mostradas as principais características e componentes da estrutura de dados do UDDI concluído assim o Capítulo 3 sobre Serviços *Web*. O próximo capítulo irá mostrar alguns tópicos sobre segurança para os Serviços *Web*. Em um capítulo simples e objetivo, serão apresentadas algumas formas de se implementar a segurança nos serviços bem como algumas das mais recentes tecnologias utilizadas.

4 SEGURANÇA EM SERVIÇOS *WEB*

Como já sabemos, todo o tipo de tecnologia que envolve comunicação em rede requer implementação de segurança. Este assunto é muito interessante e abrangente, quando se discute sobre segurança, o assunto parece não ter fim e não ter mecanismos ou técnicas 100% seguras. O que é possível fazer então é dificultar ao máximo o acesso indevido às informações (na maioria dos assuntos: relacionados à informática). O presente capítulo irá mostrar, de forma superficial, algumas formas e tecnologias para se implementar a segurança nos Serviços *Web*. Para começar: quais seriam as formas de se implementar a segurança em uma Arquitetura como a dos Serviços *Web*? Existem três formas de se implementar a segurança nessa arquitetura. São elas:

Em nível de rede: Uma boa alternativa para a segurança neste nível pode ser adotar as VPN (*Virtual Private Network*), que normalmente são baseadas em IPSec (*Internet Protocol Security*), que fornece um conjunto de padrões de segurança dentre eles o de autenticação de dados e de usuários

Em nível de transporte: Quando se fala em segurança em nível de transporte entra-se na questão de se implementar segurança nos protocolos que são destinados a este fim, tem-se então a possibilidade de utilizar SSL (*Security Socket Layer*), trata-se de uma camada de segurança implementada para troca de dados via *Internet* que pode ser “acoplada” ao HTTP, dando origem ao que conhecemos como HTTPS onde o “S” indica que o protocolo HTTP está trabalhando em conjunto com uma camada extra de segurança.

Em nível de aplicação: Neste nível é possível citar vários padrões de segurança e a maioria deles baseadas em XML. O Consórcio W3C e o grupo OASIS possuem alguns projetos

relacionados com esta forma de segurança, estes padrões serão apresentados mais adiante em forma de tecnologias XML disponíveis para implementar a segurança para os Serviços *Web*.

Segundo Hendricks (2002) a segurança em Serviços *Web* deve cobrir os seguintes aspectos:

Identificação e autenticação; Autorização; Integridade; Privacidade; Não-repúdio;

Esses são alguns dos pré-requisitos para se pensar em segurança, algumas tecnologias são utilizadas para garantir ou pelo menos tentar, a segurança da informação. Abaixo são apresentadas na Tabela 3 algumas tecnologias baseadas em XML que se destinam a garantia de segurança para os Serviços *Web*:

Tabela 3 - Lista de padrões e tecnologias empregadas em segurança para Serviços *Web*

Tecnologia	Descrição
XKMS (*XML Key Management Service*)	Este projeto destina-se a tornar-se possível o desenvolvimento de Serviços *Web* de confiança . A XKMS especifica protocolos para distribuir e registrar chaves públicas, para serem usadas pelos padrões de assinatura XML e criptografia XML.
XML *Encryption*	Criptografia da mensagem no remetente e decriptografia no destinatário. Para maiores detalhes consulte Reagle (2002).
SAML (*Security Access Markup Language*)	É um padrão de segurança para troca de credenciais de autenticação e autorização baseadas em XML, por traz do SAML está um esquema XML que define a representação de dados seguros.
XACML (*Extensible Access Control Markup Language*)	Tecnologia também baseada em XML que temos como ponto forte à expressão de regras e políticas de segurança, este tipo de controle é muito comum dentre os sistemas operacionais, onde os critérios podem ser... ...atribuídos a grupo e/ou papeis de

	usuários
WSS (*Web Service Security*)	Integra e unifica vários modelos e tecnologias de segurança existentes no mercado, permitindo que vários sistemas possam inter-operar em plataforma e linguagens distintas. Uma das tecnologias presentes é, dentre outras, a *XML Encryption*

A Tabela 3, acima, apresentou algumas das tecnologias que podem ser empregadas para se implementar um segurança mais confiável e são baseadas em padrões novos e por isso, algumas destas ainda estão em andamento e padronização para que possam atender melhor às necessidades para a arquitetura de Serviços *Web*.

A idéia central nesse capítulo foi a de apresentar e mostrar que embora bastante recente, a tecnologia de Serviços *Web* é suportada por diversos grupos e instituições que abraçaram a tecnologia e fazem com que ela cresça e adquira credibilidade e confiança. No mercado existem empresas como a IBM, SAP e Microsoft que já utilizam os Serviços *Web* em seus negócios e plataformas. Cada dia esta utilização aumenta.

No capítulo seguinte serão apresentadas algumas das tecnologias Java disponíveis para implementar os Serviços *Web* e clientes para os mesmos. Será abordada a implementação do projeto WEBCALC no Capitulo 6 que utilizará algumas das ferramentas e tecnologias apresentadas.

5 TECNOLOGIAS E FERRAMENTAS PARA SERVIÇOS *WEB* UTILIZANDO JAVA

A implementação de Serviços *Web* é bastante complexa e requer um conhecimento de diversas tecnologias e ferramentas. O presente capítulo tem como objetivo mostrar algumas tecnologias e ferramentas que podem ser usadas para a implementação de Serviços *Web*, ressaltando que todas as ferramentas usadas serão diretamente relacionadas com Java. Vale lembrar que não é o foco deste Trabalho de Conclusão de Curso uma explicação da linguagem Java e sim como utilizá-la para a implementação de Serviços *Web*.

Para um início, nada melhor que falar um pouco sobre o Java e o porque da adoção dessa tecnologia. Java na verdade é: além de uma poderosa linguagem de programação, é um mundo de tecnologias diversas que permitem um trabalho totalmente integrado entre as mesmas. Sistemas Java consistem em várias partes: um ambiente, a linguagem, a API Java e várias bibliotecas de classes. O uso de sistemas totalmente desenvolvidos em Java é o ponto chave deste Trabalho de Conclusão de Curso para a implementação do projeto WEBCALC. As Sessões seguintes mostram algumas das tecnologias e ferramentas que podem ser usadas para implementar um Serviço *Web*.

5.1 Tecnologias Java para Serviços *Web*

Conforme dito anteriormente Java é um mundo de tecnologias e padrões. Nesta sessão serão apresentadas algumas destas tecnologias de forma sucinta e introdutória. Quando o

assunto for Ferramentas Empregadas, maiores detalhes dessas tecnologias serão apresentados. A listagem abaixo mostra o que é cada uma destas:

***Servlets*:** um *Servlet* é uma classe da linguagem de programação Java que tem como objetivo expandir a capacidade dos servidores *Web* que permitem que aplicações *hosts* acessem este por meio de um modelo de programação de *request/response*. A tecnologia *servlet* é muito usada, principalmente em servidores Java como o Tomcat, por exemplo. O pacote de instalação do *Java Software Development Kit* (JSDK) fornecem um conjunto de classes e interfaces conhecidos como pacotes de desenvolvimento para implementar *servlets*, são eles: `javax.servlet` e `javax.servlet.http`. Qualquer implementação de um *servlet* deve implementar uma interface *Servlet* que define as assinaturas dos métodos que serão usados e fornecer também os mecanismos de comunicação necessários.

JSP – *Java Server Page*: a tecnologia JSP permite ao desenvolvedor a capacidade de criar páginas para *Web* de forma a conter, ao mesmo tempo, conteúdos estáticos e dinâmicos. A tecnologia JSP torna possível todas as capacidades de dinamismo oferecidas pela tecnologia *Servlet*. As principais características do JSP são: documentos baseados em texto que descreve como processar uma resposta e como compor uma requisição; é uma linguagem que trabalha no lado do Servidor; permite integração no mesmo documento de código Java e um outro que pode ser HTML ou XML, por exemplo.

Componentes *JavaBeans*: de acordo com Jendrock (2003), os componentes *JavaBeans*, ou simplesmente *Beans*, são classes Java que podem ser facilmente reutilizada e serem integradas à aplicação. Dizemos que qualquer classe Java que segue algumas convenções de desenvolvimento pode ser tratada como um *JavaBeans*. Estas convenções são as seguintes: Escrita/Leitura de propriedades leitura somente e/ou escrita somente. Normalmente um *JavaBean* define uma classe sem operações complexas. Um exemplo de *JavaBean* pode ser um classe que armazene dados de um cliente. Um *JavaBean* também deve conter um construtor *default* sem parâmetros.

RMI – *Remote Method Invocation:* Esta tecnologia permite que, utilizando Java, possa se criar sistemas distribuídos.

> “A RMI permite que objetos Java executando em um mesmo computador ou em computadores diferentes se comuniquem entre si via chamadas remotas dos métodos, que são muito semelhantes as executadas entre objetos de um mesmo programa. A RMI

> está baseada em uma tecnologia anterior utilizada pelos programas procedurais chamada RPC (*Remote Procedure Call*)". (DEITEL 2001. p. 891).

Essa capacidade de invocação remota é o que teoricamente dá a capacidade dos nossos Serviços *Web* funcionarem de forma a fornecer respostas a invocações realizadas por qualquer tipo de aplicação, a diferença é que utilizando a RMI estamos vinculados à linguagem de programação Java.

JAXP – *Java API for XML Processing:* utilizado para o processamento de dados no formato XML a API JAXP oferece várias facilidades para o tratamento de documentos XML, tendo em vista que todo o padrão dos protocolos envolvidos na tecnologia de Serviços *Web* é baseado em XML, esta API é de grande utilidade para que pretende implementar um serviço em Java. A API JAXP implementa dois padrões de tradução de documentos o SAX (*Simple API for XML Parsing*) e o DOM (*Document Object Model*), estes padrões são utilizados para interpretar os documentos XML usados por uma aplicação. Por hora, sabemos então que estes dois padrões são abrangidos pela API JAXP, uma diferença básica entre os dois está na capacidade de gravação de dados em XML o que é permitido somente pelo DOM. O SAX só permite a leitura de dados XML.

JAXR – *Java API for XML Registries:* provê uma forma conveniente para acessar o registro de negócios através da *Web*. Como visto no Capitulo 3, uma categoria do registro que convencionalmente chamamos de Classes de negócio, a API JAXR tem como objetivo fornecer meios, aos programadores Java, de se fazer o registro de um negócio que é baseados em padrões como o UDDI. Com já vimos anteriormente também é possível pesquisar por um registro de negócio esta funcionalidade também é abrangida por esta API.

JAX-RPC – *Java API XML-based RPC*: esta API é utilizada para desenvolver e usar Serviços *Web*. Um serviço baseado em RPC é um conjunto de procedimentos que pode ser executado por um cliente remoto. Neste cenário pode-se então utilizar essa API para a construção de novos Serviços *Web*. Quando se faz necessário trabalhar com o protocolo SOAP, esta tarefa é um tanto quanto árdua, para isso a API JAX-RPC fornece um encapsulamento das funções complexas de utilização do SOAP. De forma simples, o serviço pode ser criado usando apenas os métodos que serão executados remotamente sem muito esforço, fazendo-o de forma semelhante a um sistema local. A complexidade maior está em desenvolver o cliente para o serviço, porém essa tarefa se torna muito mais simples utilizando-se da JAX-RPC.

O Conjunto de tecnologias acima apresentado pode ser utilizado por Java que podem ser utilizadas nos Serviços *Web*. Maiores detalhes podem ser obtidos em (JENDROCK, 2003).

Tendo em mente quais APIs e tecnologias podem ser usadas pode-se falar então sobre algumas das ferramentas que são verdadeiros frameworks que encapsulam as funcionalidades de grande parte das tecnologias apresentadas acima. Será mostrado que para se fazer um Serviço *Web* utilizando estas ferramentas é bem menos complicado do que parece.

5.2 Ferramentas Empregadas no Cenário dos Serviços *Web*

O principal recurso para um programador Java, são as ferramentas hoje disponíveis no mercado, sendo que a maioria é de distribuição gratuita. Será tratado nesta sessão de algumas delas que podem ser empregadas ao desenvolvimento de Serviços *Web*. De forma simples e objetiva, o intuito dessa Sessão é mostrar como se deve utilizar cada uma dessas tecnologias na forma de um manual claro e simples para os propósitos práticos deste Trabalho de Conclusão de Curso.

É sabido que para se montar uma arquitetura completa e que se possa testar os serviços, é necessário um servidor *Web* para receberem os Serviços *Web* criados, se faz necessário ter um ambiente de programação para a implementação dos códigos em Java, obviamente, ferramentas diretamente ligadas com a tecnologia de Serviços *Web* e uma forma de sincronizar tudo isso. Está abordagem será separada em sessões onde iniciaremos pelo lado do servidor, neste caso o Tomcat, em seguida uma análise de duas ferramentas que são “acopladas” ao Tomcat será realizada para permitir a publicação dos serviços e em seguida um pacote de desenvolvimento Java que fornecerá todas as API’s necessárias para se implementar tanto os serviços quanto os clientes deste. Não serão apresentadas neste trabalho formas para se implementar um diretório de Registros UDDI, o foco é na implementação dos serviços.

5.2.1 Tomcat

O que é o Tomcat? Tomcat é um *servlet container* também conhecido como *Web container*, que é usado como referência oficial da implementação de *servlets* e JSP. É um "*open source*" que faz parte de um projeto conhecido por Jakarta da *Apache Foundation.* Para cada versão do Tomcat atende a uma versão *Servlet* e JSP. A Tabela 4, apresentada abaixo mostra as versões correspondentes para cada versão do Tomcat.

Tabela 4 - Versões relativas à implementação do Tomcat

	Servlets	JSP	Tomcat
Versões	2.4	2.0	5.0.16
	2.3	1.2	4.1.29
	2.2	1.1	3.3.1ª

Fonte: PROCWORK, 2004, p. 64.

Talvez haja dúvida sobre o que isso tem de relevante para um programador, pois bem. Tente impelementar um *Servlet* ou um JSP de versão 2.4 e 2.0 respectivamente em uma versão do Tomcat 3.3.1a, isso pode ser desastroso, pois muito dos recursos utilizados por versões JSP e *Servlets* mais novos não são abrangidos pelo Tomcat de versão mais antiga. A relação entre estes três é muito estreita sendo assim ao implementarmos um *servlet* ou um JSP devemos importar o pacote *Servlet.jar* localizado em (*tomcat_home/commom/lib*). Um caminho simples que será o utilizado neste Trabalho de Conclusão de Curso será a utilização de um pacote conhecido JWSDP1.4. (*Java Web Service Developer Pack versão 1.4*) Esse pacote é um conjunto de ferramentas necessárias para se implementar e testar os Serviços *Web*. Ele agrupa, dentre outras tecnologias, suporte para o Tomcat, suporte ao protocolo SOAP, as tecnologias JAXP, JAXR, JAX-RPC e algumas opções relativas à segurança, que foram abordados no Capítulo anterior.

Como ter acesso a este pacote, então? No CD anexo existe uma pasta denominada JWSDP1.4 com a instalação deste pacote. No Apêndice C também em forma de conteúdo contido no CD, é apresentado um check-list de como se proceder com a instalação. Antes, porém, de

fazê-lo é necessário ter a *Java Virtual Machine* (JVM) instalada em seu computador, confira Apêndice A no CD para saber como instalar a JVM.

A instalação desse pacote criará uma estrutura de pastas que é muito importante, pois é necessário saber onde o Tomcat publica seus arquivos e onde se pode publicar os *Servlets* ou JSP. A Figura 8 mostra esta estrutura de pastas .

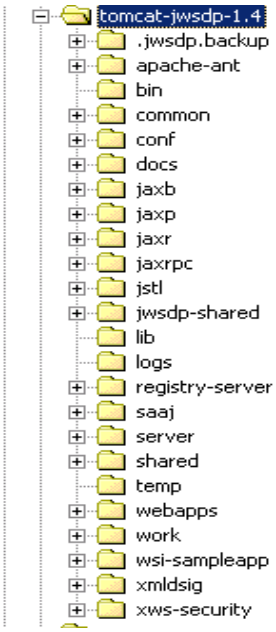


Figura 8 - Estrutura de Pastas do Tomcat

A Figura 8, acima mostra como é a estrutura de pastas do Tomcat, cada uma delas recebe um grupo de arquivos que são necessários ao Tomcat e algumas são utilizadas para receber novos arquivos, algumas das principais pastas estão relacionadas na Tabela 5.

Tabela 5 - Relação das principais pastas da estrutura criada pelo Tomcat

Pasta	Descrição
\bin	Pasta contendo os arquivos de inicialização e finalização do Tomcat, statup.bat e shutdown.bat respectivamente no caso de se utilizar o sistema operacional Linux os arquivos são startup.sh e shutdown.sh
\commom	Nesta pasta encontram-se os pacotes de acesso comuns a todas as aplicações executadas sobre o Tomcat, dentre eles o servlet-api.jar que é de uso obrigatório pelas aplicações que serão publicadas. Estes

	pacotes são colocados dentro da sub-pasta \lib
\conf	Aqui é o coração do *Web container* Tomcat, nesta pasta estão os XMLs de configuração do servidor Tomcat como o *Server*.xml que define, dentre outras coisas, a porta que será usada para a conexão http
\docs	Documentos sobre a instalação e utilização por programadores do servidor Tomcat, deve ser sempre consultado em caso de dúvidas sobre o uso do Tomcat
\webapps	É nesta pasta que estão os *servlets*, JSP e qualquer outra aplicação que será publicada no Tomcat, nesta pasta, mais adiante será adicionadas as sub-pastas para o servidor Apache-SOAP e a pasta do AXIS que serão descritos mais adiante neste capítulo.

A Tabela 5, acima será útil no momento em se for publicar um serviço ou um JSP ou qualquer outra aplicação, é muito importante distribuir os arquivos de um projeto nas devidas pastas, para que este possa funcionar corretamente.

5.2.2 Apache SOAP

Agora que uma idéia básica para a utilização do Tomcat foi dada e como este está relacionado com as tecnologias Java apresentadas na Sessão 5.1 pode-se começar a estruturar o servidor para publicação de Serviços *Web*. Embora o Tomcat seja um servidor que permite uma série de coisas e dentre elas a criação de sistemas distribuídos, devemos mudar um pouco sua estrutura inicial para adequarmos à execução dos Serviços *Web*, isso se faz, pois neste cenário um provedor de recursos SOAP faz necessário, se fossemos utilizar somente os recursos do Tomcat, que é possível fazê-lo, teríamos muito trabalho em codificação e não teríamos a empregabilidade de alguns conceitos da tecnologia de Serviços *Web*.

O Apache SOAP foi desenvolvido originalmente pela IBM quando era conhecido por IBM-SOAP, foi doado para *Apache Software Foundation*. O motivo que levou a IBM a

tomar esta atitude segundo Hendricks (2002) foi o fato de que o *toolkit* da Microsoft SOAP estava crescendo no mercado, então a IBM julgou interessante para a sobrevivência do IBM-SOAP que ele fosse fornecido como uma implementação de código fonte aberto para a comunidade de Desenvolvedores Java.

O Apache SOAP deve ser usado em conjunto com o Tomcat para isso é necessário então conseguir a instalação do Apache SOAP e em seguida adicioná-lo ao Tomcat. No cd anexo é encontrado na pasta apache_soap os arquivos necessários para se instalar o Apache SOAP e no Apêndice D (No CD) é mostrado como proceder com a instalação do Apache SOAP.

Após a instalação e configuração do Apache SOAP é possível criar um Serviço *Web* para ser publicado no apache SOAP. Para isso será utilizado o Exemplo 3.5 mostrado no Capitulo 3, para os testes iniciais. A seguir é descrito como se publica uma classe criada para que esta possa atuar como um Serviço *Web*, isto é feito através de um utilitário do Apache SOAP de forma completamente visual. Este utilitário é mostrado na Figura 9 abaixo.

Figura 9 - Utilitário de administração do Apache SOAP

Através do utilitário apresentado na Figura 9 é possível listar(*List*) os serviços já cadastrados e disponíveis, efetuar a inclusão de um novo serviço (*Deploy*) e remover (*Un-deploy*) um serviço previamente registrado, para publicarmos o exemplo teremos que saber como preencher os campos do formulário de registro solicitado pelo Apache SOAP este formulário é apresentado na Figura 10, mais adiante. O ato de preencher os campos desse formulário criará para automaticamente um XML de configuração para o serviço.

A descrição dos principais campos é apresentada abaixo:

***ID*:** o campo ID é utilizado para especificar o nome do Serviço. Os clientes SOAP utilizam o campo ID para rotear as solicitações para o serviço; Pode ser usada uma sintaxe URN (*Universal Resource Name*) para descrever o nome do serviço, bastando para isso preceder o nome do serviço pelo namespace de XML (urn:) como um exemplo poderíamos adotar para o nosso exemplo o ID = “urn:Calculadora”;

***Scope*:** define um tempo de vida que a instancia da classe de implementação do serviço terá, podendo assumir os valores: *Request* (O objeto é removido da memória após sua solicitação), *Application* (O objeto só é destruído quando o *servlet* que está atendendo as solicitações tenha terminado), *Session* (Dura o tempo que durar a sessão do usuário, em caso do protocolo HTTP estiver sendo usado, basta fechar o *browser* para encerrar a sessão);

***Method*:** conjunto de métodos da nossa classe Java que queremos tornar disponíveis para outra aplicação (em casos onde mais de um método são disponibilizados, a lista deve ter seus itens separados por espaço);

***Provider Type*:** define qual é a origem da implementação podendo assumir os valores Java (indica que o código provém de uma classe Java comum), *Scripts* (este é o caso onde se utiliza código de *script*) ou *User-defined* (para tipos de código como o *JavaBeans*, por exemplo);

Provider Class: em caso de *Provider Type* ser do tipo Java então se faz necessário especificar qual será a classe a ser utilizada;

***Static*:** também se aplica somente nos casos onde *Provider Type* é definido com Java, este método pode ser *true* ou *false* dependendo do tipo de método que está sendo exposto.

***Default Mapping Registry Class*:** Define uma classe que será responsável pelo mapeamento padrão para tradução dos XML e pelas exceções geradas. Uma classe padrão é definida: `org.apache.soap.servlet.DOMFaultListener`

Com base nestas informações qualquer serviço pode ser publicado e assim tornado disponível para as classes clientes que será mostrado em seguida neste capítulo.

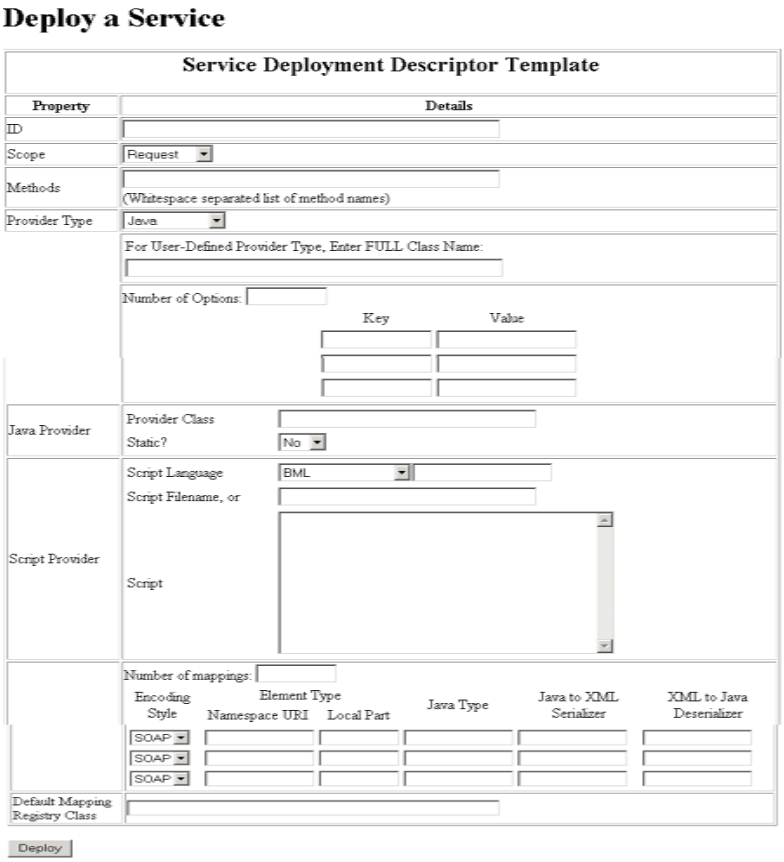


Figura 10 - Formulário para registro de Serviço no Apache SOAP

Através desse formulário é que se faz o registro dos Serviços *Web*. Para o exemplo proposto o preenchimento dos campos se faria da seguinte forma:

ID = "urn:calculadora"

Scope = "*Application*"

Methods = "somar subtrair multiplicar dividir"

Provider type = "Java"

Provider Class = "Calculadora"

Static = "*false*"

Default Mapping Registry

Class = "org.apache.soap.Server.DOMFaultListener"

Assim o serviço, calculadora, será publicado e estará pronto para ser utilizado por uma aplicação cliente, que é o nosso próximo passo sobre o Apache SOAP, de forma que este

cederá lugar à sua nova versão que chamamos de AXIS, porém isso é assunto para a próxima sessão.

Exemplo 5. 1 - Cliente para acessar um serviço publicado no Apache SOAP

```
import org.apache.soap.Constants;
import java.net.URL;
import java.util.Vector;
import org.apache.soap.Fault;
import org.apache.soap.rpc.*;
import org.apache.soap.rpc.Response;
import org.apache.soap.rpc.Parameter;
public class CalculadoraCliente {
      static String DEFAULT_ENDPOINT =
      "http://localhost:8080/soap/servlet/rpcrouter";
      public static void main(String[] args) throws Exception {
            String endpoint = DEFAULT_ENDPOINT;
            Vector vParametros = new Vector();
            // Construir a mensagem de requisição do SOAP RPC
            // usando o objeto CALL
            Call call = new Call();
            call.setTargetObjectURI("urn:Calculadora");
            call.setMethodName("somar");
            // parâmetros para execução do método somar
            vParametros.addElement(new Integer(15));
            vParametros.addElement(new Integer(5));
            call.setParam(vParametros);
            call.setEncodingStyleURI(
                  Constants.NS_URI_SOAP_ENC);
            // Criar um objeto URL, que representa a
            // extremidade
            URL url = new URL(endpoint);
            // Enviar a mensagem de solicitação do
            // SOAP RPC usando o método invoke()
            Response resp = call.invoke(url,"");
            // verificar se a resposta gerou uma exceção
            if (resp.generatedFault()) {
                  Fault fault = resp.getFault();
```

```
                        System.out.println("Ocorreu Erro");
                        System.out.println("Código = " +
                              fault.getFaultCode());
                        System.out.println("Descrição = " +
                              fault.getFaultString());
                  } else {
                        // Recupera o resultado
                        Parameter result = resp.getReturnValue();
                        System.out.println(result.getValue());
                  }
            }
      }
```

O resultado impresso para o Exemplo 5.1 acima, será o inteiro 20, caso não ocorra nenhuma falha no mecanismo ou no código da classe que originou o serviço, pois se isso acontecer o resultado será o aviso de tal falha. O mecanismo SOAP fornece um elemento chamado *SOAP Fault* citado no Capítulo 3, que tem como finalidade o tratamento das exceções criadas pelo protocolo, isto ocorre por causas diversas.

No exemplo acima é possível perceber a presença de pacotes do SOAP. Para que seja possível que um cliente se comunique com o serviço é indispensável tais pacotes. Um ponto curioso do código apresentado é a simplicidade. Bastando um conhecimento, básico, sobre Java para o seu entendimento. A constante `DEFAULT_ENDPOINT` que foi definida é um apontador para o mecanismo do Apache SOAP responsável pela localização e execução do serviço em questão e responsável pelo protocolo de RPC do SOAP. Ao se implementar um cliente para um Serviço *Web* com estas características é necessário se ter o componente soap.jar que está disponível na instalação do Apache SOAP.

Vimos que o Apache SOAP tem um ponto a seu favor que é a simplicidade com a qual podemos registrar nossos serviços e a maneira como devemos implementar os clientes que os utilizarão. Na próxima Sessão, conforme já antecipado anteriormente, será mostrada a terceira geração do Apache SOAP que teve o seu nome adaptado e passou a se chamar AXIS, muitas melhorias e novas possibilidades para o registro de serviços foram realizadas. Detalhes desta ferramenta serão apresentados para que seja possível argumentar melhor sobre uma decisão em tempo de projeto da implementação do WEBCALC.

5.2.3 AXIS (*Apache eXtensible Interaction System*)

Para completar o *tour* pelas tecnologias e ferramentas, será apresentada no presente capítulo a terceira geração do Apache SOAP, chamada: AXIS (*Apache eXtensible Interaction System*). O AXIS é uma evolução significativa em relação ao andamento dos projetos para o protocolo SOAP. O AXIS é um projeto que está em andamento e foi adotado para dar ao projeto melhorias como uma maior flexibilidade, configurabilidade, e capacitá-lo para um maior controle sobre o protocolo SOAP e para o Protocolo XML que está sendo padronizado pelo W3C. O AXIS está atualmente na versal Alpha 3.0 e possui as seguintes características chaves:

Desempenho: o Apache SOAP implementa o DOM como mecanismo para a tradução do XML enquanto o AXIS utiliza o SAX, o que o torna bem mais eficiente que o seu antecessor;

Flexibilidade: A arquitetura do AXIS permite ao desenvolvedor uma liberdade muito grande, permitindo que novas extensões possam ser feitas para customizar, por exemplo, o processamento de cabeçalho, gerenciamento do sistema e qualquer outra coisa que se possa imaginar, segundo especificações que acompanham o produto;

Desenvolvimento orientado a componentes: é possível desenvolver componentes altamente reutilizáveis que seguem um padrão comum para a sua aplicação, e distribuído aos vinculados com o negócio dentro do ambiente destinado;

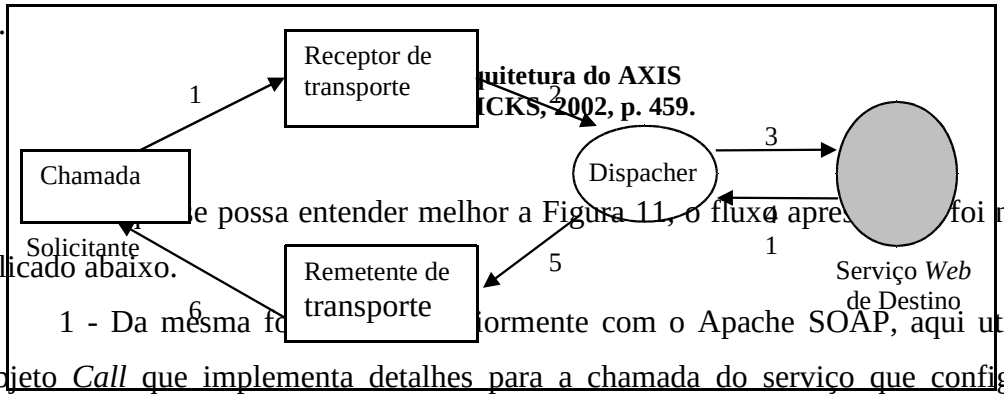
Framework de transporte: o AXIS fornece um framework que fornece de forma simples e clara uma abstração para a camada de transporte da pilha de protocolos. Desta forma ele se torna totalmente independente da camada de transporte;

Suporte a WSDL: o AXIS fornece vários mecanismos para se trabalhar com a WSDL de forma que simplesmente sejam criados os documentos correspondentes aos serviços publicados sobre o AXIS;

Como é possível notar muitos pontos temos a favor do AXIS e por isso foi a escolha para a implementação do WEBCALC. A utilização do AXIS é muito simples, esta ferramenta também como o Apache SOAP trabalha sobre um *Web container* como o Tomcat e deve ser agregado ao mesmo, sua instalação é discutida no Apêndice E (No CD), e uma cópia da sua instalação pode ser encontrada no CD anexo a este Trabalho de Conclusão de Curso.

5.2.3.1 Arquitetura AXIS

Para uma maior compreensão sobre o funcionamento dos componentes envolvidos com o AXIS uma arquitetura bem simples de ser compreendida é apresentada na Figura 11.



quitetura do AXIS
ICKS, 2002, p. 459.

Para que possa entender melhor a Figura 11, o fluxo apresentado foi numerado e será explicado abaixo.

1 - Da mesma forma que anteriormente com o Apache SOAP, aqui utiliza-se o mesmo objeto *Call* que implementa detalhes para a chamada do serviço que configura uma mensagem SOAP que em seguida é encaminhada ao nó de processamento de mensagens do AXIS. Neste ponto está com o componente Receptor de Transporte em um formato específico do protocolo da rede. O Receptor de transporte encapsula a funcionalidade para o tratamento do protocolo da rede, esta mensagem é recebida por esta extremidade que a prepara para o analisador XML e também detalhes relativos à requisição, específicos do protocolo utilizado (normalmente e mais comumente HTTP).

2 - Após conversões necessárias, a mensagem é transferida para um objeto *Message*, contido no pacote `org.apache.axis.Message`, além desta conversão são também carregadas as propriedades do SOAP como o *Action*.

Baseado no protocolo de aplicação utilizado a propriedade `transportName` é preenchida, podendo ser SMTP, HTTP ou FTP. Após feito isso, a mensagem é encaminhada ao *Dispacher*

3 - O componente *Dispacher* então se encarrega em chamar o Serviço *Web* de destino, que pode ser implementado utilizando diferentes linguagens.

4 - O Serviço *Web*, por sua vez, executa o método específico e retorna a mensagem ao *Dispatcher* .

5 - Com o retorno da execução do método pelo Serviço *Web* a mensagem é encaminhada para o Remetente de Transporte que possui características semelhantes às do Receptor de Transporte encapsulando as mesmas funcionalidades com relação aos protocolos de transporte que possivelmente podem ser utilizados.

6 - O Remetente do Transporte termina o fluxo enviando a mensagem XML de volta, através do protocolo da rede, ao solicitante.

Na prática um cliente para o AXIS é muito semelhante ao cliente do SOAP, com poucas modificações no código, o que realmente é diferente é modo como o AXIS dá o tratamento à solicitação.

Como dito na Sessão anterior, o código do serviço é muito simples e não utiliza nenhum componente específico para o fazê-lo. O que vai diferir, neste momento, em comparação com o Apache SOAP é que o AXIS fornece duas formas para que se publique um serviço, uma delas é criar um arquivo descritor do serviço conhecido como WSDD (*Web Service Deployment Descriptor*), essa é a maneira que normalmente é utilizada, pois este descritor é utilizado para encapsular detalhes da Classe que será utilizada como Serviço, porém não será abordado neste trabalho. Uma segunda forma de se fazer o registro, que se trata de uma forma acadêmica, é re-nomearmos o arquivo .java gerado para .jws, assim, se quiséssemos publicar a nossa Calculadora como um serviço para o AXIS, bastaria re-nomear o arquivo Calculadora.java para Calculadora.jws e colocá-lo na pasta do AXIS que foi introduzida dentro do Tomcat. Consulte o Apêndice E (No CD) sobre a instalação do AXIS para saber exatamente como fica a estrutura de pastas do Tomcat e do AXIS para que seja colocado o serviço.jws na pasta correta.

Essa forma de desenvolvimento não é segura, pois expomos o código fonte, porém como dito anteriormente, é uma forma acadêmica. Para uma referência mais detalhada sobre a

primeira forma de desenvolvimento, utilizando o WSDD consulte help do AXIS disponível na sub-pasta: ”/docs” da raiz do AXIS.

Abaixo é apresentado um exemplo de código fonte para um possível cliente do serviço Calculadora (Exemplo 3.5 do Capítulo 3, porém aqui não existe um utilitário para torná-lo disponível sob o AXIS, o procedimento é re-nomear o arquivo Calculadora.java para Calculadora.jws e disponibilizálo na pasta do AXIS, dentro da estrutura do Tomcat, conforme dito anteriormente).

Exemplo 5. 2 – Cliente para Serviço *Web* Calculadora.jws (AXIS)

```
import org.apache.axis.client.*;
public class Calc_cliente {
      // Construtor da Classe de teste
      public Calc_cliente (){
            try {
                  String urlWS =
                   "http://localhost:8080/axis/Calculadora.jws";
                  // Parametros para execução dos métodos
                  Calculadora
                  Object[] parametros = {new Integer(3),
                                    new Integer(2)};
                  // define o Serviço
                  Service calc_service = new Service();

                  // Objeto de Chamada
                  Call call = (Call) calc_service.createCall();

                  // Seta o caminho onde está localizado o Web
                  // service
                  call.setTargetEndpointAddress(urlWS);

                  // seta o método que será usado
                  call.setOperation("somar");
                  // Executa o método passando os parametros
                  // necessários
                  Integer result =
                        (Integer) call.invoke(parametros);

                  // Imprime o resultado da chamada acima
                  System.out.println("Resultador: " + result);

            } catch (Exception ex){
                  ex.printStackTrace();
            }

      }
      public static void main(String[] args) {
            new Calc_cliente();
      }
```

O código apresentado acima é muito semelhante ao código do Exemplo 5.1. Temos basicamente a mesma estrutura, o que é simplificado é que a quantidade de pacotes necessários, de serem importados, é bem menor, bastando importar o pacote *Client* da API do AXIS, com os objetos deste pacote é simples e fácil, a implementação de um cliente para um determinado Serviço *Web*. Um ponto notável com relação às diferenças entre os dois "clientes" é a forma de codificação com relação aos parâmetros.

Foram mostrados alguns bons motivos para se utilizar o AXIS para publicar os Serviços *Web* e uma forma simples de implementar o cliente para estes serviços, assim termina, sem o objetivo de ter esgotado o assunto, o capitulo referente a tecnologias e ferramentas Java para a implementação de Serviços *Web*. O próximo capítulo apresentado tratará da implementação do WEBCALC um Serviços *Web* implementado em Java sobre o AXIS com a finalidade de prestar serviços de cálculo de Imposto de Renda e INSS, mostrando a empregabilidade de todos os conceitos até aqui apresentados.

6 IMPLEMENTAÇÃO DO PROJETO WEBCALC

Quando se iniciaram as pesquisas sobre um tema para apresentar como projeto de conclusão de curso, vários temas poderiam ter sido abordados: Redes Neurais, Algoritmos Genéticos, Telecomunicação, Robótica, Engenharia de *Software* dentre outros. A decisão em fazer um projeto sobre sistemas distribuídos Orientados a Serviços (Serviços *Web*), vários fatores foram decisivos. Sendo o principal motivo uma tendência de mercado que está encaminhado cada vez mais para os sistemas distribuídos, e corporativos. O nome WEBCALC deriva das palavras *Web* e Cálculos, sendo que a parte de cálculos aqui será compreendida na área dos cálculos legais, focando nosso objetivo em Cálculo de IR (Imposto de Renda) e Cálculo de INSS (Imposto Nacional de Seguridade Social). Os cálculos apresentados no decorrer deste capítulo são regras determinadas pelas Leis Federais sobre tais impostos. Para que seja possível fornecer tais cálculos um Serviço *Web* será criado no decorrer do capítulo e ao término da implementação do serviço implementaremos um cliente para o mesmo. Lembrando que o desenvolvimento destes são totalmente baseados em Java adotando conceitos que foram apresentados nos capitulos anteriores.

6.1 Arquitetura WEBCALC

A criação do serviço está baseada em fornecer uma interface com duas funções publicas disponíveis para os Consumidores deste serviço; uma função para o cálculo do IR e outra para o cálculo do INSS. A Figura 12, abaixo, mostra a arquitetura do WEBCALC e como ele está organizado.

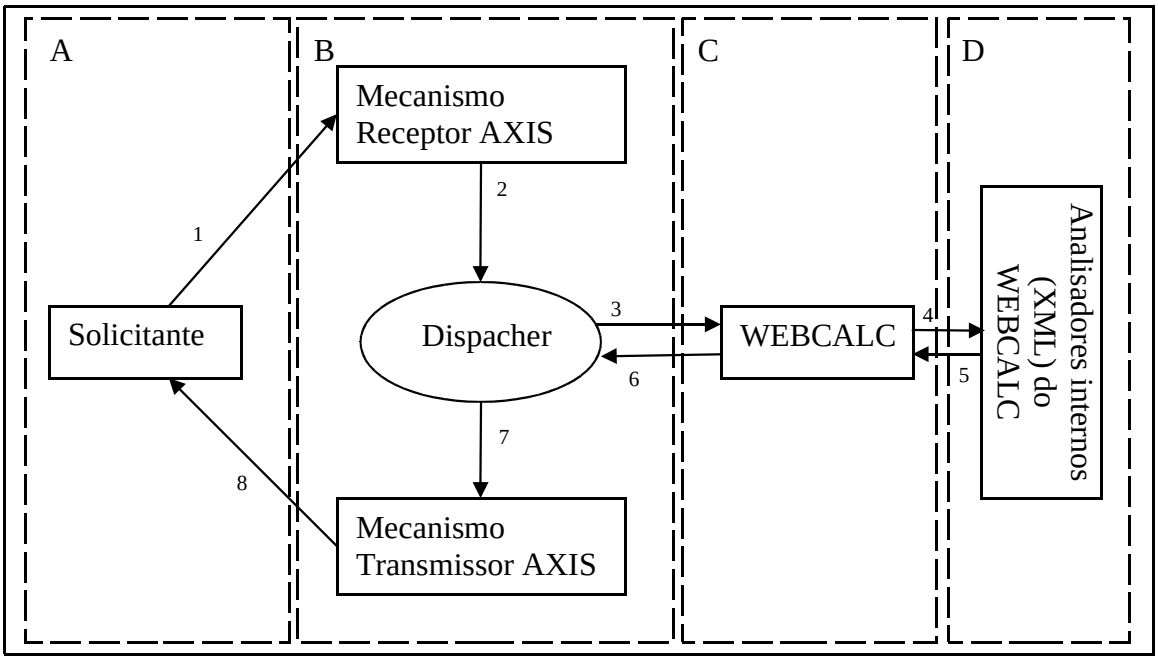


Figura 12 - Arquitetura do WEBCALC

Nessa figura (Figura 12) pode se observar a existência de uma divisão onde: A é o Consumidor do Serviço *Web* (WEBCALC), B é o mecanismo do AXIS responsável pelas rotinas *request/response* que permitem a comunicação entre Consumidor e Provedor do Serviço, C é a interface disponibilizada para execução (interfaces de IR e INSS) e finalmente D que é a camada da arquitetura que recupera informações necessárias para os cálculos serem realizados, essas informações poderiam estar em uma base de dados, então teríamos a camada de dados da arquitetura.

Dentro da arquitetura têm-se os seguintes papéis e funções:

Solicitante - o solicitante é o responsável por iniciar o ciclo de vida dessa arquitetura, pois ele faz uma requisição de serviço. Essa, como já visto é feita a través do SOAP sobre um protocolo de aplicação que no nosso caso será o HTTP.

Mecanismo de recepção AXIS - descrito anteriormente (na Figura 11 do Capitulo 5 a arquitetura do AXIS), alguns componentes do AXIS aqui são aplicado na prática, pois a requisição do Solicitante tem um caminho a percorrer, então o mecanismo de recepção é a "porta" de entrada dessa requisição. Uma atividade que também é de responsabilidade deste

mecanismo é a validação das Mensagens SOAP encaminhada pelo Solicitante que possuem um formato baseado no padrão XML, o que requer um "validador" para tal.

Dispacher - o *Dispacher* é o responsável pela localização do serviço, o Solicitante obrigatoriamente tem que informar um caminho e um nome do serviço que pretende executar, e para ser localizado entra em cena então o *Dispacher*. Dizemos que o *Dispacher* é mediador entre AXIS e o Serviço *Web*, pois a resposta do serviço também passará por este.

WEBCALC - Este é o Serviço *Web* que foi implementado e que estaremos descobrindo mais sobre ele no decorrer do capítulo.

Analisadores Interno (XML) – o intuito de utilizar este nome é que o WEBCALC trabalha com arquivos de dados que são armazenados em XML (*Data XML*), esses arquivos são tabelas de faixas de IR (MINISTÉRIO DA FAZENDA, 2004) e faixas de INSS (MINISTÉRIO DA PREVIDÊNCIA SOCIAL, 2004). Para que se possa recuperar as informações destes arquivos é necessário saber interpretar os XML que os compõem. Esta camada da aplicação poderia ser uma base de dados como SQL *Server* ou outra qualquer, mas como o intuito é trabalhar independente de plataforma ou linguagem de programação a opção foi pelos documentos XML. Mais adiante serão apresentados detalhes sobre estes analisadores.

A partir deste ponto, temos o serviço integrado à arquitetura proposta para os Serviços *Web* sobre o AXIS, pode-se então, começar a implementação do serviço, propriamente dito.

6.2 Provedor WEBCALC

Durante esta sessão será mostrado o processo que foi adotado para se implementar o serviço WEBCALC, como um passo inicial, um caso de uso para o serviço será apresentado. Trata-se de um cliente que está implementando uma ferramenta nova para a *Web* que necessita do calculo de imposto de renda, pois sua aplicação trabalha com um esquema semelhante a uma folha de pagamento, este por sinal decidiu não implementar tal funcionalidade e pesquisou por um Serviço *Web* em um diretório UDDI e encontrou o WEBCALC, (neste trabalho não foi

publicado o serviço em diretório UDDI, essa é uma situação ilustrativa). O caso de uso é mostrado abaixo na Figura 13.

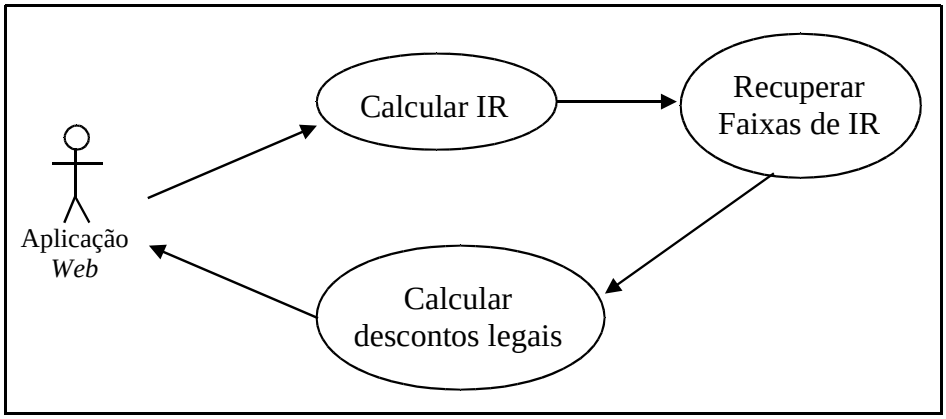


Figura 13 - Caso de uso: Calcular IR

Através do caso de uso apresentado na Figura 13, é simples perceber como pode ser utilizado o Serviço *Web*. Alguns diagramas da Engenharia de *Software* serão mostrados neste capitulo, porém sem o intuito de descrevê-los detalhadamente, o que sairia do escopo deste trabalho.

Para saber como seria a interação Solicitante/Serviço *Web* um diagrama de seqüência ilustraria melhor o ciclo de vida de uma requisição de um Solicitante. De forma a simplificar a explicação a Figura 14, mostra este diagrama.

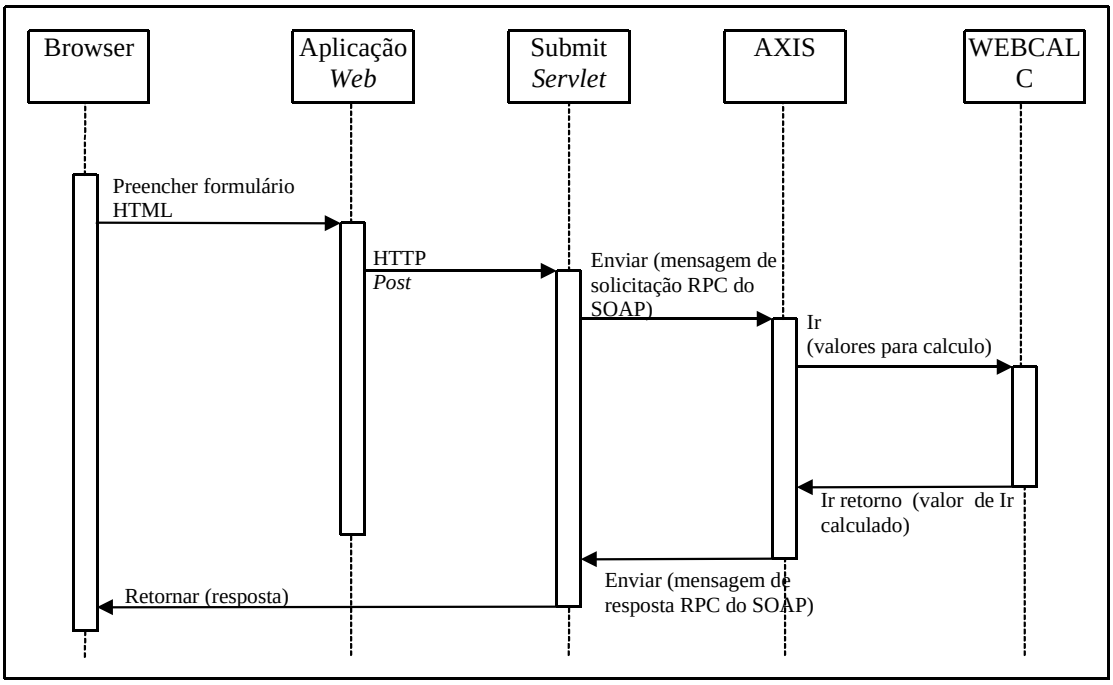


Figura 14 - Diagrama de Seqüência - Requisição

No diagrama de seqüência apresentado na Figura 14, o usuário final preenche um formulário HTML que é submetido para a Aplicação *Web*, que posta estas informações para um *servlet* denominado *SubmitServlet* que recupera os valores do formulário postado e assim constrói uma chamada para a API do AXIS tendo como parâmetro uma mensagem SOAP. O AXIS então prepara esta mensagem, localiza o serviço WEBCALC especificado e executa o método `Ir` que tem como parâmetro os valores necessários para o cálculo de IR; O WEBCALC efetua os cálculos necessários e retorna um valor de IR para o AXIS que se encarrega da formatação da mensagem SOAP e retorna para o Solicitante.

É bem simples o funcionamento dos Serviços *Web*, se comparado com outros sistemas distribuídos como CORBA e COM.

O próximo passo será demonstrar como foi implementado o serviço WEBCALC. Todo o processo foi feito em ambiente de teste, com dito anteriormente o serviço não foi publicado em um registro UDDI. Por tanto, foi necessário montar toda uma configuração para se fazer isso. A primeira coisa a se fazer é recuperar do CD, anexo, as ferramentas necessárias: Java J2SDK, o kit JWSDP que contém os pacotes e componentes necessários para se publicar um serviço *Web*, um *Web* container, esse se trata do Tomcat. Será necessária também uma versão do AXIS, e um componente chamado Xerces (opcional) para a interpretação dos documentos XML.

Primeiro as classes que tratam os arquivos XML, por motivo simplicidade, fora criada uma classe para cada arquivo XML a ser trabalhado.

Três arquivos XML: um para a tabela de IR, outro para a tabela de INSS e um com o valor a ser abatido por Dependente. Estes arquivos são encontrados no CD anexado a este Trabalho de Conclusão de Curso.

Será necessária a criação de uma pasta chamada WEBCALC que vai ficar na raiz do AXIS (Apêndice E – No CD), dentro desta pasta é que colocaremos nossas classes na estrutura que será definida mais adiante. Por se tratar de um desenvolvimento em Java, a opção adotada foi a criação de pacotes adotando a seguinte estrutura:

default: isto indica que o seu conteúdo não entra dentro de uma sub-estrutura ficando assim pertencente a raiz do projeto.

WEBCALC: (Serviço *Web* para o cálculo de IR e INSS)

faculdade.projetos.*Beans* : responsável pelos *Beans* de dados, sendo que estes representam um mapeamento dos arquivos XML citados anteriormente.

BeanDependentes (mapeamento do arquivo dependentes.xml);

BeanInss (mapeamento do arquivo inss.xml);

BeanIr (mapeamento do arquivo ir.xml);

faculdade.projeto.WEBCALC: neste pacote é que se encontra a interface padrão para qualquer usuário que queira implementar uma variação do WEBCALC que se baseia nesta interface.

WEBCALC_IR (Interface base para o *Web* Calc)

faculdade.projeto.xml: pacote destinado às classes responsáveis pelos analisadores de XML.

WC_xml_Deps (Analisador do documento dependentes.xml);

WC_xml_Inss (Analisador do documento inss.xml) ;

WC_xml_Ir (Analisador do documento ir.xml) ;

Na Figura 15 é apresentado o relacionamento entre estes pacotes:

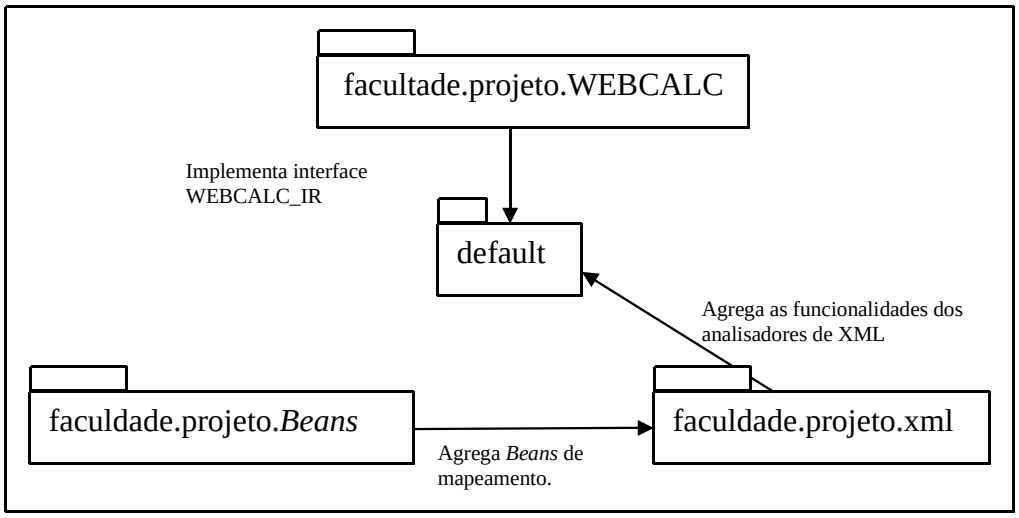


Figura 15 - Relacionamento entre os pacotes do WEBCALC

O centro *default* é onde está o serviço WebCalc.java que implementa a *interface* WEBCALC_IR, agrega os analisadores WC_xml_Deps.java, WC_xml_Inss e WC_xml_Ir. Para cada um destes analisadores é agregado o *Bean* correspondente: BeanDependentes, BeanInss e BeanIr respectivamente. O relacionamento entre estas classes é, melhor apresentada em um diagrama de classes que é mostrado na Figura 16.

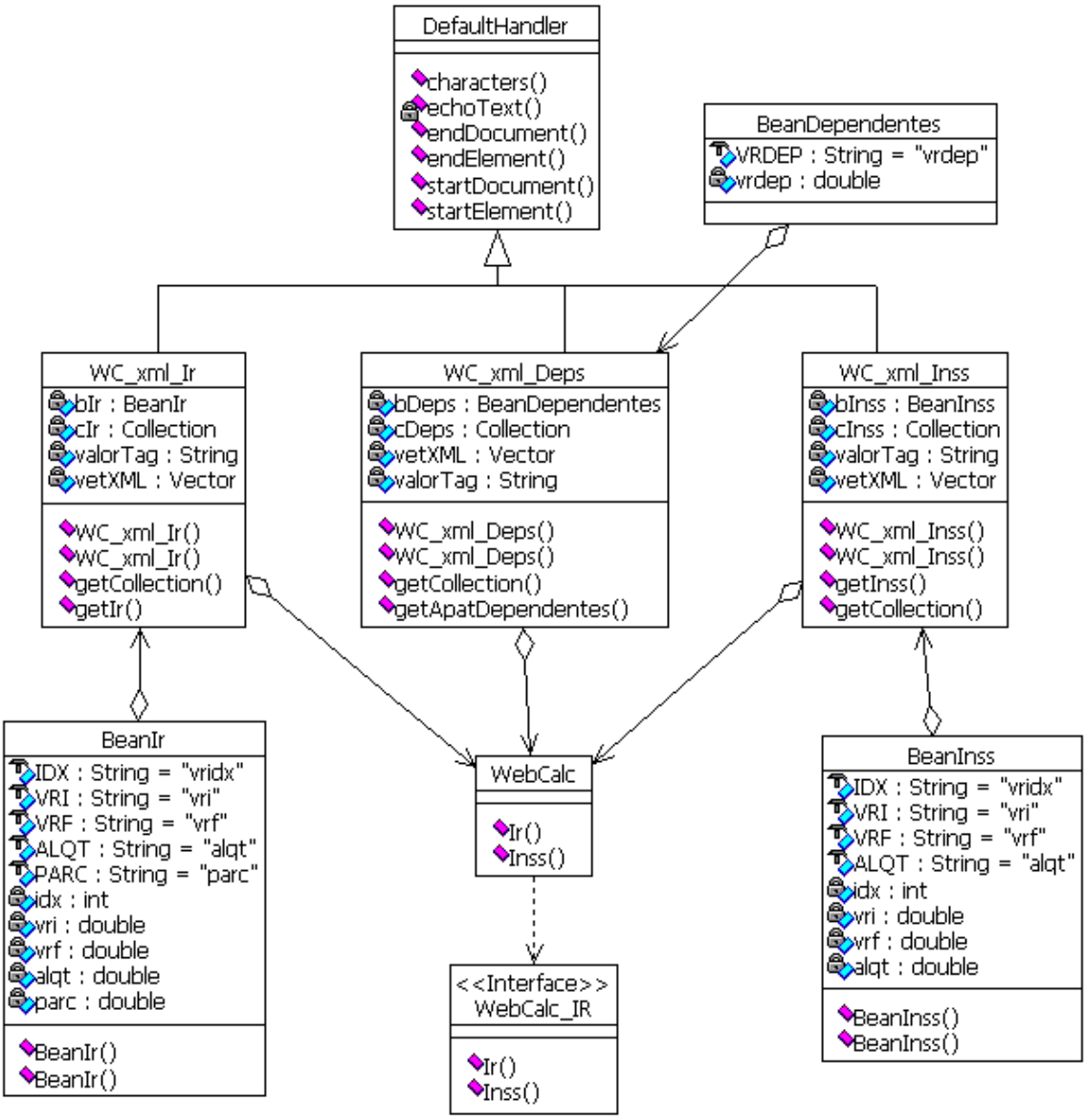


Figura 16 - Diagrama de classe do WEBCALC

O diagrama apresentado na Figura 16, mostra o relacionamento entre as classes criadas para o serviço WEBCALC. As classes de nomes precedidos por “WC”, herdam da classe DefaultHandler, que é uma classe do pacote SAX (org.xml.sax) que é nativa do Java 1.4, todas as características para um analisador XML, para uma adaptação ao desenvolvimento feito os métodos mostrados para a classe *DefaultHandler* são sobrescritos (Polimorfismo) de forma a dar um tratamento mais adequado na análise dos XMLs montando em tempo de análise as coleções[4]

[4] Entenda por coleções uma espécie de vetor que receberá os dados contidos no XML

necessárias. Os *Beans* são criados e agregados aos analisadores. Essa junção será agregada ao WEBCALC que disponibiliza dois métodos:

Ir que recebe como parâmetro (em ordem) rendimentos, valor pago a previdências privadas, número de dependentes e tipo de execução (anual = 0 ou mensal = 1) e retorna o valor do IR já calculado para o solicitante;

Inss que tem como parâmetro o valor de rendimentos e retorna o valor correspondente ao INSS encontrado para o valor informado.

Os códigos das classes apresentadas nesse diagrama (Figura 16), por questões simplesmente de organização e separação, estão em anexo n CD.

O cálculo de IR e do INSS são apresentados abaixo. Para esclarecimento estes cálculos são baseados em dados da Receita Federal (MINISTÉRIO DA FAZENDA, 2004) e da Previdência Social (MINISTÉRIO DA PREVIDÊNCIA SOCIAL, 2004). As tabelas de consulta podem sofrer alterações de um ano para outro ou conforme novas leis forem criadas. Um exemplo desta mudança, e que já é tratada pelo WEBCALC, é um abatimento de R$100,00 na base que é analisada junto à tabela de IR.

Tabela 6 - Formulas para os cálculos do WEBCALC

INSS	IR
O cálculo do INSS é baseado totalmente em uma tabela do Governo que define faixas. Para cada uma dessas faixas existe um percentual correspondente. Em caso do valor passado como parâmetro for maior que o teto definido na tabela, o desconto é feito com a porcentagem desta faixa sobre o teto e não sobre o valor parâmetro.	Para se calcular IR alguns pontos extras são levados em consideração. Existe também uma tabela que possui faixas (hoje 15% e 27,5%) a diferença entre a tabela de INSS e a de IR é que a de IR contém um valor chamado Parcela de Abatimento no IR que será abatido do valor de IR calculado. Esta tabela é consultada com o valor base com parâmetro, conhecido como Base de IR A formula de cálculo de IR é: IR = (R – D) * P – PA onde R corresponde ao parâmetro rendimentos; D é um conjunto de deduções legais (Valor pago ao INSS +

	Valor pago a previdências privadas + Numero de dependente * valor por dependente e a partir de agosto de 2004 + 100,00); P é o percentual da tabela de IR em que foi enquadrado o Valor Base e PA é valor da parcela a deduzir.

O cálculo de IR é bastante complexo quando se está fazendo-o, por exemplo, para um sistema de folha de pagamento, aonde um demonstrativo de IR retido na fonte deve ser mantido no processo. Assim se trabalha com valores já abatidos e bases anteriores em um cálculo contínuo mês-a-mês. Algumas outras deduções podem ser efetuadas. Para saber quais são consulte o *site* da receita federal.

6.3 Consumidor para o WEBCALC

Para que o serviço tenha alguma finalidade um Consumidor deve implementar um cliente, neste caso já temos o cliente pronto. Foi apresentado no Capítulo 5 (Exemplo 5.2) sobre o cliente de um Serviço *Web* publicado no AXIS, este exemplo é perfeitamente aceito para o WEBCALC, bastando efetuar algumas mudanças simples que são apresentadas abaixo (Mudanças apenas de parâmetros – Obs: os valores informados como parâmetros são fictícios e podem ser qualquer valor):

De:

```
String urlWS =
 "http://localhost:8080/axis/Calculadora.jws";
```

Para:

```
String urlWS =
 "http://localhost:8080/WEBCALC/WEBCALC.jws";
```

De:

```
Object[] parametros = {new Integer(3), new Integer(2)};
```

Para:

```
Object[] parametros = {new Double(102960.00), new
   Double(794.02), new Integer(2), new
         Integer(1)};
```

De:

```
call.setOperation("somar");
```

Para:

```
call.setOperation("Ir");
```

De:

```
Integer result = (Integer) call.invoke(parametros);
```

Para:

```
Double result = (Double) call.invoke(parametros);
```

Desta forma é só iniciar o Tomcat e executar o Cliente para ver o resultado.

6.4 Considerações Finais

Seguindo uma linha simples de desenvolvimento foram constatados dois pontos com relação ao AXIS considerados interessantes:

- O desempenho que em teoria é melhor que o Apache SOAP, parece, não ser tão nítida na prática, pois os exemplos executados sobre o Apache SOAP mostraram melhores desempenhos com relação ao AXIS. O cenário e massa de testes não foram tão significativos, pois apenas um exemplo foi comparado
- A tecnologia empregada ainda tem muito para evoluir, pois as ferramentas possuem configurações “escondidas” que não foram encontradas na documentação, por exemplo: os arquivos XML usados pelo serviço WEBCALC. Estes arquivos devem estar no Servidor e em teoria na mesma estrutura de diretórios, porém, quando on-line as classes apontam para a pasta classes dentro da pasta *WEB-INF* da raiz do AXIS, além de outros assuntos que foram de difícil acesso.

Com a implementação do WEBCALC foi necessário um estudo detalhado sobre IR e algumas formas de cálculo que puderam ser utilizadas. A ferramenta Java mostrou-se

bastante eficaz, atendendo todas as expectativas e necessidades para o assunto Serviços *Web*, além de se mostrar bastante flexível quanto à aplicação da teoria.

Não foi criado nenhum *Software* com a intenção de que ele seja comercial ou que possua uma interface com o usuário. O WEBCALC foi uma implementação de um serviço, portanto, sem interface com o usuário (humano). O Consumidor do WEBCALC poderia ter sido implementado utilizando-se de uma interface para o usuário, porém o objetivo maior foi mostrar a utilização dos serviços oferecidos pelo WEBCALC, valores fixos foram usados para exemplificar as chamadas realizadas ao serviço de forma a exemplificar o funcionamento.

O termo “Sistemas distribuídos” foi adotado no sentido de que é possível criar uma aplicação com partes integrantes distribuídas geograficamente e logicamente em várias partes não tendo uma ligação direta com balanceamento de carga, por exemplo. Vale frisar que embora não abordado o assunto o SOAP também permite a adoção de servidores de distribuição de carga não deixando nada a desejar para o demais modelos e padrões para sistemas distribuídos.

7 CONCLUSÃO

A combinação de tecnologias mostrou-se possível no decorrer dos estudos e se faz muito simples após ter o conhecimento necessário. A utilização de Java como linguagem de programação para este trabalho, definitivamente foi a escolha mais acertada, pois, oferece uma alta aplicabilidade dos conceitos aprendidos tanto durante o curso quanto durante os estudos "extraclasse" realizados. A matéria de redes auxiliada pela Engenharia de *Software* foi muito útil para a utilização dos protocolos e para um compreendimento maior sobre como funciona realmente os Serviços *Web*. O aproveitamento em um projeto como esses é sempre muito grande e neste caso abriu-se um leque para utilizações de tecnologias novas e inteiramente "*open-source*".

Um *tour* pelos capítulos pôde mostrar que o pensamento foi evolutivo e que um conhecimento uniforme sobre o assunto pôde ser adquirido.

7.1 Trabalhos futuros

Como trabalho futuros, vários pontos poderiam ser abordados, dentre eles:

1. Implementar um Consumidor para o WEBCALC utilizando outra linguagem de programação que não Java;
2. Disponibilizar o WEBCALC utilizando-se o Apache SOAP como base de publicação;

3. Publicar o WEBCALC em diretório UDDI para que este esteja disponível para outros interessados e assim se faça um estudo sobre o funcionamento dos diretórios UDDIs;
4. Efetuar um estudo detalhado nos padrões de comunicação baseados em XML;
5. Aplicar algumas das tecnologias de segurança apresentadas no Capítulo 4.

8 - BIBLIOGRÁFIAS

BRYAN, Douglas; *et al*. **UDDI *Version* 2.04 API *Especification***, *Organization for Structured Information Standards* (OASIS). Disponível em: <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm> Acesso em: 05 ago 2004.

CHRISTENSEN, Erik *et. al*. ***Web* Services *Description Language* (WSDL) 1.1** *World Wide Web Consortium* (W3C). Disponível em: <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>. Acesso em: 30 out 2004.

DEITEL H. M.; DEITEL P.J. **Java Como Programar.** 3. ed. Porto Alegre: Bookman, 2001. 1200p.

GERAGHTY, Ronan. **COM-CORBA *Interoperability*.** Sanddler River: Prentice Hall PTR, 1999.

GETTYS, Jim, **HTTP: *Hipertext Transfer Protocol*.** *World Wide Web Consortium* (W3C). Disponível em: <www.w3c.com.br/protocolos/rfc2616>. Acesso em: 20 set 2004.

GUDGIN, Martin et. al. ***Simple Object Access Protocol* (SOAP) 1.2**. *World Wid Web Consortium* (W3C). Disponível em: <http://www.w3.org/TR/2003/REC-soap12-part0-20030624/> Acesso em 10 set 2004.

HENDRICKS, Mack, *et al*. ***Java Web* Services**, Rio de Janeiro: Alta Books, 2002.

IETF, The Internet Engineering Task Force, **RFC 793,** 1981, 84p. Disponível em: <http://www.ietf.org/rfc/rfc0793.txt> Acesso em: 20 out 2004.

_____ **RFC 2821,** 2001, 78p. Disponível em: <http://www.ietf.org/rfc/rfc2821.txt>, Acesso em: 28 nov 2004.

JENDROCK, Eric, *et al*. ***The Java Web* Services Tutorial.** Santa Clara, Califórnia-USA: Sun Microsystems. 2003. 1660p.

MICROSOFT, **Building XML-Based *Web* Applications.** Argentina, 2001 (Série Microsoft Official Curriculum)

MINISTÉRIO DA FAZENDA. **Tabela Progressiva para Cálculo anual do Imposto de Renda de Pessoa Física**. Net 2004. Disponível em: <http://www.receita.fazenda.gov.br/Aliquotas/TabProgressiva.htm> Acesso em: 20 maio 2004.

MINISTÉRIO DA PREVIDÊNCIA SOCIAL: **Tabelas de Contribuições mensal.** Net 2004 Disponível em: <http://www.previdenciasocial.gov.br/03_01_04.asp> Acesso em 20 maio 2004

ORCHART, David, *et all*. ***Web Service Architecture*.** *World Wide Web Consortium (W3C) working Group Note*, Net 2004. Disponível em: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>. Acesso em: 20 maio 2004.

PAUL, V. Biron; ASHOK, Malhotra. ***XML Schema part 2: DataTypes***. *World Wide Web Consortium (W3C) working Group Note*, Net 2001. Disponível em: <http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/> Acesso em: 20 jul 2004.

PROCWORK, **Java *Web* Development**, 2004, 366p. Apostila – Procwork Tecnologia, São Paulo. 2004.

POSTEL, J. ***File Transfer Protocol (FTP)***. *World Wide Web Consortium (W3C)*, Net 1985. Disponível em: <http://www.w3.org/Protocols/rfc959/4_FileTransfer.html> Acesso em: 15 ago 2004.

REAGLE, Joseph. ***XML Encryption Requerements.*** *World Wide Web Consortium (W3C)*, Net 2002. Disponível em: <http://www.w3.org/TR/xml-encryption-req> Acesso em: 30 out 2004.

TINDALL, David. **Desenvolvendo Aplicações Corporativas.** Rio de Janeiro: Campus, 2000. 400p.

WEISSINGER, A. Keyton. **ASP – Guia Completo**. Rio de Janeiro: Ciência Moderna, 1999.

ANEXO 1 - CD DE APOIO PRÁTICO AO MATERIAL