ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

SERVIÇO DE DISTRIBUIÇÃO DE CONTEÚDOS RSS PARA DISPOSITIVOS MÓVEIS ATRAVÉS DE WEB SERVICES

Trabalho de Conclusão de Curso

Engenharia da Computação

Rafael Bandeira de Oliveira
Orientador: Prof. Dr. Sérgio Castelo Branco Soares

Recife, dezembro de 2005

UNIVERSIDADE
DE PERNAMBUCO

ESCOLA POLITÉCNICA
DE PERNAMBUCO

Departamento de
Sistemas
Computacionais

SERVIÇO DE DISTRIBUIÇÃO DE CONTEÚDOS RSS PARA DISPOSITIVOS MÓVEIS ATRAVÉS DE WEB SERVICES

Trabalho de Conclusão de Curso

Engenharia da Computação

Este Projeto é apresentado como requisito parcial para obtenção do diploma de Bacharel em Engenharia da Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Rafael Bandeira de Oliveira
Orientador: Prof. Dr. Sérgio Castelo Branco Soares

Recife, dezembro de 2005

UNIVERSIDADE
DE PERNAMBUCO

Rafael Bandeira de Oliveira

SERVIÇO DE DISTRIBUIÇÃO DE CONTEÚDOS RSS PARA DISPOSITIVOS MÓVEIS ATRAVÉS DE WEB SERVICES

Resumo

Este trabalho apresenta um estudo sobre dispositivos móveis e a tecnologia Java, especificamente na plataforma J2ME, interagindo com um servidor *web* através de um *Web Service*. O principal objetivo deste estudo é especificar e implementar um serviço de busca, para posterior leitura, de informações contidas em documentos RSS disponíveis publicamente na Internet. Um segundo objetivo é permitir o envio de informações para um arquivo RSS, realizada pelo usuário do aplicativo a partir de um arquivo de propriedade dele. Esse serviço foi dividido em dois sistemas, um cliente e um servidor, caracterizando uma arquitetura cliente-servidor. O software cliente foi desenvolvido em J2ME e é responsável por fazer requisições ao *Web Service* disponível no servidor e exibir os dados recebidos na tela do dispositivo. O servidor de aplicação disponibiliza o serviço, o qual coleta as informações contidas nos arquivos RSS solicitados, processa as informações encontradas, e as retorna ao cliente. A transmissão dessas informações é realizada através de trocas de mensagens entre o cliente e o servidor, utilizando protolocos pré-definidos quando tratamos de *Web Services*. O estudo demonstra uma inovação na maneira de integração entre dispositivos móveis e servidores de aplicações, explicando como as tecnologias encontradas em cada sistema são interligadas, demonstrando ser uma solução viável.

Palavras chave: Dispositivos Móveis; Web Services; RSS

Abstract

This work presents a study on mobile devices and the Java technology, specifically in the J2ME platform, interacting with a server web through a Web Service. The objective of this study is to specify and to implement a search service for reading information contained in available documents RSS public in the internet. A second function is to allow sending information into an RSS file, which is made by the application user using an archive of his property. This service was splited into two systems: one as a client and the other as a server, complying with a client-server architecture. The client software was developed in J2ME and is responsible for making requests to the Web Service available in the server and for presenting the data received in the device screen. The server application makes available the service to collect the information in requested RSS files, it processes the information returning them to the customer. The transmission of this information is carried through messages exchanges between the client and the server using specific protocols to deal with Web Services. The study demonstrates an innovation in the way mobile and applications servers integrate also explaining how such technologies are linked, also demonstrating to be a viable solution.

Keywords: Mobile devices; Web Services; RSS

Sumário

Índice de Figuras

Índice de Tabelas

Tabela de Símbolos e Siglas

API	Application Programming Interface
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
DOM	Document Object Model
GPL	General Public License
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transport Protocol
JAR	Java Arquive
JVM	Java Virtual Machine
J2EE	Java 2 EnterpriseEdition
J2ME	Java 2 Micro Edition
J2SE	Java 2 Standard Edition
KVM	Kilo Virtual Machine
MIDP	Mobile Information Device Profile
MVC	Model-View-Controller
PC	Personal Computer
PDA	Personal Digital Assistants
RPC	Remote Procedure Call
RSS	Rich Site Summary
SAX	Simple API for XML
SMS	Short message service
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
UDDI	Universal Discovery, Description, and Integration
URL	Uniform Resource Locator
WAP	Wireless Application Protocol
WML	Wireless Markup Language
WSDL	Web Service Description Language
XML	Extensible Markup Language
W3C	World Wide Web Consortium

Agradecimentos

Gostaria de agradecer a Deus por tudo que tem me proporcionado; a fé persistente, a saúde forte, os desejos alcançados e o sonho ainda melhor do futuro.

Agradeço ao meu pai Luiz Alberto e a minha mãe Teresinha Bandeira, cuja educação e caráter me ensinam até hoje. Aos meus irmãos, Isabella e Rodrigo Bandeira, pelas experiências do dia-a-dia. E a todos os meus familiares pelas ausências que tive que fazer.

Aos professores da Escola Politécnica de Pernambuco que se esforçam o melhor de si para manter e melhorar este curso, em especial Ricardo Massa e Carlos Alexandre pelos ensinamentos aprendidos. Ao meu professor orientador Sergio Soares pelo tempo e correções necessários. Ao professor Emanuel Leite pelo apoio e crença dados na iniciativa de seguir com minha empresa.

Aos meus amigos da Poli, em particular, os amigos e engenheiros da 1ª Turma de Engenharia da Computação formada, "Os cobaias da Poli". Aos amigos que seguiram outros caminhos, mas que são ainda presentes nas nossas lembranças, aos que, simplesmente, foram especiais e aos que darão vida a novas esperanças no mundo, aqui representados por Kíssia Cavalcanti, Aretha Maria, Aline Senna, Lívia Brito, Reinaldo Melo e Rodrigo Brayner.

Ao amigo e sócio Rodrigo Cursino por todos os obstáculos enfrentados e recompensas alcançadas.

Aos demais amigos, que mesmo longe da faculdade se fazem presentes na minha vida.

E agradeço especialmente a minha namorada Greice Paola, por tudo.

Capítulo 1

Introdução

Através das tecnologias sem fio, qualquer pessoa tem hoje a possibilidade de acessar informações importantes para o seu dia-a-dia, de qualquer lugar que disponha de um serviço de comunicação - como um *shopping center*, aeroporto, loja de conveniência, ou mesmo da rua; e a qualquer momento que se queira obter tais informações. Para isso utilizam-se aparelhos capazes de se comunicar com tais tecnologias, seja um *notebook*, PDA, ou celular; como menciona Genilson César [11] em seu artigo. Tais informações estão accessíveis na rede mundial de computadores, a Internet, nos mais diferentes formatos, como texto, imagens, áudio ou vídeo. Essas podem ser encontradas desde em *sites* pessoais, como diários virtuais (conhecidos por *blogs*), álbuns virtuais (conhecidos por *fotoblogs*), *sites* institucionais, sejam privados ou públicos, *sites* jornalísticos especializados em dispor os mais novos fatos (notícias) do Brasil ou do mundo, praticamente no momento em que eles acontecem.

Em particular, os *sites* jornalísticos diferenciam-se dos demais *web sites* por terem no seu dinamismo o fator decisivo para o interesse dos usuários. Esse dinamismo é caracterizado pela rápida atualização das informações contidas na página *web* e pelas informações ditas "em primeira mão", as quais, até aquele determinado momento, não haviam sido publicadas em qualquer outro *site*. Logo, para os usuários que buscam por informações novas, de fácil acesso e precisas, esse dinamismo e a correta publicação dessas, tornam-se características importantes na escolha de qual *web site* acessar em busca das notícias que possam interessá-los.

De fato, o que acontece no dia-a-dia é um internauta ter não apenas um *site* de notícias preferido, mas sim, vários; além de ter também alguns *blogs* prediletos. Desta forma, para saber se existe ou não novas informações de seu interesse, o usuário acessa todos os seus *sites/blogs* prediletos em busca de alguma atualização pertinente, o que demanda um tempo razoavelmente grande para essa tarefa. Percebendo essa dificuldade e pensando no gasto de tempo de busca e visualização dessas notícias, em meados do ano de 1999, foi criado um arquivo de texto padronizado, no qual estaria um resumo das últimas notícias inseridas no *web site/blog*.

Nesse arquivo, encontram-se informações sobre o *web site*, como seu nome, uma descrição de seus serviços e *link* para o mesmo, assim como, uma listagem contento o título, uma pequena descrição e um *link*, das últimas notícias/matérias/informações inseridas na respectiva página *web*. O *link,* nesse caso, dá acesso ao conteúdo completo da notícia desejada. Dessa forma, os usuários podem acessar esse resumo disponibilizado por alguns *web sites*, para saber quais são, se existirem, as informações mais atuais desde sua última visita. Assim, o usuário não perde mais tempo à procura por informações em seus *sites* preferidos; basta agora, acessar esse resumo

e ter de forma clara, simples e objetiva uma listagem das últimas notícias cadastradas nos mesmos.

Esse resumo encontrado nos *web sites* é um arquivo de texto conhecido pela sigla RSS (*Rich Site Summary*) [3] disponível abertamente por eles. Entraremos em mais detalhes nesse arquivo posteriormente. O arquivo RSS tem sua estrutura padronizada, ou seja, tem sempre a mesma estrutura independente das informações contidas nele, o que o torna uma referência para a criação de sumários de informações em páginas *web* de forma segura, rápida e simples. Essa estrutura padrão é baseada na tecnologia XML (*Extensible Markup Language*) [47].

Dessa forma, através desse arquivo, o usuário tem como permanecer atualizado com as principais notícias de seu interesse, não mais acessando diretamente o *site* e seus canais favoritos, mas agora, simplesmente acessando esse arquivo disponibilizado pela página *web* e verificando se foi inserida alguma nova notícia ou não. Esse trabalho de verificação pode ser feito através de *softwares* específicos chamados de 'Agregadores', capazes de verificar centenas desses arquivos em poucos segundos e informar na tela do programa, quais notícias e de quais *sites* são os mais recentes. Para saber mais sobre os Agregadores, veja [18].

Toda essa facilidade é encontrada para usuários de computadores pessoais, os chamados PCs, que possuem alta capacidade de processamento da máquina, ampla tela de visualização das informações colhidas, e que, por muitas vezes, estão conectados 24 horas por dia na Internet. Por outro lado, para os usuários de aparelhos móveis, como celulares ou PDAs, terem acesso a essas mesmas informações, os celulares, por exemplo, têm que contar com o serviço que sua operadora oferece, normalmente através de trocas de mensagens SMS (*Short Message Service*), o qual (serviço) normalmente não é muito agradável e tem seu conteúdo muito restrito e curto. Outra opção seria buscar as notícias acessando a Internet diretamente pelo celular, onde a navegabilidade e usabilidade não são as ideais devido a restrições de hardware do celular – como *display* pequeno – e pela pouca aceitação da tecnologia WAP (*Wireless Application Protocol*) [10] pelos *web sites* de notícias. Pois ainda hoje o padrão HTML (*Hyper Text Markup Language*) [28] da Internet tem que ser adaptado ao padrão requerido pelo WAP (WML - *Website META Language* [10]), o que torna o acesso a *web sites* pelo dispositivo, restrito e pouco usual.

Já os aparelhos *Personal Digital Assistant*, conhecidos por PDAs, tem uma facilidade maior de buscar informações devido ao fato de poderem se conectar ao computador e baixar as notícias dos sites da Internet por programas específicos como o Hands [27], tornando mais ágil, rápido e em maior quantidade as informações procuradas. Contudo, nem sempre se tem em mãos um computador ligado à internet ou a interface requerida para conexão entre o PC e o aparelho, ou mesmo o *software* adequado instalado na máquina para fazer tal integração.

Portanto, para se ter acesso as notícias de *web sites* na Internet por um dispositivo móvel, o dispositivo tem que acessar o serviço que a operadora de telefonia celular lhe dispõe, que é restrito e com poucas informações, ou, conectar o dispositivo a um computador e, a partir desse, busca pelas informações desejadas, o que é um processo demorado e que nem sempre se tem disponibilidade de fazê-lo.

Tendo essas características em mente, surgiram no mercado vários *softwares* e serviços direcionados para tais dispositivos, capazes de buscar pelos arquivos de resumo (RSS) dos *web sites* desejados de forma peculiar para cada programa, funcionando de forma semelhante aos já existentes Agregadores para PCs. Um deles, já mencionado, é o Hands, que se assemelha a um outro *software* chamado, JPluck [29]. Para se ter acesso às informações contidas nos arquivos RSS, precisa-se, em ambos os casos, conectar o PDA ao computador, e com uma conexão à Internet disponível, o software realiza uma busca pelas informações, o qual converte e comprime as informações encontradas e, uma vez o processo realizado, as informações são transferidas para o dispositivo através do *HotSync* (sincronismo) de forma que se possa ter as informações para leitura posterior. Uma visão dessa solução é apresentada na Figura 1.

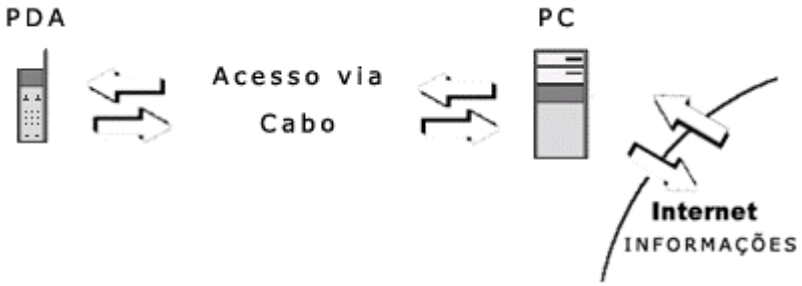


Figura 1. Visão geral do software JPlucker e Hands

A principal diferença entre os softwares é que o Hands além de se conectar com o PC para colher as informações desejadas, pode também 'baixar' as informações diretamente por uma conexão Internet do próprio PDA. Isso facilita o acesso aos arquivos RSS e aumenta a facilidade de aprendizado pelo usuário. Contudo, uma desvantagem de ambos é o fato de ter o processamento das informações colhidas da Internet a cargo do próprio dispositivo, acarretando um trabalho de processamento maior por parte desse e estar disponível apenas para PDAs.

Com o mesmo objetivo, existem também serviços na Internet que se dispõem em realizar tal tarefa. Podemos destacar o MobileRSS [33] por estar disponível há mais tempo. Criado em meados de julho de 2003, esse serviço traz uma maneira diferente de buscar as informações nos arquivos RSS: ao invés de instalar um programa no PDA, o serviço MobileRSS disponibiliza as informações colhidas diretamente numa página *web* dentro do seu próprio *web site*. Essa página *web* atualiza a lista de RSS que o usuário cadastrou previamente e, quando acessada, busca as informações mais atuais naquele momento; esse acesso é feito diretamente pelo *browser* do dispositivo, incluindo aqui, aparelhos celulares.

Esse serviço é gratuito, porém é necessário um registro prévio no site do mesmo, para definir ou editar quais documentos RSS são de interesse do usuário. Também é necessário ter instalado no dispositivo móvel um *browser* para visualizar páginas na Internet, uma vez que a visualização de seus arquivos prediletos é feita *on-line* diretamente no *site* do serviço. Por esse motivo, torna-se uma desvantagem, pois é necessário estar conectado à Internet tempo suficiente para ler todas as informações desejadas; também encontramos dificuldades em não poder definir quais sites RSS escolher para fazer uma busca mais especifica e restrita (todos os *sites* cadastrados na área restrita do usuário são usados na busca). Apesar de um funcionamento simples, encontramos dificuldades em simular um acesso através de um aparelho celular, mesmo seguindo todos os passos indicados no *site* do serviço, fato que impossibilitou uma abordagem mais detalhada.

Vislumbrando os cenários supracitados, este trabalho tem como objetivo disponibilizar um *Web Services* que busque informações em documentos RSS para dispositivos móveis, em especial, celulares. Para isso, será apresentado um estudo sobre dispositivos móveis e a tecnologia Java, especificamente na plataforma J2ME, interagindo com a tecnologia de *Web Services* com o fim de especificar e implementar um *software* cliente-servidor. A arquitetura do projeto proposto tem a visão geral mostrada na Figura 2.

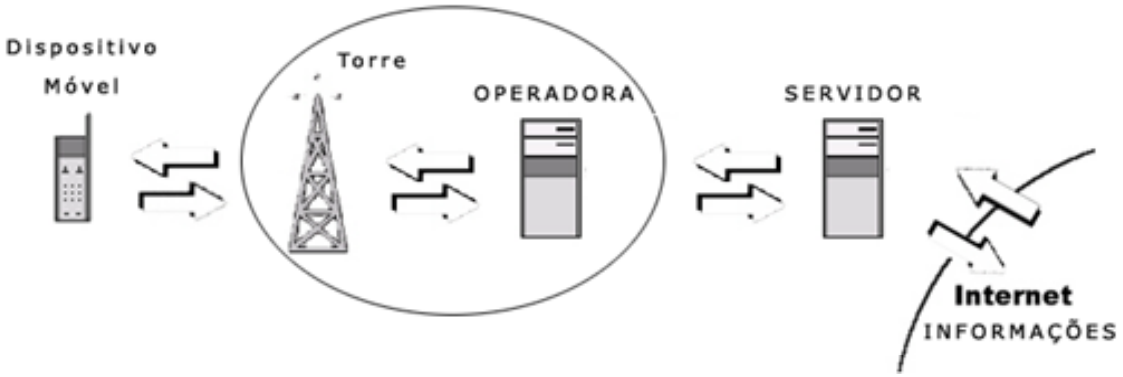


Figura 2. Visão geral da arquitetura proposta

A parte de transmissão das informações pela operadora de telefonia móvel, não será tratada no projeto, uma vez que há muitos detalhes da forma e das tecnologias envolvidas nesse processo.

O aplicativo cliente foi desenvolvido na plataforma J2ME [6][9] voltado em especial para aparelhos celulares que faz requisições e recebe respostas de um *Web Service*. O aplicativo servidor será um *Web Service* [2][7] disponível num servidor *web*. Esse serviço tem como finalidade buscar, processar e retornar as informações contidas nos documentos RSS publicamente disponíveis na Internet. Para tanto, um estudo detalhado nessa tecnologia também é abordada neste trabalho.

O software sugerido dá ao usuário duas formas de obter acesso às informações desejadas, contidas em arquivos RSS:

- Na primeira delas, o cliente faz uma requisição ao servidor por uma listagem que contém os últimos *web sites* que dispõem suas informações no formato RSS, conhecidos por fornecedores. O servidor tem uma listagem de fornecedores que podem ser atualizados, até então, de forma manual. Ao retornar essa listagem ao dispositivo móvel, o usuário pode escolher dentre os fornecedores existentes, de quais deseja receber informações. Uma vez feita a seleção, uma nova requisição será feita ao servidor, e esse, através da preferência escolhida pelo usuário, fará uma busca pelas informações desejadas - arquivos RSS dos fornecedores escolhidos. Uma vez terminada a busca, o servidor pré-processa os dados colhidos e envia as informações pertinentes ao dispositivo móvel para visualização pelo usuário.
- Na segunda forma, o próprio usuário do *software*, cadastra quais fornecedores RSS específicos tem interesse em obter informações, dando ao usuário flexibilidade de obter informações que são de seu interesse e não estão contidas na listagem do servidor. O único detalhe nessa forma, é que o endereço completo do arquivo RSS deve ser conhecido pelo usuário, algo como, *www.fornecedor.com.br/esporte/resumo.rss*, exigindo um pouco mais de conhecimento por parte do usuário do aplicativo.

Resumindo, o que este trabalho propõe é fazer um estudo na tecnologia RSS utilizado como fonte de informações de um *Web Service*, que se comunica com um aplicativo num dispositivo móvel, mostrando as tecnologias envolvidas nesse processo e como é possível tal integração.

A razão de escolher como cliente um dispositivo móvel deve-se ao fato de termos poucos softwares específicos para buscar informações nos aparelhos celulares e para permanecer proporcionando mobilidade ao usuário do mesmo. Já a escolha de um *Web Service* como

aplicativo servidor para dispor tais buscas e processamentos, deve-se à questão de torná-lo um serviço accessível publicamente para que outros sistemas também possam acessá-lo. Já a tecnologia RSS foi utilizada por se tratar de um padrão já aceito e usado mundialmente para trocas de resumos de informações.

Esta monografia está organizada da seguinte forma. No Capítulo 2 são apresentados alguns conceitos sobre as tecnologias abordadas no trabalho, incluindo os softwares do cliente e do servidor utilizados no desenvolvimento da aplicação. A integração da plataforma J2ME com o servidor *Web Services* é apresentada no Capítulo 3. O Capítulo 4 aborda a arquitetura do sistema, a metodologia adotada, diagramas de classes e de casos de uso, assim como a interface gráfica gerada pelo sistema. E Capítulo 5, serão expostas as conclusões finais sobre este trabalho e os trabalhos que podem ser realizados a partir desta monografia.

Capítulo 2

Conceitos Fundamentais

Para um melhor aproveitamento deste trabalho, este capítulo aborda conceitos sobre as tecnologias utilizadas ao longo do desenvolvimento da aplicação.

2.1 XML

Extensible Markup Language, que em português corresponde a "Linguagem de Marcação Expansível", ou simplesmente XML [47], é um formato de texto simples, muito flexível, que provê uma representação estruturada dos dados.

O contexto de XML no projeto é compreendido nas próximas Seções, quando falarmos das demais tecnologias utilizadas tendo como base o formato XML, por isso a necessidade de conhecer e detalhar essa tecnologia nesse momento.

2.1.1 História

O XML é derivado da linguagem SGML - *Standard Generalized Markup Language* - ISO 8879 [34] e da linguagem HTML [28]. Essa última ganhou popularidade e força com o advento e avanço da Internet. Já o SGML mostrou qualidade e força industrial inerente ao formato estruturado em árvore dessa linguagem, em aplicações industriais. A principal diferença do SGML para o XML é que esse último foi elaborado para distribuição através da *web* A diferença para o HTML é que o XML é extensível, formalmente estruturado e possui poucos elementos pré-definidos.

A especificação do formato XML foi definida pelo consórcio W3C - *Word Wide Web Consortium*, a qual assegura que os dados estruturados serão uniformes e independentes de sistemas tanto em *hardware* quanto em *software*. O XML é um formato padrão que pode exemplificar o conteúdo, as semânticas e os esquemas de uma enorme variação de sistemas desde os mais simples até os mais complexos.

Uma característica importante é que uma vez que o dado seja recebido pelo cliente, o mesmo pode ser manipulado, alterado e visualizado sem a necessidade de acionar novamente o servidor que o enviou. Dessa forma, os servidores têm menor sobrecarga de requisições, o que

reduz o processamento da máquina, reduzindo também o tráfego de dados pela rede para as comunicações entre cliente e servidor.

2.1.2 Exemplo

De forma a esclarecer quaisquer dúvidas quanto ao formato e estrutura, vamos, através de um exemplo, mostrar detalhadamente as características de um documento XML. A Figura 3 mostra um exemplo de seu formato.

```
1  <?xml version="1.0"
2    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
3  ?>
4  <book>
5    <title>Learning XML</title>
6    <authors>
7        <author city="Cidade1" type="xsd:string">
8                Nome do 1° autor </author>
9        <author city="Cidade2" " type="xsd:string">
10               Nome do 2° autor </author>
11    </authors>
12  </book>
```

Figura 3. Exemplo de formato XML

Como se pode ver, um arquivo XML é muito semelhante a um arquivo HTML. Assim como no HTML, o XML também possui elementos marcadores (tags). Na quarta linha, por exemplo, temos o início de um novo elemento (*tag)* `<book>` e a linha doze indica o fim desse elemento, indicado pela inserção do símbolo da barra (/), `</book>`. De forma geral, os elementos que dão início a uma *tag* e o seu fechamento, são chamados de *start tag* e *end tag*, respectivamente. Entretanto, o XML possui algumas diferenças básicas como, por exemplo:

- XML não possui um conjunto de *tags* definidas pela sua especificação. As *tags* são criadas e definidas de acordo com a aplicação desejada;
- Todo documento XML possui uma *tag* raiz. No caso da Figura 3, a raiz é a *tag* `<book>`;
- Além disso, não são permitidas *tags* mal formadas, isto é, todas as *tags* devem estar presentes no documento aos pares - um indicando seu inicio e outra *tag* indicando seu fim; também não deve haver violações na hierarquia das *tags*.

A primeira linha do documento é a definição do documento XML. Nela, definimos a versão do XML que estamos usando. A segunda linha mostra a referência para um *namespace* que está definido fora do documento. A implicação disso no documento é explicada mais claramente na próxima Seção.

Dentro da *tag* `<book>`, encontramos mais dois elementos, ou filhos, `<title>` e `<authors>`, linhas 5 a 11; e para cada autor, temos um atributo `city`. Claramente podemos perceber que se trata de um arquivo XML que define um determinado livro, contendo seu título, seus autores, e a cidade dos autores.

Nos documentos XML, os atributos são freqüentemente usados para metadados (i.e. informações sobre os dados), descrevendo propriedades para seus índices. No nosso exemplo, a propriedade `city`, na *tag* `<author>`, contém a cidade de cada autor referenciado.

O usuário da linguagem XML pode construir suas próprias representações, criando nomes

de elementos (*tags*) e atributos específicos a sua necessidade. E como seu principal objetivo é a estruturação de dados, o usuário pode criar múltiplas formas de visualização desses dados estruturados.

2.1.3 Namespaces

Na segunda linha da Figura 3, vimos que existe uma definição de um *namespace* no arquivo XML exemplificado, e uma referência a ele visto nos tipos dos elementos `<author>`.

Namespaces [48] provê um método para evitar conflito de nomes (elementos, atributos e tipos) dentro de um documento XML. Como as representações de nomes não são fixados neste tipo de documento, frequentemente, há um conflito de nomes quando dois diferentes documentos usam os mesmos nomes descrevendo dois diferentes tipos de elementos. Vejamos a Figura 4 para entender melhor essa situação.

<book> <title>About XML</title> <authors> <author city="Cidade1" > Nome do 1° autor </author> <author city="Cidade2" > Nome do 2° autor </author> </authors> </book>	<book title="About XML"> <author city="Cidade1" > Nome do 1° autor </author> <author city="Cidade2" > Nome do 2° autor </author> </book>
(a)	(b)

Figura 4. Exemplos de arquivos XML

Pela Figura 4, vemos que ambos os documentos são similares, contendo inclusive vários elementos definidos com o mesmo nome, mas definem uma estrutura totalmente diferente entre si. No caso de juntarmos esses arquivos em um único documento, como poderíamos distinguir os elementos que estão definidos com o mesmo nome (o que nos exemplifica um conflito)? A resposta para essa questão está nos *namespaces*.

Com o uso de *namespace*, podemos definir, ou reutilizar, uma referência única e distinta para ambos os documentos – um prefixo – de forma a serem identificados sem conflito. A inserção de *namespace* num documento XML é feita através do atributo padrão `xmlns` dentro do elemento desejado. Essa situação pode ser vislumbrada na Figura 5.

```
<x:book xmlns:x="http://www.livrariaA.com/ns/book">
 <x:title>About XML</x:title>
 <x:authors>
   <x:author city="Cidade1" >
     Nome do 1° autor </x:author>
   <x:author city="Cidade2" >
     Nome do 2° autor </x:author>
 </x:authors>
</x:book>
<y:book title="About XML" xmlns:y="http://www.livrariaB.com/ns/book">
   <y:author city="Cidade1" >
     Nome do 1° autor </y:author>
   <y:author city="Cidade2" >
     Nome do 2° autor </y:author>
</y:book>
```

Figura 5. Exemplo de arquivo XML usando namespaces

Com os prefixos `x` e `y` percebemos claramente que foram criados dois diferentes tipos de elementos `<book>`. Com isso, podemos identificar que autor, de que livro e de qual livraria estamos tratando. Quando um prefixo é definido no início da *tag* como um elemento, todos os elementos filhos com o mesmo prefixo são associados com o mesmo *namespace*.

Note que o endereço usado para identificar o *namespace*, não é o usado para validar o documento XML. O único propósito é dar ao *namespace* um único nome. Contudo, frequentemente ele é usado como ponteiro para uma página *web* contendo informação sobre o *namespace*.

Os *namespaces* não são usados para identificar apenas os elementos de um arquivo XML, como vimos, mas podem também identificar os atributos ou mesmo os tipos dos mesmos. Com isso podemos definir um *namespace* que restringe um tipo *string*, por exemplo, em ter no máximo 30 caracteres. No caso da Figura 3, vemos esse exemplo, onde o tipo do elemento `<author>` é `xsd:string`, que referencia o *namespace* definido no início do documento. Essa atribuição define que esse tipo *string* deve se comportar da forma definida no referido *namespace*, pois, dessa maneira, torna o tipo do elemento `<author>` padrão, de forma a ficar transparente na troca de dados com outros sistemas. Isso é útil, pois impede que cada proprietário de arquivo XML defina um tipo específico para sua aplicação, dificultando a compatibilidade com outros sistemas.

Com todas as características supracitadas em mente, a tecnologia XML tornou-se rapidamente um padrão para troca de informações via Internet, devido a sua simplicidade de uso e implementação. Atualmente, existem bibliotecas para manipulação de XML disponíveis para as principais linguagens de programação, além de diversos produtos e ferramentas.

No caso específico de Java para XML, há dois tipos de bibliotecas bastante utilizadas: SAX [39] e DOM [25]. Um *parser* SAX é uma classe Java que lê um arquivo XML e executa comandos conforme as *tags* são encontradas. Para cada tipo de *tag*, há um código associado. Já um *parser* DOM lê um arquivo XML e converte-o para uma árvore de objetos Java, que podem ser facilmente manipulados. Entraremos em mais detalhes nessas bibliotecas no decorrer do trabalho quando for preciso ler documentos XML.

2.2 RSS

Originalmente desenvolvido pela Netscape em 1999, RSS (que pode ser traduzido para "Distribuição Realmente Simples de Conteúdo", "Sumário de Site Enriquecido" ou ainda, "Sumário de Site RDF") é um formato de arquivo baseado em XML que permite a desenvolvedores *web* descreverem e disponibilizarem sumários para páginas *web*.

Nessa Seção discutiremos brevemente a definição de documentos RSS no escopo da Internet, o porquê de torná-los públicos, a história do RSS, as características das versões existentes, exemplos e o contexto deste padrão no objetivo do trabalho.

2.2.1 O que são documentos RSS?

Documentos RSS nada mais são do que arquivos de texto contendo parte ou o conjunto de resumos das matérias publicamente disponíveis em um *web site* para uso por outros serviços. Não necessariamente, o conjunto de resumos são todos os resumos existentes no *web site*, mas com certeza, sempre são aqueles mais atuais.

Cada resumo encontrado no arquivo RSS pode ser chamado também de sumário e consiste numa lista de títulos e descrições desses títulos, conhecidos também por índices e metadados, respectivamente. Logo, o fato de tornar um arquivo RSS público e disponível na Internet, se dá o nome de "Distribuir Índices", ou *Syndicated Content* em inglês, e o *web site* que o está publicando é conhecido por "Alimentador", ou *Feed*, na língua inglesa.

O *Feed* pode criar sumários para assuntos específicos dentro do próprio *web site*, ou para assuntos que estejam disponíveis em outros *web sites* da Internet. O formato padrão mais comum para construir esses sumários na Internet é conhecido pela sigla RSS, baseado em XML.

A Figura 6 mostra um trecho de código de um arquivo, de como seria a distribuição de índices usando o padrão RSS. Os detalhes do arquivo serão explicados posteriormente, mas podemos perceber desde já a facilidade de compreender as informações contidas neste arquivo, herdada das próprias características do formato XML.

```
1  <rss version="XXX">
2
3  <channel>
4  <title>Title of the Channel</title>
5  <link>http://www.channelweb.com/</link>
6  <description>A Web-based Feed RSS</description>
7  <language>en-us</language>
8
9  <image>
10 <title>Logo Channel!</title>
11 <url>http://www.channelweb.com/icons/powered.gif</url>
12 <link>http://www.channelweb.com/</link>
13 </image>
14
15 <item>
16 <title>The First Item</title>
17 <link>http://www.channelweb.com/example/001.html</link>
18 <description>This is the first item.</description>
19 </item>
20
21 <item>
22 <title>The Second Item</title>
23 <link>http://www.channelweb.com/example/002.html</link>
24 <description>This is the second item.</description>
25 </item>
26
27 </channel>
28 </rss>
```

Figura 6. Trecho de código de um arquivo RSS

O RSS facilita a vida do usuário ao dispensá-lo de ter que acessar, diariamente, todos os seus sites, blogs e serviços *on-line* favoritos, pois ele recebe o sumário de informações (arquivos RSS) atualizado de tempos em tempos e confere apenas os índices (notícias) que foram inseridas desde sua última visualização. Atualmente, essa é a maneira mais prática de manter-se atualizado por meio da Internet.

Além disso, o RSS está sendo considerado uma alternativa aos boletins (*newsletter*), que necessitam de preenchimento de formulários na *Web* e podem favorecer a proliferação de *spam*. Dessa forma, os desenvolvedores de *web sites* podem usar RSS para recolher e distribuir publicamente sumários de informações atuais deles ou de parceiros, e o usuário, por sua vez, pode usar aplicativos específicos, chamados Agregadores, para coletar e processar as informações contidas nos arquivos RSS de seus sites preferidos, disponibilizando o material encontrado de

forma prática e mais rápida do que pela maneira tradicional de acessar diretamente o *web site* em busca de informações.

Por que disponibilizar RSS?

As vantagens de distribuir sumários em *web sites* são as mais variadas possíveis, pois atendem, principalmente, na fidelização dos usuários do *web site* e segundo Ben Hammersley [3], podemos ainda destacar:

- O aumento do tráfego de acesso ao *web site*, tanto ao arquivo RSS, quanto às páginas das matérias completas encontradas no mesmo;
- Pode ajudar no aumento de *rankings* nos sistemas de sites de busca;
- Melhora o relacionamento entre o *web site* e seus usuários - o usuário passa a confiar e dar mais credibilidade ao *site*;
- Ajuda a fortalecer o relacionamento com outras comunidades *on-line*;
- Com o avanço da tecnologia, pode-se sugerir diversas aplicações para seu uso, como notificações de novas notícias dentro de seu programa de trocas de mensagem instantânea favorito, dentre outras.

2.2.2 História e Versões

Após seis anos desde sua criação, podemos perceber que o RSS é o padrão mais aceito e divulgado para trocas de sumários na Internet, mas ainda não possui um formato único para descrevê-lo. Hoje temos duas vertentes de como escrever arquivos RSS, existem vantagens e desvantagens para ambas, e ainda existe o estudo de um terceiro formato RSS que visa unificar as melhores qualidades dos outros dois, para, finalmente, vir a se tornar o padrão para descrever arquivos RSS.

Podemos destacar como principal diferença entre as duas vertentes, o fato de uma seguir uma linha mais objetiva e simples na criação de arquivos RSS, pois acredita que a simplicidade dá aos usuários uma adaptação mais fácil e rápida para criarem arquivos RSS. Enquanto a segunda, segue um raciocínio no qual acredita que uma complexidade maior no formato RSS com características mais robustas e a utilização de módulos adicionais ao formato, é o melhor caminho para descrever sumários na Internet, pois acredita que assim terão condições de contemplar um número maior de resoluções de problemas.

As versões de RSS que se enquadram na primeira vertente são: 0.91, 0.92 e 2.0. Já a segunda vertente é encabeçada pela versão 1.0 do formato RSS. No Apêndice D há maiores detalhes sobre essas vertentes, o surgimento das várias versões e detalhes mais técnicos dessas versões de RSS disponíveis hoje na Internet.

2.2.3 Abordagem RSS no Projeto

Diante do exposto, podemos dizer que RSS é um arquivo texto, escrito no formato XML que contém, além de outras informações, títulos de notícias, trechos dessas notícias e seus respectivos *links* para a notícia completa, além de detalhes do próprio *web site* que está disponibilizando tais informações.

Como a ênfase dada no projeto nos arquivos RSS é de buscar as informações do *web site* que está disponibilizando o documento, e nas informações que o mesmo está publicando, os

elementos que estaremos tratando durante todo o desenvolvimento serão os que contêm o título da informação, trecho do mesmo e link para o conteúdo completo dentro de cada versão RSS disponível, respectivamente, *title*, *description* e *link*.

Apesar de várias versões disponíveis para descrever arquivos RSS, há uma igualdade na estrutura hierárquica básica de todas, em que o elemento raiz *channel* possui sempre como filhos *title*, *link* e *description* que se referem a detalhes do *feed* (alimentador) que está disponibilizando as informações contidas nos elementos *item*. E esse último, também sempre contém os elementos básicos *title*, *link* e *description* – que se referem a detalhes da notícia em questão contida num determinado elemento *item*. Este cenário pode ser vislumbrado na Figura 7.

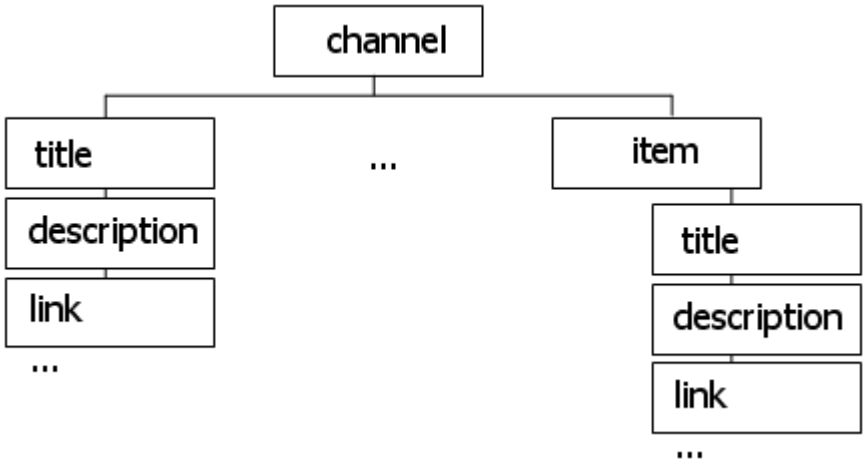


Figura 7. Estrutura hierárquica básica de um documento RSS

Dessa forma, independentemente da versão RSS que estaremos tratando futuramente, todas as informações que estamos procurando, sempre serão encontradas nos arquivos através dos mesmos elementos. Assim, qualquer elemento diferente dos supracitados será ignorado quando estivermos analisando um documento RSS, como por exemplo, o elemento `image` mostrado na Figura 6 anteriormente apresentada.

Perceba que a funcionalidade por trás do estudo das tecnologias deste projeto, é buscar por notícias e informações que estejam publicamente disponíveis em *web sites*. Para isso foi escolhido o formato RSS, por se tratar de "um padrão" aceito mundialmente para essas trocas de informações, nos reservando o trabalho apenas a encontrar o *link* (url) para tais arquivos RSS e o seu processamento, nos retorna a um mesmo processo de um arquivo XML.

2.3 Web Services

Nesta Seção trataremos dos conceitos, tecnologias e arquitetura abordadas quando trabalhamos com *Web Services*, ou "Serviços Web" – em português. A Seção define o que são *Web Services*, seu funcionamento, seu modelo, sua pilha básica e as tecnologias utilizadas.

2.3.1 O que são Web Services?

A W3C define *Web Services* como "*The World Wide Web is more and more used for application to application communication. The programmatic interfaces made available are referred to as Web services*". Dessa forma, podemos entender que *Web Services* é um termo dado a um serviço

disponibilizado através de uma aplicação servidora para ser acessado por sistemas clientes. Esses clientes podem ser aplicações que acessam esse serviço através da própria máquina em que se encontra o serviço, ou aplicações que estejam acessando o serviço através de uma rede interna ou externa caso o serviço em questão esteja sendo disponibilizado pela intranet ou internet, respectivamente.

Note que a definição para *Web Services* é bastante abrangente, pois tem por objetivo integrar sistemas distintos através de uma rede usando protocolos padronizados de forma independente de *hardware* e *software* em que esses sistemas estão inseridos. Dessa forma, é possível disponibilizar um conjunto de métodos em um servidor e permitir que sejam acessados por diversos aplicativos clientes [15]. Para isso, o cliente precisa conhecer a URL do servidor, o método disponível e os tipos de dados que são usados na chamada e na resposta desse método. Contudo, o cliente não precisa saber se o serviço foi construído em Java e se está sendo executado em Linux, ou se é um serviço desenvolvido em ASP.NET que é executado no Windows [4].

Todo sucesso e aceitação dos *Web Services* estão justamente na sua padronização, a qual permite que sistemas escritos em linguagens e plataformas diferentes possam fazer o intercâmbio de dados [15].

2.3.2 Funcionamento

Um *Web Service* funciona com um cliente enviando uma requisição para o *Web Service* através de uma URL usando um protocolo. O *Web Service* recebe essa requisição, processa-a, e retorna uma resposta ao cliente. Esse esquema pode ser visto na Figura 8.

Figura 8. Esquema de funcionamento de um *Web Service*

Mas, para que esse processo funcione, temos que encontrar o *Web Service*, passar os argumentos da forma correta, conhecer e esperar a resposta do serviço para que assim, os sistemas se comuniquem corretamente. Para entender como isso é feito, observaremos o modelo conceitual.

Modelo conceitual de Web Services

O modelo apresentado destaca dois elementos importantes de qualquer sistema: papéis e operações. Por papéis, entendemos os diferentes tipos de entidades; as operações representam as funções executadas por essas entidades para que o *Web Service* possa funcionar [4]. A Figura 9 mostra o diagrama de modelo.

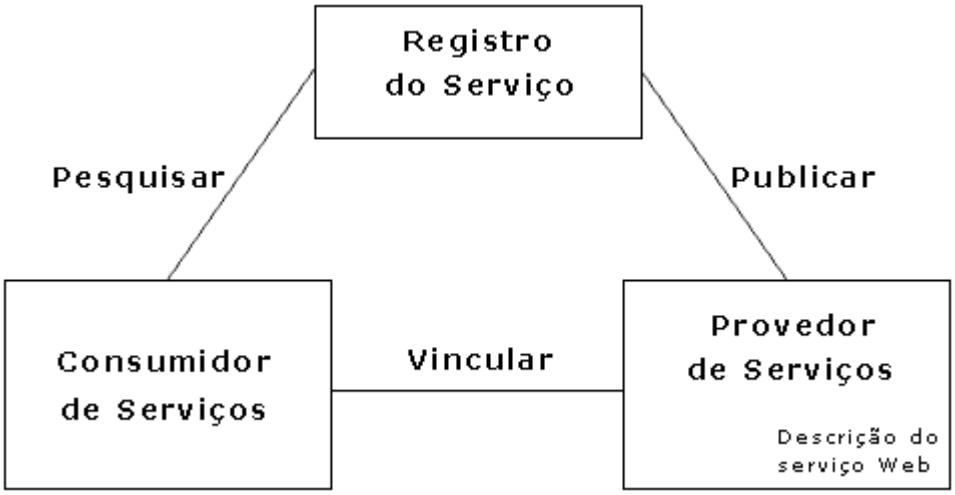


Figura 9. Diagrama de modelo conceitual de *Web Services*

Na Figura 9, podemos identificar três tipos de papéis em um ambiente de *Web Services* típico – representado pelos retângulos – bem como as operações que eles executam – representado pelas linhas. Segundo Mack Hendricks et al. [4], podemos descrever os papéis mostrados no diagrama como:

- **Provedor de Serviços** – O provedor de serviços é a entidade que cria o *Web Service*. Geralmente, a entidade apresenta alguma funcionalidade em que dispõe como um *Web Service*, para que seja encontrada e chamada por algum outro sistema de outra entidade. Como exemplo, temos o caso onde um vendedor de livros *on-line* deseja apresentar seu sistema de pedidos *on-line* como um *Web Service*. Esse vendedor precisa fazer essencialmente duas coisas: Primeiro, precisa **descrever** o serviço em um formato padrão, que seja compreensível por qualquer empresa que possa vir a usar esse serviço, chamado Descrição do *Web Service*. E em segundo, para alcançar um público maior, o provedor de serviços deve **publicar** os detalhes sobre seu serviço em um registro central que esteja publicamente disponível para todos os interessados.

- **Consumidor de Serviços** – Qualquer entidade que utilize um *Web Service* criado e publicado por um provedor de serviços é chamada de consumidor de serviços. O consumidor pode conhecer a funcionalidade de um serviço, a partir da descrição disponibilizada pelo provedor. Para recuperar tais detalhes, o consumidor realiza uma **pesquisa** sobre as informações onde o provedor publicou a descrição do serviço e o próprio serviço. O mais importante é que o consumidor pode obter, a partir da descrição, todos os detalhes para **vínculo** com o serviço do provedor, podendo então chamar esse *Web Service*.

- **Registro de serviços** – Um registro de serviços é a localização central onde o provedor pode relacionar seus *Web Services* que deseja disponibilizar, e no qual um consumidor pode pesquisar por serviços disponíveis. Tipicamente, informações como detalhes da empresa, serviços por ela fornecidos e detalhes sobre cada serviço, inclusive detalhes técnicos, são armazenadas no registro do serviço.

Na Figura 9, podemos ainda identificar as três operações que são fundamentais para o funcionamento dos *Web Services* – pesquisa, vínculo e publicação. Essas operações precisam ser

padronizadas para que qualquer aplicação, independente da plataforma em que esteja sendo executada e da linguagem em que foi escrita, possa se comunicar com qualquer serviço sempre da mesma forma.

Na próxima Seção, veremos quais são esses padrões no contexto da arquitetura de *Web Services* e suas características.

2.3.3 Tecnologias

Na Seção anterior, mostramos as operações e papéis fundamentais, em um ambiente de *Web Services* através de um modelo conceitual. Para que isso se torne possível, faz-se necessário o uso de diferentes tecnologias para que os *Web Services* possam se comunicar, além de uma forma de organização entre elas. A Figura 10 mostra uma pilha básica de camadas de um *Web Service* e como elas estão organizadas.

Publicação e descoberta do serviço
Descrição do serviço
Mensagem XML
Protocolo de Rede

Figura 10. Pilha básica de *Web Services*

Observe que as camadas da Figura 10 são conceituais e cada uma das camadas disponibiliza serviços para a camada imediatamente abaixo dela. As tecnologias e padrões reais encontrados em cada uma delas são apresentados a seguir.

Protocolo de Rede

Essa primeira camada da pilha é responsável pela disponibilização dos *Web Services*, tornando-os acessíveis por intermédio de algum dos protocolos de transporte, como HTTP, SMTP, FTP e outros. Vale lembrar, que os *Web Services* são desenvolvidos com base em padrões já existentes, por isso se tornam independentes. No cenário atual, o HTTP tornou-se o protocolo de comunicação mais amplamente utilizado, principalmente com a ajuda da Internet, e a maioria dos navegadores *web* suportam esse protocolo.

Como vimos, os *Web Services* podem ser disponibilizados através de redes internas ou externas. Caso sejam implementados para acessos via Internet, é recomendado que se escolha o protocolo de rede HTTP, por já ser largamente difundido e aceito; já em caso de se implementar dentro de uma empresa, pode-se escolher protocolos diferentes, como o *Messaging Standars* – MS, como sugere Hendricks [4].

No nosso projeto, o protocolo de rede utilizado é o HTTP, uma vez que transmitiremos as informações pela Internet e será publicamente disponível para outros sistemas.

Mensagem XML

A camada seguinte da pilha é a troca de mensagens XML. Essa camada define o formato de mensagem usado na comunicação entre aplicações. O padrão usado com freqüência pelos *Web Services* é o *Simple Object Access Protocol* – SOAP [40].

Alguns sistemas distribuídos utilizam protocolos como CORBA, RMI e DCOM, que usam formatos binários para se comunicar com outras aplicações. Em vez disso, o SOAP se comunica com sistemas distribuídos usando o protocolo XML que é criado num arquivo texto.

Essa característica torna o SOAP altamente transparente, podendo trafegar por meio de várias plataformas de hardware, sistemas operacionais, linguagens de programação e protocolos de rede, por não ser um padrão específico para um determinado grupo de sistemas, o que permite que se beneficie dos investimentos em infra-estrutura, como servidores *web*, de *proxy* e *firewalls* [4].

É importante mencionar, que o SOAP não foi o primeiro padrão para computação distribuída, mas foi o primeiro a receber apoio dos principais fornecedores de *software* e *hardware*. Dessa forma, tornaram-no o protocolo mais aceito e divulgado para trocas de mensagens em aplicações distribuídas.

- **SOAP**

A especificação SOAP consiste em duas partes: encapsulamento de **mensagens** e de **Chamada de Procedimento Remoto ou** "**Remote Procedure Call (RPC)**", em inglês**.** A primeira parte consiste num *framework* para a transmissão de mensagens entre sistemas distribuídos, a segunda define como embutir chamadas e respostas de procedimento remoto dentro das mensagens.

A parte de especificação relativa ao encapsulamento permite que o SOAP forneça uma abstração na camada de trocas de dados. A abstração é proporcionada pelo fato do desenvolvedor mapear os dados de uma linguagem de programação para SOAP e vice-versa. Por isso, essa abstração fornece que sistemas remotos tenham acesso a objetos através de plataformas, sistemas operacionais e ambientes, que poderiam ser até incompatíveis [4]. Por exemplo, uma camada de SOAP pode ser adicionada a um objeto JAVA e J2ME. Cada camada cria um ambiente de comutação distribuído comum para os dois objetos – tornando os objetos do JAVA acessíveis pelos objetos J2ME e vice-versa, sem que o usuário ou programador precise se preocupar com as características de cada objeto e o ambiente em que estão inseridos (veja a Figura 11).

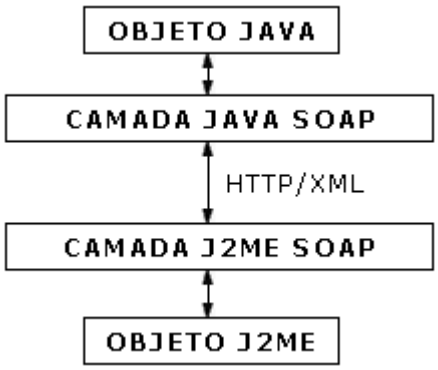


Figura 11. Diagrama de encapsulamento de objetos

- **Mensagem SOAP**

Para entender o SOAP um pouco melhor, vejamos uma mensagem simples em SOAP conceitualmente representada conforme a Figura 12.

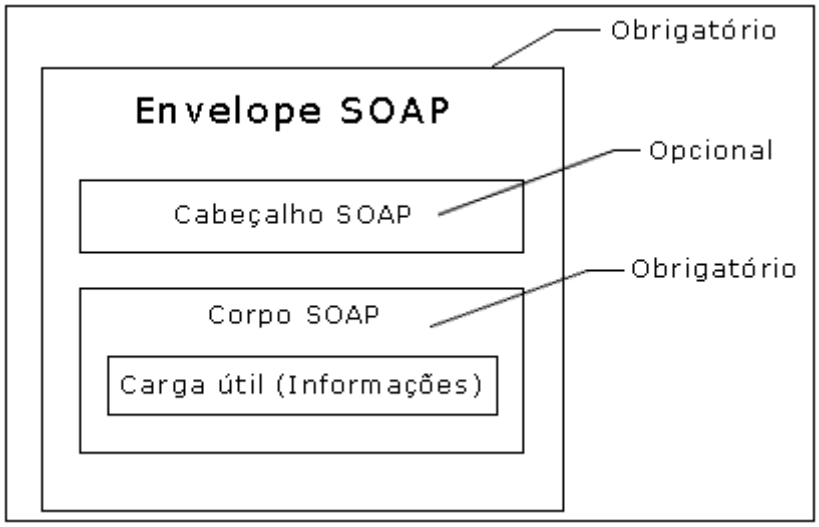


Figura 12. Representação conceitual de mensagem SOAP

Uma mensagem SOAP é um documento XML que pode conter, no máximo, três partes: Envelope, Cabeçalho e Corpo. Como visto na Figura 12, o Envelope (*Envelope)* é obrigatório e define a estrutura para a composição da mensagem e a forma que ela deve ser processada. O Cabeçalho (*Header*) é opcional e pode ser usado pelo usuário que está criando a mensagem, para fornecer quaisquer dados adicionais necessários. Já o Corpo (Body), também obrigatório, contém a carga efetiva da mensagem, que pode conter a codificação da chamada para algum procedimento, as respostas da chamada do procedimento, ou mesmo, o relatório de falhas, caso existam. A Figura 13 mostra um trecho do código de uma mensagem SOAP.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:SOAP-ENV=”http://schemas.xmlsoap.org/soap/envelope/”
>
<SOAP-ENV:Header >
      … //Opcional
</SOAP-ENV:Header >
 <SOAP-ENV:Body >
      ... //informacoes
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figura 13. Trecho de código de mensagem SOAP

Observe na Figura 13 que existe a definição de dois *namespace* (`xmlns`) no documento. Um referencia os dados de codificação (*SOAP-ENC*) e o outro as informações contidas no envelope do arquivo transmitido (*SOAP-ENV*). Dessa forma, todos os elementos do SOAP estão associados a um dos prefixos para garantir que os elementos vinculados a um dos *namespace* não entrem em conflito com elementos que possam ser definidos pelo usuário.

Descrição do Serviço

A terceira camada encontrada na pilha básica de um *Web Service*, é a camada de Descrição do serviço. Ela fornece um mecanismo ao provedor de serviços, a fim de descrever a funcionalidade

encontrada no serviço que está sendo disponibilizado. A *Web Service Description Language* – WSDL [46] é um formato padrão que se baseia em XML para descrever *Web Services*.

Em geral, as ferramentas de desenvolvimento criam automaticamente os arquivos WSDL, restando pouco trabalho ao desenvolvedor quanto a esse aspecto, o que será mostrado posteriormente.

- **Componentes WSDL**

Basicamente, o documento WSDL contém:

- ***Types***: Define os tipos de dados trocados;
- ***Messages***: Definem os parâmetros de entrada e saída de um serviço;
- ***Operation***: Define mensagens solicitação-resposta, usadas pelas operações (métodos) oferecidos por um serviço;
- ***PortTypes***: Encapsula uma coleção de operações;
- ***Bindings***: Descreve como um tipo de porta é mapeado para um protocolo de chamada de rede, como o SOAP, HTTP ou MIME;
- ***Service*** e ***Port***: incluem o nome do serviço e sua localização de implementação.

A Figura 14 resume os principais componentes que podem ocorrer em um documento WSDL.

```
<definitions>
<types>
      Tipos de dados utilizados
</types>
<message>
      Parâmetros de entrada e saída do serviço
</message>
<portType>
      <operation>
            O relacionamento entre parâmetros de entrada e saída
      </operation>
</portType>
<binding>
      Descrição do protocolo de rede utilizado
</binding>
<service>
      <port>
            Referência para o endereço atual do serviço
      </port>
</service>
</definitions>
```

Figura 14. Diagrama dos principais componentes WSDL

Publicação e descoberta do serviço

A última camada da pilha básica de um *Web Services* a ser explicada é a que disponibiliza a publicação e descoberta de um *Web Service*.

A partir de um documento WSDL, um consumidor de serviço pode determinar os detalhes do *Web Service*, como as diferentes operações, tipos de dados, extremidades, protocolos e outros

para poder ter acesso ao mesmo. Entretanto, em vez de publicar o documento WSDL para cada possível cliente, ou mesmo no servidor *web* de sua empresa, um provedor terá mais sucesso e visibilidade se publicar as informações sobre seu *Web Service* em um registro central que esteja publicamente disponível e seja mundialmente conhecido como repositório de serviços, para os consumidores interessados. Pois, além de publicar a descrição de seu serviço, o provedor poderá também publicar detalhes relacionados ao negócio da empresa, como dados completos, áreas de atuações, produtos, entre outras, além dos diferentes *Web Services* por ele oferecido.

Já os consumidores podem acessar o registro central para encontrar os diferentes serviços disponibilizados pelos diferentes provedores existentes, podendo testar e avaliar os serviços que a eles interessam, antes de integrá-los às suas aplicações.

O formato para publicação e localização de informações comerciais é a especificação *Universal Discovery, Description, and Integration* – UDDI [45]. Sua especificação contém as seguintes informações [4]:

- **Negócio**: contém as informações comerciais da entidade: como o nome da empresa, o contato comercial, dentre outras. Permite também que a entidade tenha um código de classificação conforme uma determinada categoria de negócios, como por exemplo, instituições bancárias.

- **Serviço**: Uma estrutura de serviço captura os diferentes *Web Services* fornecidos por uma entidade. Cada serviço contém uma ou mais estruturas de especificação técnica. Uma entidade pode ter, por exemplo, dois *Web Services*: o primeiro representando o *status* da conta bancária e o segundo representando o *saldo* da conta. A especificação UDDI permite ainda que o serviço ofertado tenha um código de classificação conforme uma determinada categoria, como a de verificação de crédito, por exemplo.

- **Especificação técnica**: Essa especificação contém detalhes técnicos sobre um determinado *Web Service*. Logo, através dessa especificação, um consumidor de serviços pode fazer referência, e determinar os detalhes específicos sobre a chamada para o *Web Service* que está pesquisando. Um dos formatos para escrever essa descrição é a linguagem WSDL, como visto anteriormente.

- **Registros UDDI**

Os *web sites* de registros centrais públicos mais conhecidos, segundo Hendricks [4], são:

- Registro IBM – localizado em http://www.ibm.com/services/uddi. A página apresenta uma área de teste e uma área de produção. A área de teste está aberta para todos como um ambiente para analisar e testar o registro, assim como, verificar o funcionamento do serviço que está disposto a publicar. Assim, caso necessário, um provedor de serviços pode corrigir falhas ou melhorar seu serviço, antes mesmo de querer adicionar publicamente quaisquer informações do serviço proposto. Traz ainda, novidades em eventos, notícias e documentações de auxílio ao usuário.

- Registro Microsoft – localizado em http://uddi.microsoft.com. Também fornece um ambiente de produção e de teste e assemelha-se bastante ao registro da IBM.

Possui ainda busca dos serviços por ele ofertados, ajuda online e mostra o suporte de seus produtos à UDDI.

Caso seja de interesse a procura por outros registros, o *web site* UDDI.org (http://www.uddi.org/) é o principal centro de repositórios de entidades de registro do mundo.

2.3.4 Tecnologias no Projeto

Nas Seções anteriores vimos, as tecnologias mais usadas no contexto de sistemas *Web Services*. Logo a pilha básica de serviços, composta pelas tecnologias abordadas no desenvolvimento deste trabalho é indicada na Figura 15.

UDDI
WSDL
SOAP
HTTP

Figura 15. Pilha básica das tecnologias de *Web Services*

Uma vez conhecidas e definidas essas tecnologias, a próxima Seção nos mostra as características de aplicações para dispositivos móveis, sendo necessário seu estudo, por ser parte integrante do projeto, sendo o lado cliente da aplicação que se comunicará com o *Web Services* disponível no servidor *web*.

2.4 Plataforma J2ME

A plataforma J2ME - Java 2, Micro Edition [17] - é uma edição Java, criada pela Sun Microsystems destinada ao desenvolvimento de aplicações para dispositivos de baixa capacidade de processamento e pouca memória disponível. Como exemplos de dispositivos que se enquadram nessas características, podemos citar os telefones celulares, PDAs (*Personal Digital Assistants*), sistemas automotivos, etc [6].

Essa plataforma proporciona ao desenvolvedor criar aplicativos para diversos dispositivos, tal que funcionem da mesma maneira devido ao fato de não serem desenvolvidos tratando de aspectos específicos de um determinado aparelho, como veremos em mais detalhes adiante. Lembrando que, para buscar o melhor aproveitamento de processamento e memória, esses detalhes específicos, muitas vezes como sistema operacional do aparelho ou informações técnicas do fabricante, sejam importantes na criação de certos aplicativos voltados para um determinado grupo de dispositivos.

Nos primeiros dispositivos, todas as funcionalidades oferecidas eram implementadas pelo fabricante. Logo, para uma nova funcionalidade ser inserida no dispositivo, essa deveria ser implementada na linguagem nativa do proprietário, utilizando as bibliotecas existentes do próprio aparelho. Dessa forma, a possibilidade de aumentar o conjunto de aplicativos de um determinado dispositivo era de difícil acesso. Através do J2ME, tornou-se possível desenvolver, implantar, e até mesmo, atualizar novas ferramentas conforme as necessidades particulares de cada usuário, respeitando as tecnologias do fabricante, mas não mais, restrita a elas. “Além disso, com o grande número de celulares que já possuem máquinas virtuais Java e com o compromisso dos fabricantes em incluí-la em mais modelos, a flexibilidade dessa plataforma tende a se firmar como um padrão na indústria”, afirma Lima [12].

É importante mencionar que quaisquer aspectos técnicos quanto a tecnologia adotada pela operadora de telefonia celular ou quanto a forma de transmissão por ela estabelecido, não afetam diretamente os sistemas desenvolvidos para os aparelhos celulares que ela atende. Além disso, não é necessário nenhum tipo de licença da operadora, ou mesmo do fabricante do aparelho, para que o dispositivo possa executar outros aplicativos.

Existe uma diversidade muito grande de dispositivos no mercado. Alguns aparelhos têm pontos similares entre si, e podem fazer parte da mesma família ou série, mas mesmo assim, ainda se diferenciam em suas formas, funcionalidades e características. Existem também dispositivos que se diferenciam na capacidade de processamento, memória e interação com o mundo exterior (interface com o usuário, conectividade, formas de trocas de informações, etc).

Para contornar as diversificações dos dispositivos mencionadas acima, dois conceitos fundamentais foram definidos pela plataforma J2ME que são: Configurações e Perfis [6][9]. Essa divisão permite ao desenvolvedor um maior conhecimento prévio dos detalhes técnicos sobre as diferentes famílias de dispositivos e as funções e bibliotecas (APIs) disponíveis em cada uma delas.

2.4.1 Configurações

A configuração define um conjunto mínimo de tecnologias suportadas por um grupo de dispositivos com características similares em termos de memória e poder de processamento. Sendo assim, uma configuração especifica as características suportadas pela máquina virtual Java encontrada no dispositivo e as bibliotecas e APIs que estarão implementados num determinado grupo de dispositivos. [6][12].

Cada configuração é classificada de acordo com as capacidades de cada dispositivo. Essa classificação se baseia em características como memória disponível, display, conectividade de rede e poder de processamento do dispositivo.

Atualmente, existem duas configurações definidas e validadas pela Sun:

- Configuração para dispositivos conectados (CDC);
- Configuração para dispositivos com limites de conexão (CLDC).

CDC

A Configuração para dispositivos conectados (CDC) é projetada para dispositivos geralmente sem mobilidade com:

- pelo menos dois megabytes de memória disponível;
- no mínimo com 512 quilobytes de memória para execução do Java;
- no mínimo com 256 quilobytes para alocação de memória;
- conectividade com redes, possibilitando persistência e grande largura de banda.

Baseada na máquina virtual Java convencional (*Java Virtual Machine* - JVM), a *Compact Virtual Machine* (CVM) define um ambiente poderoso em recursos, similares aos encontrados na máquina virtual para PCs. Apresenta, porém, bem menos recursos de memória, e menos bibliotecas auxiliares. Alguns exemplos desses dispositivos são: televisão com acesso a Internet, sistema de navegação de automóveis, entre outros.

CLDC

A Configuração para dispositivos com limites de conexão (CLDC) é projetada para aparelhos com processadores de baixa velocidade, pequena capacidade de armazenamento e conexão de rede intermitente. A CLDC é projetada para dispositivos com:

- no mínimo com 128 quilobytes de memória para execução do Java;
- no mínimo com 32 quilobytes para alocação de memória;
- conectividade a algum tipo de rede (em geral sem-fio), com baixa largura de banda e acessos intermitente;
- normalmente, com interface e tela de visualização restritas para o usuário.

Usa uma máquina virtual reduzida em relação a máquina virtual CVM da configuração CDC, conhecida por KVM (*Kilo Virtual Machine*). Podemos citar como exemplos os dispositivos: telefones celulares, pagers e PDAs.

2.4.2 Perfis

Os perfis ou *profiles,* em inglês, procuram prover os recursos opcionais (modelo do ciclo de vida da aplicação, a interface com o usuário e acesso a propriedades específicas do dispositivo em questão) que não são dispostos pelas configurações. Ou seja, um perfil funciona como o complemento da configuração, em que esta define o mínimo de funcionalidades e o perfil atende detalhes específicos do dispositivo oferecendo bibliotecas e serviços para desenvolver aplicações para tais aparelhos, ou seja, o perfil é a camada mais visível para usuários e desenvolvedores de aplicações.

Para cada configuração particular (CLDC ou CDC) temos perfis específicos implementados, podendo ainda um dispositivo sustentar múltiplos perfis. Os tipos de perfis mais conhecidos em J2ME são: MIDP (*Mobile Information Device Profile*), *Foundation Profile*, *Personal* e *Personal Basis Profile*, RMI Profile, PDA Profile e *Game Profile* [6][8][9].

As aplicações são escritas para um perfil específico sendo, desse modo, portáveis para qualquer dispositivo que suporte aquele perfil. No contexto deste trabalho, abordaremos apenas o perfil MIDP, que é o perfil utilizado em aparelhos celulares com configuração CLDC. Este perfil provê interface mínima com o usuário, entrada e persistência de dados, manipulação de eventos, modelo de armazenamento orientado a registro, rede, mensagens, etc.

2.4.3 KVM

Como já mencionamos, a *Kilo Virtual Machine* (KVM) é uma nova implementação da JVM para dispositivos portáteis com recursos limitados. A KVM aceita o mesmo conjunto de *bytecodes* e o mesmo formato para o arquivo '.class' que a máquina virtual clássica.

A KVM tem aproximadamente 60 *quilobytes*, o que explica a designação "K" na definição da máquina virtual para dispositivos com a configuração CLDC.

2.4.4 Ofuscador

Os arquivos de classe Java gerados a partir do código fonte contém ainda muitas informações sobre o código fonte original. Um programa Java compilado pode ser facilmente decompilado através de engenharia reversa, tornando o código fonte original acessível a qualquer pessoa que se utilize desse mecanismo.

O ofuscador é uma ferramenta cujo objetivo é dificultar, ou mesmo evitar, o acesso ao código fonte original a partir do programa compilado. O ofuscador reescreve o código compilado - *bytecode*, tornando-o impossível de ser decompilado - difícil de ser encontrado através de uma possível engenharia reversa [36]. Um bom ofuscador gera um código final tão eficiente quanto o código compilado original.

Além de reescrever o código, o ofuscador também remove do código todas as classes e métodos não utilizados pelo programa; renomeia classes, campos e métodos usando nomes curtos e sem sentido. Com isso, o ofuscador consegue ainda reduzir bastante o tamanho em *bytes* do programa final. Dessa maneira, os ofuscadores são muito utilizados para a produção de programas desenvolvidos para dispositivos móveis, em especial celulares, pois além de evitar que o código fonte original seja encontrado, há uma diminuição no tamanho final do aplicativo que será instalado no celular, e como a memória disponível é bastante limitada nesse tipo de dispositivo, cada *kilobyte* (kB) economizado faz uma grande diferença.

O RetroGuard, da RetroLogic [36], é um dos ofuscadores mais conhecidos. Desenvolvido em Java, é distribuído como *open source* sob licença GPL [26]. Em nosso projeto, enquanto o programa compilado, ficou com o tamanho de 66 kB, o compilado usando o RetroGuard permaneceu com apenas 52 kB. O que comprova a redução do tamanho final do aplicativo em cerca de 20% com as mesmas funcionalidades dos encontrados no original e sem perdas aparentes de desempenho, por esse motivo o código compilado usando o ofuscador foi utilizado no projeto. Detalhes aprofundados dessa redução e o uso de ofuscadores não serão abordados no decorrer do projeto, sendo mencionados na Seção de Conclusão como possíveis complementos desta monografia.

2.4.5 Arquitetura J2ME

Como vimos, para suportar o tipo de diversificação dos dispositivos, a arquitetura do J2ME é composta por Configurações e Perfis. Essa divisão permite ao desenvolvedor conhecer informações específicas sobre as diferentes famílias de dispositivos e as APIs disponíveis em cada uma delas.

Como todos os dispositivos referidos possuem um sistema operacional previamente instalado, a arquitetura do J2ME faz também referência ao sistema operacional do dispositivo como sendo a base para amparar a plataforma J2ME. Segundo a Sun, a arquitetura J2ME é modular e escalável. Essa modularidade e escalabilidade são definidas pela tecnologia J2ME em um modelo de 3 camadas: a camada de perfil, a camada de configuração e a camada representada pela máquina virtual Java. A Tabela 1 ilustra as camadas genéricas do J2ME.

Tabela 1. Camadas do J2ME

Camadas genéricas	Camadas para dispositivos MIDP
Perfis	MIDP
Configurações	CLDC
Máquina Virtual Java	KVM
Sistema Operacional	Sistema Operacional

Em telefones celulares, é encontrada principalmente a combinação CLDC 1.0 e MIDP 1.0; alguns lançamentos recentes atendem a especificação MIDP 2.0 e/ou CLDC 1.1, mas ainda não são dispositivos massificados [12]. Por esse motivo este trabalho segue a configuração CLDC, o perfil MIDP, evidentemente, a máquina virtual Java utilizada será a sua versão reduzida, a KVM, e o sistema operacional será o encontrado no dispositivo, não acarretando restrições aos aplicativos desenvolvidos. A Tabela 1 mostra as camadas do J2ME para dispositivos do tipo MIDP que serão utilizadas no projeto.

Apesar da grande variedade de aparelhos celulares dispostos hoje no mercado mundial, existindo aparelhos extremamente simples, com visor monocromático, telas pequenas e teclas básicas para entrada de dados; e existindo também celulares bem mais complexos, com maior poder de processamento, telas grandes e com suporte a uma grande de quantidade de cores, além de várias teclas de entrada de dados a mais; o sistema aqui proposto visa superar essas diferenças para garantir que o mesmo aplicativo funcione da mesma maneira mesmo que em aparelhos completamente diferentes. Isto deverá ser alcançado, pois não serão utilizados detalhes técnicos dos aparelhos utilizados, como bibliotecas ou funções específicas a um determinado grupo de dispositivos ou fabricante, exigindo apenas que os mesmos disponham da arquitetura encontrada na Tabela 1.

As aplicações Java baseadas na configuração CLDC e no perfil MIDP, e, portanto voltadas a dispositivos móveis, são chamadas de Midlets. Os Midlets são pacotes do tipo JAR (*Java Archive*) com arquivos de classes java.

2.5 Ferramentas utilizadas

Nessa Seção são apresentadas as ferramentas utilizadas na criação e execução do serviço proposto. São abordadas tanto as ferramentas presentes no desenvolvimento do lado cliente quanto do lado do servidor.

2.5.1 Servidor Apache Tomcat

Todo *Web Service* precisa ficar sempre ativo e esperando requisições, portanto, necessita estar executando em um servidor e deve estar conectado a uma rede (internet, intranet, etc).

O software Tomcat [20], desenvolvido pela Fundação Apache, permite a execução de aplicações para *web* funcionando como um servidor de aplicação que mostrou ser bastante confiável. Sua principal característica técnica é estar centrada na linguagem de programação Java, mais especificamente nas tecnologias de *Servlets* e de *Java Server Pages* (JSP). Essa abordagem rivaliza, por exemplo, com a usada pela Microsoft com o ASP (baseada na linguagem *Visual Basic*). A Fundação Apache, mais conhecida pelo seu servidor *web* de mesmo nome, permite, como no caso do servidor Apache, que o Tomcat seja usado livremente, seja para fins comerciais

ou não.

O Tomcat está escrito em Java. No entanto, não basta ter a versão *runtime* de Java instalada, pois ele necessita compilar (e não apenas executar) programas escritos em Java, e, por isso, necessita que a versão Java 2 Standard Edition (J2SE) esteja instalada no computador onde ele é executado. O Tomcat é o subprojeto mais conhecido do projeto Jakarta, da mesma Fundação, cujo objetivo é o desenvolvimento de soluções de código aberto baseadas na plataforma Java.

Como era de se esperar, o desenvolvimento de uma aplicação *web* a ser executada pelo Tomcat deve ser escrito na linguagem Java. Sua escolha no projeto deve principalmente ao fato de ser um servidor de fácil implantação, confiável e que executa aplicações Java. A versão do servidor Apache Tomcat utilizada no projeto é a 5.0.28 encontrada no site da Fundação Apache (http:// jakarta.apache.org/tomcat/).

Infelizmente, o servidor Tomcat não implementa as tecnologias e padrões usados para construir *Web Services*. Logo, o uso de uma biblioteca adicional foi preciso, sendo escolhido o pacote Apache Axis visto no Capítulo 2.

2.5.2 TCP Monitor

Como observamos anteriormente, o Axis oferece uma classe de utilitário conhecida como *tcpmon* para o monitoramento do fluxo de mensagens por meio do mecanismo do Axis, agindo como um roteador TCP. O *tcpmon* permite visualizar a movimentação das mensagens TCP entre o cliente e o servidor. Essa ferramenta permite examinar o protocolo de rede do SOAP e até mesmo verificar as várias implementações do SOAP para fins de compatibilidade.

Para iniciar este utilitário, iniciamos o servidor Tomcat, e numa janela de *prompt* de comando, executamos o arquivo de classe `java org.apache.axis.utils.tcpmon`. A Figura 16 mostra a janela inicial do TCPMonitor.

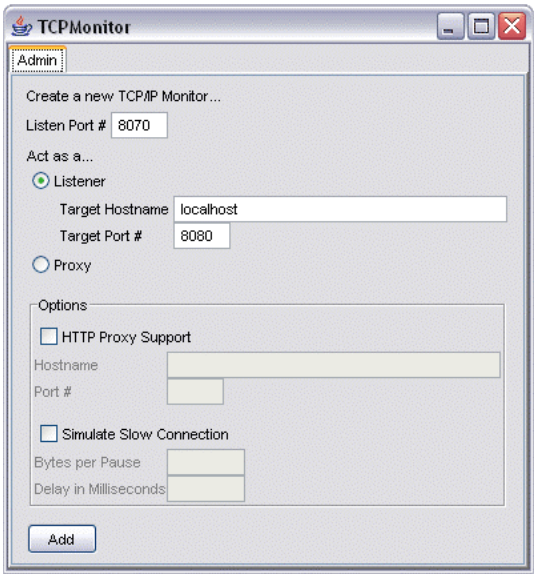


Figura 16. Janela Inicial do TCPMonitor

Por padrão, o botão de rádio *Listener* é selecionado. A outra opção, *Proxy*, é usada para especificar os detalhes de um servidor *proxy* HTTP. No nosso projeto, os valores para *Listen Port*, *Target Hostname* e *Target Port*, respectivamente, foram 8070, *localhost* e 8080. Ao clicar

no botão *Add*, uma nova janela será exibida mostrando o monitor aguardando solicitações na porta especificada. A Figura 17 mostra esta nova janela.

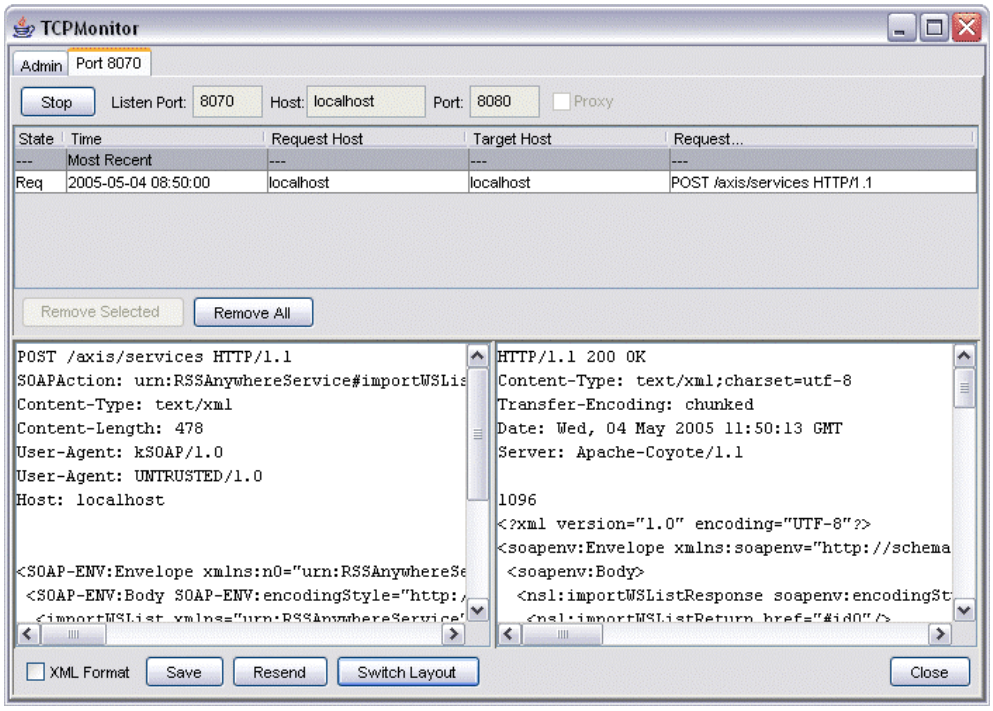


Figura 17. TCPMonitor monitorando as mensagens SOAP enviadas

Nesse ponto, o monitor está atuando como um intermediário entre o cliente e a porta 8080, esperando por requisições na porta 8070. Quando receber uma requisição, ele passará para o sintonizador na porta 8080.

A partir desse ponto, sempre que o cliente chamar um serviço disponível no servidor *web*, será feita uma conexão do SOAP para a porta local. A requisição do SOAP aparece no painel *Request*, e a resposta do servidor está no painel *Response*. O aplicativo *tcpmon* ainda mantém um *log* com todos os pares de requisição-resposta, e permite visualizar qualquer par bastando selecionar uma entrada no painel superior. É possível também remover as entradas selecionadas ou todas elas ou optar por salvar em um arquivo para futura visualização.

Na Seção 4.5 de Interface Gráfica, todas as mensagens SOAP referenciadas serão 'capturadas' através do TCP Monitor.

2.5.3 Java Wireless Toolkit

O Java Wireless Toolkit (WTK) [42], desenvolvido pela Sun Microsystems, é um ambiente de desenvolvimento que emula dispositivos móveis, no padrão MIDP que é o padrão suportado na maioria dos celulares. Além do emulador, o WTK compila e pré-verifica os códigos Java desenvolvidos e contém ainda vários exemplos de aplicações em J2ME e possui ótima documentação descrevendo o MIDP.

Nele é possível configurar, dentre outros aspectos, a velocidade da CPU do aparelho e o fluxo de rede, para que a simulação seja a mais próxima possível da realidade estudada, já que computadores possuem muito mais memória e CPU e nem sempre o desenvolvedor dispõe do aparelho celular para instalar e testar os sistemas criados por ele.

O Wireless Toolkit é gratuito e simplifica o desenvolvimento, porém, não possui um IDE (*Integrated Development Environment*) e, portanto, a edição do código fonte deve ser feita em ferramentas separadas, como o Sun ONE Studio, o JBuilder, um editor de textos simples, ou qualquer outro editor. No projeto utilizamos a versão 2.1 do WTK e editamos o código no bloco de notas do Windows.

Capítulo 3

Integração J2ME e Web Services

Como vimos, para trabalhar com *Web Services*, algumas tecnologias são definidas e aceitas como padrão para criar provedores e consumidores de *Web Services*. Para tanto, tais tecnologias são requeridas tanto no servidor que disponibiliza o serviço, quanto para o cliente que irá acessá-lo.

A comunicação, por exemplo, entre o cliente e o servidor é feita através do protocolo SOAP, baseado em XML. No nosso caso, veremos que o servidor de aplicação escolhido não disponibiliza bibliotecas (APIs) que trabalham com o SOAP. Por isso, faz-se necessário o uso de um pacote adicional específico para trabalhar com *Web Services*, chamado Apache Axis [5][19]. Da mesma maneira, o MIDP não possui classes pré-definidas para tratar esse protocolo. Logo, é necessário utilizar uma API auxiliar para atender essa tecnologia. Essa API é conhecida por kSOAP [14][30].

Esses pacotes funcionam como limites entre tipos Java e tipos SOAP transmitidos, realizando mapeamentos de tipos Java para SOAP e vice-versa tanto do lado cliente quanto do lado do servidor. Esse esquema pode ser visto na Figura 18.

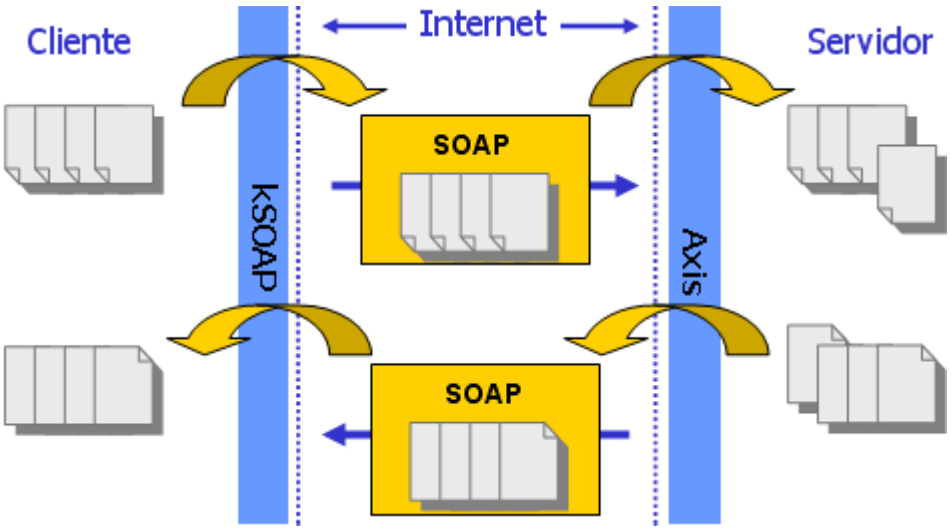


Figura 18. Esquema de mapeamentos entre tipos Java e SOAP

Nas próximas Seções estudaremos em detalhes esses pacotes.

3.1 Pacote Apache Axis

O *Apache eXtensible Interaction System* (Axis), desenvolvido pela Fundação Apache [19], é um *framework* que permite a construção de clientes e servidores de *Web Services* utilizando o protocolo SOAP. Como vimos, um *Web Service*, basicamente, é um serviço instalado num servidor que aceita arquivos XML que representam requisições (*requests*) e retornam respostas (*responses*). O SOAP é o padrão para transferência de dados entre sistemas e é o que rege a formação dos arquivos XML de requisição e resposta. Portanto, podemos dizer que Apache Axis é um *framework* para se construir processadores SOAP.

O Axis implementa a API Java *XML-based Remote Procedure Call* (API JAX-RPC), definida pela comunidade Java *Process Program* (JCP) e que permite, como o próprio nome já diz, a chamada de procedimentos remotos por meio de arquivos XML. A versão atual do pacote Axis utiliza a especificação SOAP 1.1 [40].

Testes são necessários para verificar a correta instalação do Axis, e são realizados automaticamente acessando a página *http://localhost:8080/axis/happyaxis.jsp* pelo navegador *web*. Essa página, após processada, indica se as bibliotecas do Axis estão sendo encontradas, assim como qualquer biblioteca adicional que se queira usar, como uma classe específica para a possibilidade de enviar emails, por exemplo. Caso alguma biblioteca não seja encontrada, indica onde e como proceder para corrigir o problema. Essa página também informa a configuração do servidor *web* que está sendo executado. As principais características do pacote Axis são:

- Diversos sistemas de exemplos;
- Configuração flexível e um fácil registro (*deploy*) de serviços;
- Suporte para registros rápidos de serviços como o *Simpls Classes Java* arquivos Java com extensão .JWS ou ainda, um registro mais poderoso utilizando um arquivo de descritor de organização que rege o serviço;
- Suporte para transmissão via SOAP de todos os tipos de dados primitivos de Java e a possibilidade de transmitir tipos de dados definidos pelo usuário chamados de *type mapping;*
- Geração automática do WSDL para registros de serviços;
- Disponibilizar uma ferramenta para monitorar pacotes TCP/IP (*tcpmon*);
- Disponibilizar uma ferramenta para criar as classes Java a partir de um WSDL (WSDL2Java) e outra para construir o WSDL a partir das classes Java (Java2WSDL).

Para que um *Web Service* seja accessível no servidor, três passos são necessários:

(i) implementar a classe que irá representar o serviço;
(ii) escrever um descritor de organização do *Web Service*;
(iii) registrar o serviço desejado no Axis para responder às requisições.

Todos esses passos serão abordados no Capítulo seguinte, que aborda dentre outros aspectos, as etapas necessárias no processo de registro de um *Web Services*.

3.1.1 Arquitetura

Analisando a arquitetura do Axis para melhor compreender o seu funcionamento, vejamos como

uma requisição chega ao *Web Service* desejado e como ocorre o retorno das informações ao requisitante. Para isso, examinaremos o fluxo de comunicações entre seus componentes básicos, mostrados na Figura 19.

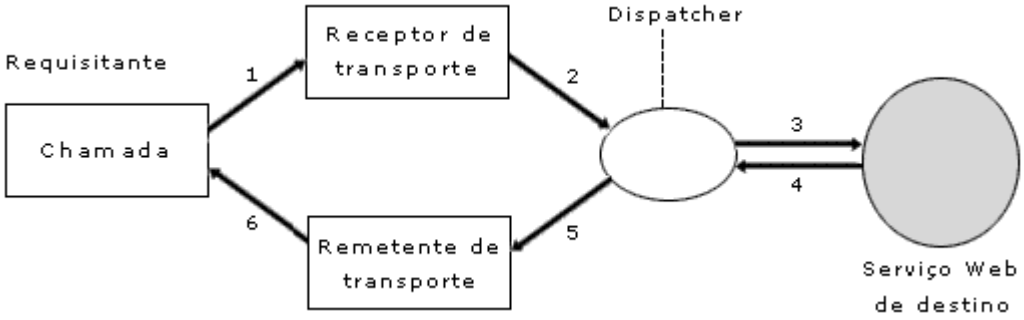


Figura 19. Fluxo de comunicação entre componentes do Axis

O processamento de uma mensagem SOAP no Axis é feito passando um contexto de mensagem entre cada um de seus componentes. O contexto de mensagem é um conjunto de informações que contém uma mensagem de requisição, uma mensagem de resposta e um conjunto de propriedades (como os atributos do cabeçalho do SOAP, por exemplo) [4].

As etapas do processamento de uma requisição para um *Web Service*, segundo Hendricks [4], são:

1. O requisitante constrói uma requisição do SOAP especificando o local de destino, os detalhes do serviço (como nome do serviço, o nome do método chamado e os parâmetros de entrada), estilo de codificação, etc; usando para isso um objeto *Call*. Logo após, a mensagem relativa à requisição é enviada ao nó do processamento da mensagem do Axis, no receptor de transporte. A mensagem do requisitante, nesse ponto, está em formato específico para o protocolo de rede escolhido;
2. O receptor de transporte converte a mensagem de entrada em um objeto *Message* (*org.apache.axis.Message*) e o inclui em um objeto *MessageContext*. O receptor também carrega várias propriedades como atributos do SOAP e define a propriedade *transportName* no *MessageContext*. Esse *transportName* pode ser *http*, *smtp* ou *ftp* baseado no protocolo usado no mecanismo de transporte. Em seguida, a mensagem é enviada ao *dispatcher*;
3. O *dispatcher*, como o próprio nome já sugere, é responsável pelo encaminhamento da requisição ao *Web Service* de destino;
4. O serviço, por sua vez, executa o método e retorna a resposta ao *dispatcher*;
5. O dispatcher, então, encaminha a resposta ao remetente do transporte;
6. Por fim, o remetente do transporte, junto com o objeto do contexto do transporte, envia a mensagem do SOAP de volta através do protocolo de rede ao requisitante. O objeto do contexto do transporte encapsula os detalhes do receptor do transporte, o contexto relacionado ao iniciador da requisição e o destino da resposta/falha da mensagem, os detalhes da sessão, etc.

Como podemos perceber, a mensagem SOAP é criada pelo próprio mecanismo do Axis, deixando o *Web Service* isento dessa tarefa, dando a idéia que o serviço está se comunicando diretamente com o cliente através de chamadas a métodos, passando argumentos como parâmetros e aguardando a resposta dos mesmos.

3.2 Pacote kSOAP

Como vimos, a comunicação é realizada através do protocolo SOAP e, assim como o servidor que não possui bibliotecas específicas para o SOAP, o cliente, no caso MIDP, também não possui classes pré-definidas para tratar esse protocolo e, por isso, é necessário utilizar uma classe (API) para atender essa tecnologia. Essa API é conhecida por kSOAP da *ObjectWeb*, sendo disponibilizada sob licença pública pelo web *site* da empresa.

O projeto kSOAP é baseado no projeto kXML [31], uma vez que o protocolo SOAP é baseado em XML. A classe kXML oferece um conjunto de funcionalidades contendo *parser* e *writer*, para processamento e escrita respectivamente, que são utilizados pelo pacote kSOAP. Já esse último possui as classes responsáveis para o empacotamento/descompactamento de chamadas SOAP através do protocolo HTTP, através das classes `SoapObject` e `HttpTransport` respectivamente.

Esses projetos foram desenvolvidos especificamente para trabalhar com dispositivos na configuração CLDC e, conseqüentemente, executados sob a máquina virtual KVM. Por isso, eles possuem um conjunto menor de funcionalidades e objetos que juntos não passam de 42kB de tamanho em disco.

Como estamos tratando de plataformas distintas, o fato de trocar mensagens SOAP não implica necessariamente na compatibilidade das informações (dados) transmitidas (enviadas e recebidas). Quando uma mensagem SOAP do servidor chega ao cliente, um parser interno do pacote do kSOAP faz o mapeamento dos tipos encontrados na mensagem para os tipos de dados referentes na linguagem Java. Vejamos em detalhes como funciona esse *parser*.

3.2.1 Parser

Numa mensagem SOAP, o atributo `xsi:type` dentro de um elemento qualquer, especifica o tipo de dado daquele elemento XML. Por exemplo, `<myValue xsi:type="xsd:int">123</myValue>` especifica um valor inteiro 123 do elemento `myValue`, e `<myValue xsi:type="xsd:string">123 </myValue>` especifica o *string* "123".

O kSOAP faz um mapeamento automático de cinco tipos SOAP para tipos Java conforme a Tabela 2.

Tabela 2. Mapeamento de Tipos Padrão do kSOAP

Tipo SOAP	Tipo Java
`xsd:int`	`java.lang.Integer`
`xsd:long`	`java.lang.Long`
`xsd:string`	`java.lang.String`
`xsd:boolean`	`java.lang.Boolean`
`xsd:array`	`java.lang.Array/Vector`

Quando o *parser* do kSOAP encontra um elemento SOAP, ele lê o elemento XML para uma memória reservada de dados de objeto Java de acordo com as seguintes regras:

1. Se o elemento SOAP em questão é um dos tipos primitivos padrões encontrados na Tabela 2, converta para o tipo Java associado;
2. Se o elemento SOAP em questão não tem um nó filho, e é um tipo padrão da Tabela 2, converta para um objeto do tipo `SoapPrimitive`. Podemos acessar o valor *string* do elemento através do método estático `toString`, e ainda, podemos acessar o tipo original do elemento SOAP recebido através dos métodos estáticos `getNamespace` e `getName`;
3. Se o elemento SOAP em questão tem um nó filho, e é um tipo complexo, converta para um objeto do tipo `KvmSerializable`. `KvmSerializable` é uma interface que o pacote kSOAP provê como implementação do objeto `SoapObject`. Similar ao objeto `SoapPrimitive`, podemos acessar o tipo original do elemento SOAP recebido através dos métodos estáticos `getNamespace` e `getName`;
4. Se o elemento SOAP em questão for um nó filho do tipo complexo, converta para dentro das *propriedades* do objeto `SoapObject` pai, de acordo com as três regras descritas acima. Cada *propriedade* tem um objeto `PropertyInfo` associado, que contém informações como o nome do elemento SOAP e os tipos de objetos Java encontrados.

Como um `SoapObject` pode ter outros `SoapObjects` como propriedades, nós podemos usá-lo para representar estruturas hierárquicas complexas. A Figura 20 mostra um exemplo genérico de tal estrutura.

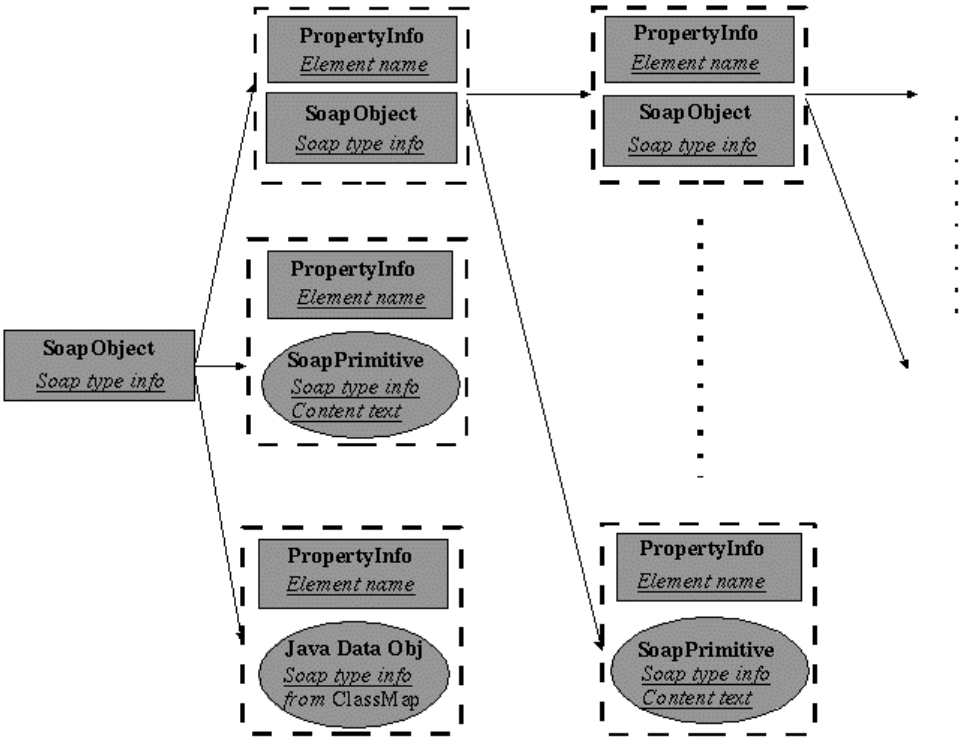


Figura 20. Representação hierárquica usando SoapObject

Cada caixa com uma linha tracejada representa um elemento SOAP da mensagem recebida depois do *parser* kSOAP. Dentro de cada caixa, o nome original do elemento SOAP é armazenado no objeto de `PropertyInfo` e o tipo do atributo name/namespace SOAP é

armazenado no objeto *propriedade*, logo abaixo de `PropertyInfo`. As setas representam relacionamentos da *propriedade*. Podemos encontrar a informação original do tipo SOAP dos objetos dos tipos de dados de Java procurando os mapeamentos no objeto de `ClassMap`, que contém a informação da Tabela 2 (veremos o objeto `ClassMap` mais tarde).

Como podemos ver, a raiz `SoapObject` não tem um objeto `PropertyInfo` emparelhando-se para fazer uma "caixa cheia" (`PropertyInfo + SoapObject`). Conseqüentemente, nós perdemos o nome do elemento da raiz após o *parser* kSOAP. Entretanto, isso não é um problema, pois o nome do elemento normalmente serve somente como índices para alcançar objetos do tipo de dados de Java, e não necessitamos de um índice para alcançar o tipo de dados do elemento raiz.

3.2.2 Mapeamento de Tipos

Para automatizar ainda mais o processo de mapeamento de tipos do *parser*, podemos preparar o *parser* do kSOAP para executar duas tarefas:

- Todo o relacionamento de mapeamentos entre tipos do SOAP aos tipos de Java deve ser conhecido pelo *parser*. Para isso, adicionamos cada par de tipos mapeados ao objeto `ClassMap` do *parser* em questão.
- Como todos os tipos do SOAP são apresentados na forma de texto puro (*strings*), o *parser* também deve saber como converter esse texto para um objeto desejado em Java. Assim, ele converte a *string* encontrada através de um objeto `Marshal`, o qual corresponde ao mapeamento do par dos tipos SOAP e Java registrados no objeto `ClassMap`.

Qualquer objeto Java que se queira criar para ser transportado via SOAP, tem que implementar a interface `kvmSerializable` do pacote kSOAP, o qual possui quatro métodos (mostrados na Tabela 3). Esses métodos são usados pelo kSOAP para mapear cada elemento do objeto num tipo conhecido pelo kSOAP no cliente. Se algum elemento do objeto em questão for um outro tipo complexo, esse tipo também tem que implementar a interface `kvmSerializable`, e assim por diante.

Tabela 3. Sumário dos métodos da interface kvmSerializable

`java.lang.Object`	**`getProperty`**`(int index)` `Retorna a propriedade de um índice especifico`
`int`	**`getPropertyCount`**`()` `Retorna o numero de propriedades`
`void`	**`getPropertyInfo(`**`int index, PropertyInfo info)` `Preenche a Propriedade desejada com um objeto designado de PropertyInfo`
`void`	**`setProperty`**`(int index, java.lang.Object value)` `Ajusta a propriedade com o índice do valor especificado`

Veremos mais adiante, que a comunicação entre o cliente e o servidor se dará por trocas de mensagens SOAP contendo objetos `FeedRSS`, os quais não são mapeados automaticamente pelos pacotes Axis (no servidor) e kSOAP (no cliente), uma vez que este objeto está fora dos tipos pré-definidos da Tabela 2. Logo, teremos que adaptar o objeto `FeedRSS`, para funcionar integrado com tais pacotes, alterando os códigos criados para esta finalidade. Na Seção de

Implementação, no próximo Capítulo, veremos os trechos de códigos que refletem este mapeamento no servidor e no cliente.

Toda conversão das mensagens SOAP transmitidas entre o cliente e o servidor, e vice-versa, para fluxos de bytes que serão enviados e retornados no nível de rede, não serão tratadas neste projeto, assim como a serialização e desserialização dos objetos `FeedRSS` transportados, uma vez que o uso de APIs já oculta do desenvolvedor conhecer detalhes desse processo.

Capítulo 4

Estudo de Caso

Neste Capítulo, mostraremos todo o desenvolvimento do aplicativo gerado, tanto do lado cliente quanto do lado servidor. Serão abordadas questões como arquitetura, metodologias, funcionalidades assim como códigos, dificuldades e testes.

4.1 Arquitetura

A fim de disponibilizar um serviço para dispositivos móveis, em especial celulares, que centralizasse de forma simples e objetiva a busca por informações contidas em arquivos RSS disponíveis em diversos *web sites* na Internet, foi criado um *Web Service* que busca e processa essas informações e as envia para o dispositivo solicitante. Dessa forma, continuamos proporcionando mobilidade ao usuário final da aplicação, uma vez que podemos acessar este *Web Service* do aparelho celular em qualquer lugar que a rede de telefonia esteja presente.

A arquitetura da aplicação consiste basicamente em dois módulos (sistemas): a aplicação MIDlet do lado cliente - a qual chamaremos de **RSSAnywhere** - e o *Web Services* do lado do servidor de aplicações, que definiremos de **RSSAnywhereService**. Ambos trocam mensagens SOAP através da Internet (HTTP). A Figura 21 mostra o esquema dessa arquitetura.

É importante lembrar que o *Web Services* só envia mensagens, se esse for solicitado por um agente externo.

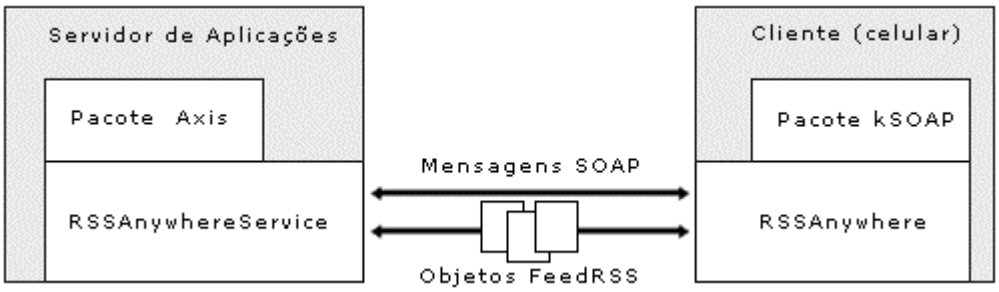


Figura 21. Arquitetura do Projeto

A Figura 21 ainda apresenta os pacotes adicionais em cada sistema, que permitirão as trocas de mensagens SOAP. O pacote Axis, do lado do servidor, e o pacote kSOAP do lado do dispositivo móvel.

O cliente foi desenvolvido em duas camadas (*View-Model*), semelhante ao padrão MVC. A camada *View* é responsável pela visualização das informações ao usuário do aplicativo, enquanto a camada *Model* é responsável, num nível mais interno, por acessar o *Web Service* na camada de rede. O modelo MVC não foi usado completamente, pois a camada de Controle (*Control*) iria trazer uma complexidade maior durante a implementação do cliente J2ME, que seria desnecessária pelo fato do aplicativo não ser complexo. A Figura 22 mostra o esquema utilizado no aplicativo cliente.

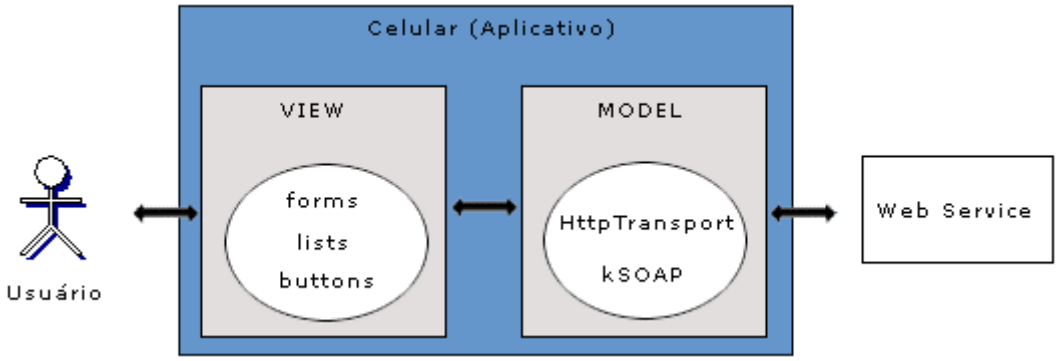


Figura 22. Esquema de camadas do aplicativo cliente

Uma chamada para um método disponível no *Web Service* pode ser analisada partindo do método `importWSList`, por exemplo, encontrado no cliente, até a resposta do método com o mesmo nome encontrado no servidor, enumerado a seguir:

1. O usuário faz uma requisição para o aplicativo, solicitando algum serviço (método) do *Web Services* do servidor através do menu principal do aplicativo (camada *View*). No exemplo, o usuário vai pedir pela lista de Feeds RSS disponíveis no Servidor. O nome desse serviço registrado no *Web Services* é `importWSList`;
2. A requisição do cliente segue para uma camada interna do aplicativo, responsável pela chamada ao método no *Web Service* (camada *Model*). Depois de feita a chamada para o WS, a camada *Model* espera pela resposta para retorná-la para a camada *View* que exibirá as informações recebidas. Lembrando que o empacotamento da mensagem em SOAP e seu transporte via HTTP é transparente para desenvolvedor do aplicativo assim como para o usuário do mesmo.
3. Quando o servidor *Web* recebe a requisição e identifica qual método, dos disponíveis no serviço, o cliente está requisitando, esse faz as conversões da mensagem SOAP dos dados recebidos e os prepara de forma adequada para repassar a requisição para o método implementado internamente no *Web Services*, no caso, a função `importWSList`;
4. Uma vez a lista de Feeds RSS processada e criada pela função `importWSList`, o *Web Service* encarrega-se de empacotar esses dados no protocolo SOAP e os envia pela rede ao cliente que fez a chamada ao método.

4.2 Metodologia

A Metodologia de desenvolvimento do estudo de caso está fundamentada em *Extreme Programming* (XP) [1], criada inicialmente por Kent Beck, Ron Jeffries e Ward Cunningham. É baseada nos valores de:

Simplicidade: O projeto do software é simplificado continuamente e seus processos também podem ser adaptados, basta que algum componente do grupo de trabalho dê uma sugestão para torná-los mais simples. É isso que sustenta a metodologia no seu extremo.

Comunicação: Tornar a interação dos grupos de desenvolvimento mais íntima, preferindo: *messengers* a emails; telefonemas a *messengers*; conversar pessoalmente a telefonemas; trabalhar na mesma sala a ter salas isoladas; trabalhar em conjunto a fazer reuniões isoladas.

Coragem: É preciso coragem para: apontar problemas no projeto ou no seu desenvolvimento; pedir ajuda quando necessário; simplificar códigos que já estejam funcionando; dizer ao cliente que não será possível implementar um requisito no prazo estimado; realizar mudanças e alterações no processo. Ou seja, fazer a mudança certa mesmo que não seja a mais rápida ou não seja mais popular naquele momento.

Feedback: Ter em mente: "todo problema é evidenciado o mais cedo possível para que possa ser corrigido o mais cedo possível e toda oportunidade é descoberta o mais cedo possível para que possa ser aproveitada o mais cedo possível".

É importante lembrar, que esses valores foram aplicados a partir de um único desenvolvedor, o qual também fez o papel do cliente avaliando e sugerindo mudanças durante o decorrer do desenvolvimento.

Além desses valores, existe um conjunto de práticas que, "levadas ao extremo", confere à metodologia XP uma característica bastante voltada para a programação propriamente dita. As práticas não são novidades, e podemos destacar: *Small Releases* - O software é entregue em pequenas versões para que o cliente possa obter o seu ganho o mais breve possível e para minimizar. A combinação dessas práticas e o fato de serem "levadas ao extremo" é que provocam todo o resultado esperado.

O **padrão de codificação** da linguagem de programação Java escolhida, é a recomendada pela Sun Microsystems que define desde a forma dos nomes das classes Java utilizadas até a forma de comentários recomendados dentro do código. Mais detalhes sobre esse padrão podem ser encontrados em [23].

4.3 Requisitos Funcionais

Das 8 funcionalidades encontradas no cliente (aplicativo RSSAnywhere), detalhamos apenas 3 delas, que são as responsáveis por comunicar com o servidor *Web Services*, sendo assim, as funcionalidades principais do trabalho. Essas funcionalidades são mostradas na Figura 23.

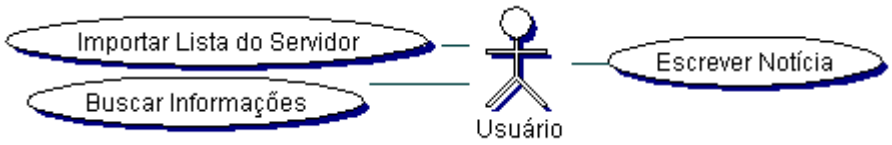


Figura 23. Principais requisitos funcionais do aplicativo RSSAnywhere

Nessa Seção mostramos através de diagramas de seqüência as funcionalidades mostradas na Figura 23. Além das mensagens enviadas e recebidas da comunicação entre o cliente e o servidor, algumas funções internas podem também ser exibidas nos diagramas.

O diagrama de seqüência mostra a comunicação entre atores, no qual o cliente fará o ator do dispositivo móvel (celular) e o servidor do ator do Apache Tomcat disponibilizando o *Web Service*.

4.3.1 Importar Lista do Servidor

Essa funcionalidade tem como objetivo solicitar ao servidor a lista mais recente de fornecedores `FeedRSS` disponíveis que está registrada no *Web Service*, pelo método `importWSList`.

O tipo de dado `FeedRSS` se comporta como um arquivo RSS. Nele encontramos o nome do *web site* que está disponibilizando o arquivo RSS, a descrição desse *web site*, assim como, o *link* para acessar diretamente o mesmo. Além desses elementos básicos, sua estrutura contém dois vetores: um contém os títulos das matérias encontradas no arquivo RSS e o segundo contém o resumo dessas matérias. No caso do método `importWSList`, esses vetores estarão vazios em cada `FeedRSS` retornado ao cliente, uma vez que o mesmo pode não estar interessado nas informações de determinado fornecedor.

Uma vez a requisição chegando ao servidor, esse cria um vetor de resposta contendo os elementos `FeedRSS` que mantém em sua estrutura interna e retorna esse vetor ao cliente. O cliente por sua vez, ao receber o vetor do servidor, processa as informações colhidas e as lista na tela do cliente. Tal seqüência pode ser acompanhada pela Figura 24.

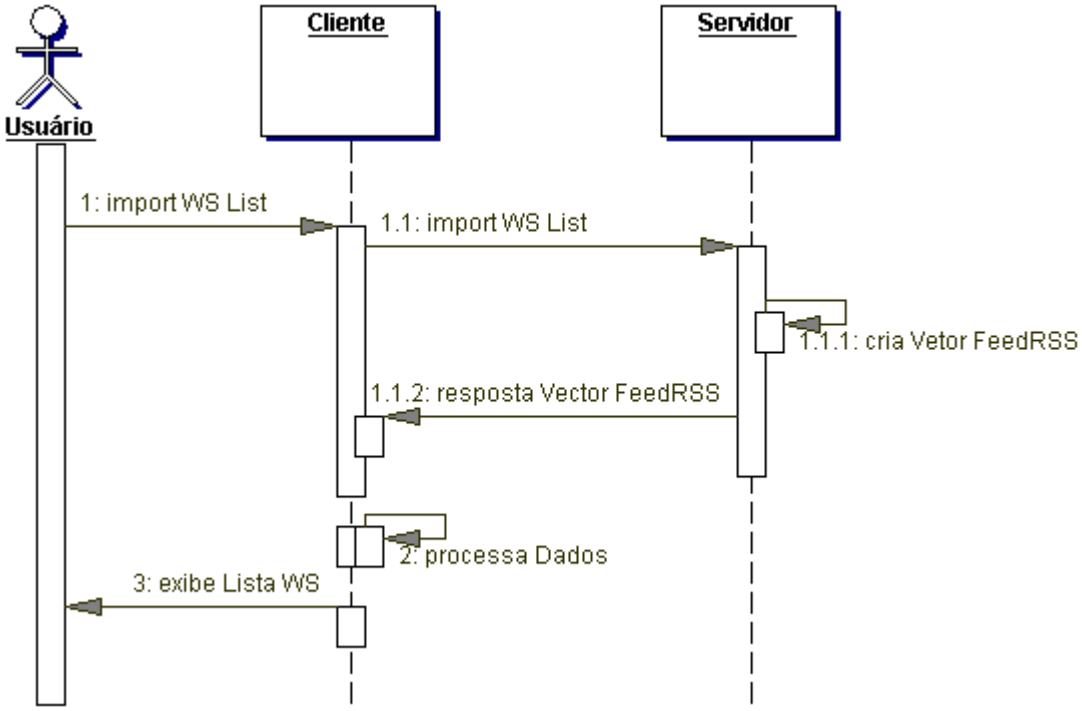


Figura 24. Diagrama de seqüência do método importWSList

4.3.2 Buscar Informações

Essa funcionalidade tem como objetivo solicitar ao servidor as informações contidas numa determinada lista `FeedRSS` desejada que está registrada no *Web Service*, pelo método `searchNews`. O método possui um argumento que será um vetor contendo todos os `FeedRSS` que o cliente deseja colher por informações.

Uma vez a solicitação chegando ao servidor, esse acessa a Internet em busca das informações contidas nos `FeedRSSs` escolhidos pelo cliente, e uma vez as encontrando, retira de seu conteúdo qualquer elemento que possa confundir o cliente, como, por exemplo, o símbolo "` `", que indica um espaço em branco dentro de um documento HTML, a função trata de retirar o símbolo e coloca em seu lugar efetivamente um espaço em branco sem codificá-lo, deixando o texto totalmente puro, sem formatações, uma vez que no cliente tais formatações poderiam não ser alcançadas. Por fim, o método altera o vetor recebido contendo os elementos `FeedRSS` escolhidos pelo usuário, no qual os vetores internos de sua estrutura armazenam os títulos e resumos das informações colhidas da Internet, e retorna este vetor ao cliente.

O cliente, ao receber o vetor do servidor, processa as informações colhidas e as lista na tela. Toda essa seqüência é mostrada na Figura 25.

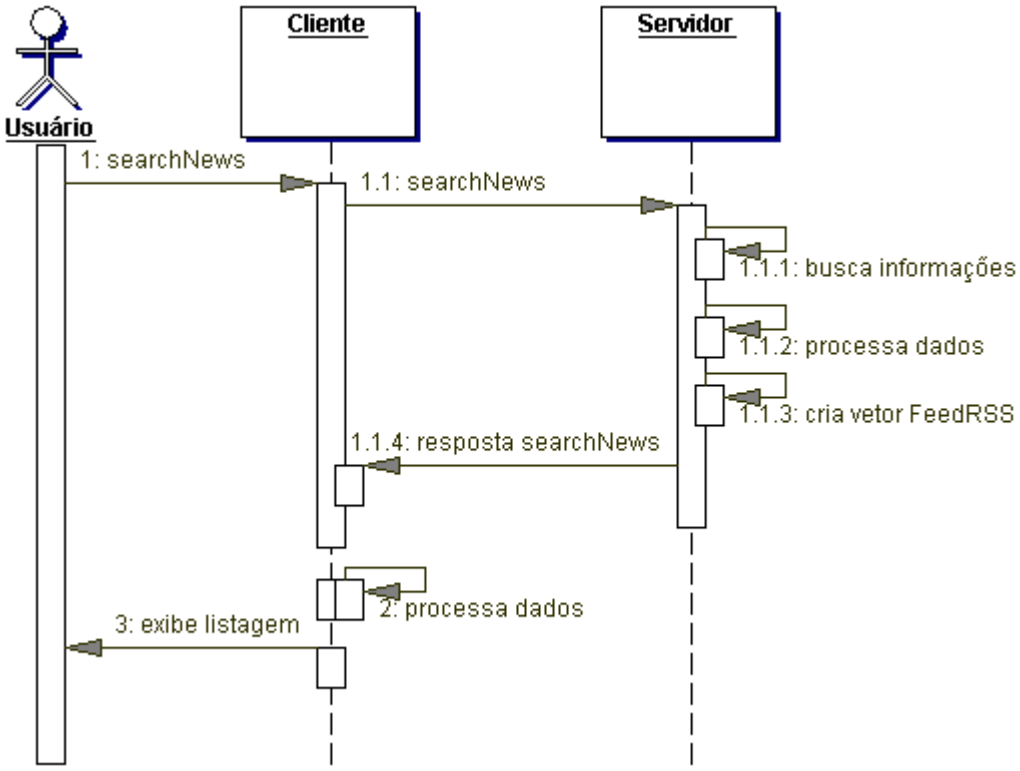


Figura 25. Diagrama de seqüência do método `searchNews`

4.3.3 Escrever Notícia

No decorrer do trabalho observou-se que, além do usuário poder ter acesso a informações de documentos RSS disponíveis publicamente na Internet, também seria interessante o usuário poder atualizar um documento RSS de propriedade dele. Por exemplo, um usuário pode ter uma página *web* pessoal, na qual ele disponibiliza um arquivo RSS contendo as últimas notícias de que lhe aconteceu no dia. Logo, ao passar no vestibular, por exemplo, ele poderia, via seu celular, atualizar seu documento RSS pessoal com tal informação, estando, praticamente no mesmo momento, disponível aos visitantes do *web site* pessoal dele. É claro que esse exemplo pode ser estendido para outras finalidades.

Para esse objetivo, foi disponibilizado no *Web Service* um terceiro método registrado como `writeNews`. O método possui um argumento que será um objeto `FeedRSS` que contém o título e o resumo da informação que o usuário, através do dispositivo, pretende disponibilizar. Além disso, o objeto ainda contém o *link* para o arquivo RSS que será atualizado. É importante salientar que esse arquivo deve estar apto a ser lido e gravado por agentes externos. Esse fato pode causar transtornos ao dono do arquivo uma vez que, se o arquivo é público, qualquer usuário mal intencionado poderia ter acesso a esse arquivo e alterá-lo da forma que desejasse, sendo tais alterações desconhecidas pelo dono do arquivo. Algumas soluções são mencionadas no Capítulo 5.

Uma vez a requisição chegando ao servidor, esse acessa a Internet em busca do documento RSS que será atualizado com a nova informação desejada pelo cliente, e insere no mesmo o título e resumo passados como atributos no objeto. Por fim, o método retorna ao cliente uma resposta de sucesso ou erro de toda a operação. O cliente por sua vez, ao receber a resposta do servidor, processa a informação recebida e a exibe na tela. Tal seqüência pode ser acompanhada pela Figura 26.

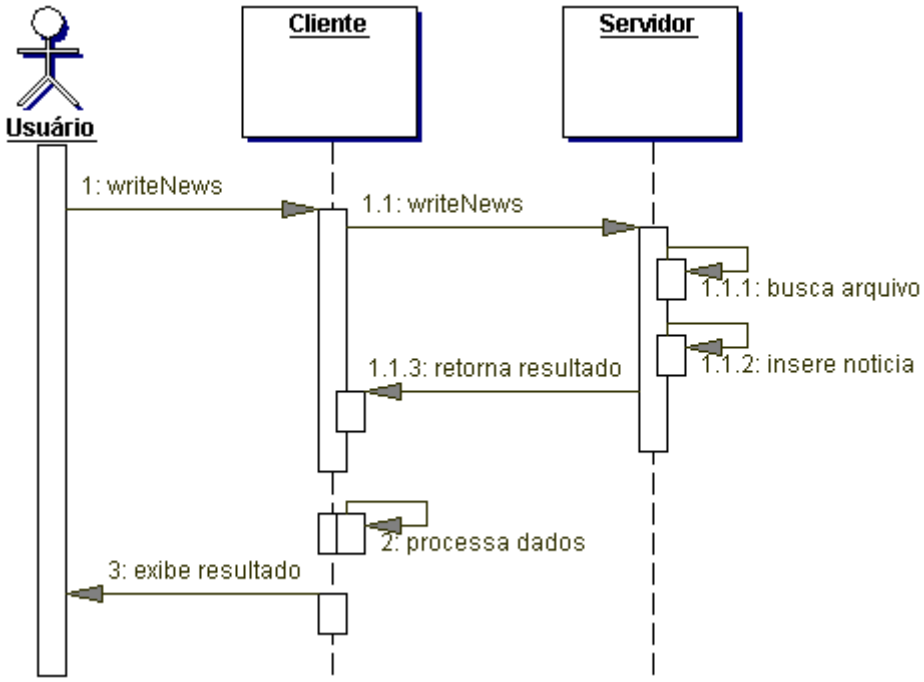


Figura 26. Diagrama de seqüência do método writeNews

4.3.4 Outros requisitos funcionais

Além dos métodos já descritos, outros requisitos funcionais estão presentes no aplicativo RSSAnywhere, lado cliente da aplicação. Esses requisitos são essenciais para o bom funcionamento do aplicativo.

Não trataremos detalhes desses requisitos, pois acreditamos não ser de interesse no tema central do projeto, uma vez que não estão diretamente responsáveis por transmitir dados com o servidor *Web Services*. A Figura 27 mostra as demais funcionalidades desenvolvidas através de um diagrama de casos de uso.

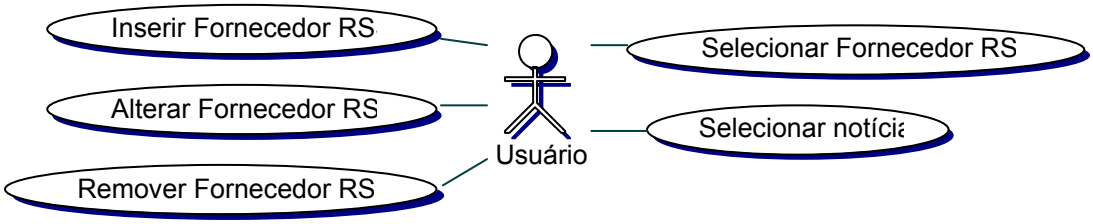


Figura 27. Outros requisitos funcionais do aplicativo RSSAnywhere

4.4 Implementação

A implementação do estudo de caso está dividida em dois projetos já citados, que são:

1. a implementação do servidor que será o *Web Services* (projeto **RSSAnywhereService**); e
2. a implementação do cliente para dispositivos móveis em J2ME (projeto ***RSSAnywhere***).

Seus códigos e características serão mostrados nas Seções seguintes.

4.4.1 RSSAnywhereService

No servidor encontramos o *Web Services* que está publicamente disponível. As classes e suas associações que compõem o RSSAnywhereService podem ser vistas no diagrama de classes UML ilustrado na Figura 28, omitindo seus atributos e métodos.

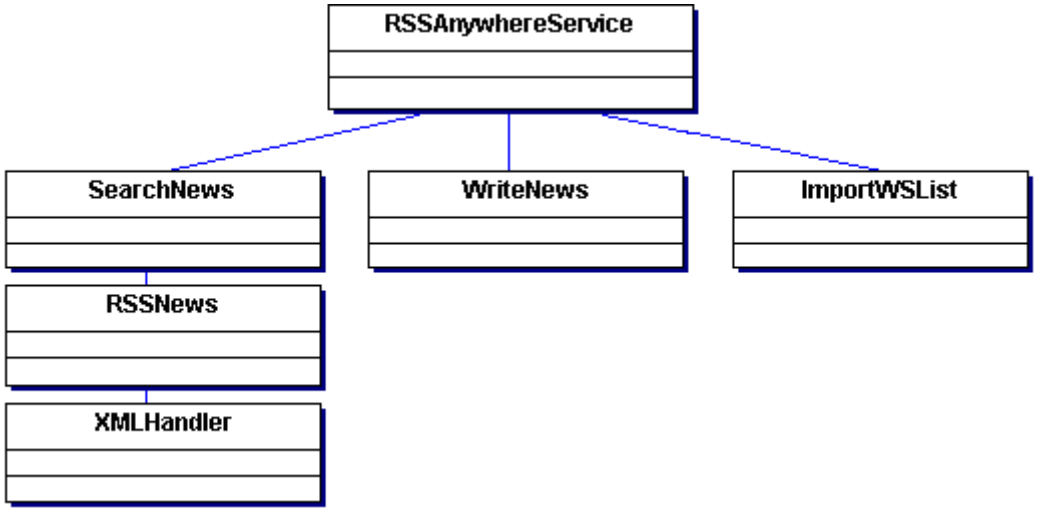


Figura 28. Diagrama de classe RSSAnywhereService

A classe `RSSAnywhereService` é a principal classe do *Web Service*. Nela encontramos os métodos que estarão publicamente disponíveis para serem acessados por outras aplicações. Seu código assemelha-se a qualquer outra aplicação escrita em Java, como podemos ver pela Figura 29.

```
import java.util.*;
public class RSSAnywhereService {

    public RSSAnywhereService(){}

    public Vector searchNews(Vector temp) {
        SearchNews varSearchNews = new SearchNews(temp);
        return varSearchNews.getNews();
    }
    public String writeNews(FeedRSS paramFeedRSS) {
        WriteNews varWriter = new WriteNews(paramFeedRSS);
        return varWriter.updateNews();
    }
    public Vector importWSList() {
        ImportWSList varImportWSList = new ImportWSList();
        return varImportWSList.getList();
    }
}
```

Figura 29. Código da classe RSSAnywhereService

Pode-se perceber a definição de três métodos: `searchNews` – responsável pela busca

das informações contidas em arquivos RSS que são definidos nos objetos `FeedRSS` passado como parâmetro, `writeNews` – método que busca e insere uma informação num arquivo RSS indicado pelo usuário da aplicação cliente, e `importWSList` – responsável por retornar a lista de `FeedRSS` disponível na aplicação servidora.

A estrutura da classe `FeedRSS` é visualizada no esquema da Figura 30. Na qual definimos os atributos e métodos *get* e *set* para acessá-los. São esses objetos `FeedRSS` que são transportados entre as aplicações cliente-servidor empacotados com SOAP.

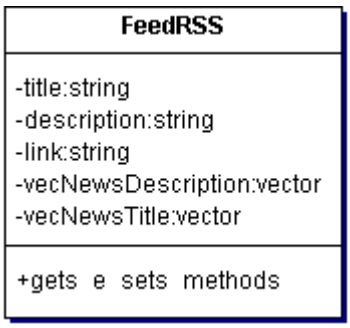


Figura 30. Classe servidora FeedRSS

Para serem transportados pelo SOAP, através do HTTP, esses objetos precisam ser serializados e deserializados. No caso do cliente, esse objeto precisa implementar a interface `kvmSerializable` e re-escrever quatro métodos, cujo código será visto mais adiante. No caso do servidor, a classe `FeedRSS` precisa implementar a interface `Serializable` através dos métodos `serialize` e `deserealize`. Dessa forma, alterando a classe `FeedRSS` encontrada no servidor, ficamos com a estrutura indicada na Figura 31.

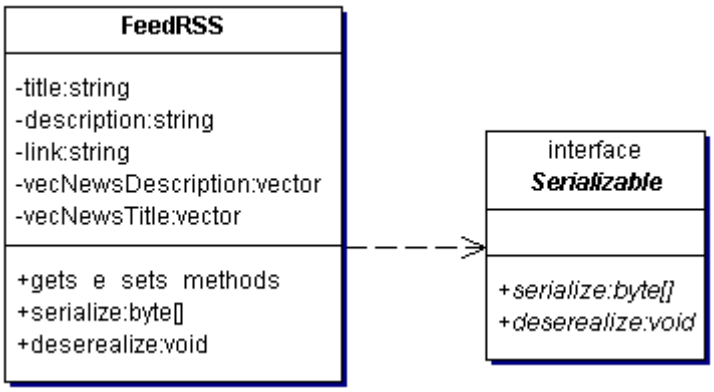


Figura 31. Classe servidora FeedRSS com a interface Serializable

Vale salientar que ao serializar um objeto que será transmitido como parâmetro, estaremos criando uma estrutura no nível de bytes que possibilitará que o lado receptor, seja ele o cliente ou o servidor, recupere o mesmo objeto através da operação inversa de desserialização. Assim, um grande número de parâmetros pode ser encapsulado em um único objeto e transmitidos como um único parâmetro: o objeto serializado. O trecho de código que mostra essa implementação é ilustrado na Figura 32.

```
public class FeedRSS implements java.io.Serializable {
...
  /**
   * Get Custom Serializer
   */
  public static org.apache.axis.encoding.Serializer getSerializer(
        java.lang.String mechType,
        java.lang.Class _javaType,
        javax.xml.namespace.QName _xmlType) {
     return
        new  org.apache.axis.encoding.ser.BeanSerializer(
          _javaType, _xmlType, typeDesc);
  }
  /**
   * Get Custom Deserializer
   */
  public static org.apache.axis.encoding.Deserializer getDeserializer(
         java.lang.String mechType,
         java.lang.Class _javaType,
         javax.xml.namespace.QName _xmlType) {
     return
         new  org.apache.axis.encoding.ser.BeanDeserializer(
           _javaType, _xmlType, typeDesc);
  }
  ...
}
```

Figura 32. Trecho do código da classe FeedRSS implementado a interface Serializable

Não entraremos em detalhes da serialização e desserialização realizada pelo pacote, uma vez que todo processo de transformar os dados em bytes é feito pelo Apache Axis, de forma transparente ao usuário, no momento que o mesmo está empacotando ou desempacotando a mensagem SOAP.

Uma vez implementada a interface, a classe `FeedRSS` está pronta para ser transmitida via SOAP pelo servidor. Analisaremos então os métodos disponíveis no *Web Service*, começando pelo `importWSList`;

importWSList

O método `importWSList`, ao ser invocado, pelo cliente, cria um objeto da classe `ImportWSList` repassando a solicitação do método do serviço. Esse objeto por sua vez trata da listagem de `FeedRSS` disponíveis para acesso pelo cliente do serviço. Por exemplo, o usuário cliente quer acessar informações de uma fonte de notícias internacionais especializada em índices de investimentos, mas esse usuário não lembra no momento do endereço (*url*) para o documento RSS que o *web site* especializado disponibiliza. Então, ele solicita ao *Web Services* a listagem de fontes (`FeedRSS`) que estão disponíveis através do método `importWSList`. Ao receber essa listagem, o usuário pode consultá-la à procura de fontes que tratem do tipo de informação que o interessa e, tendo sorte, pode encontrar o respectivo *web site* desejado dentro da listagem.

No nosso caso, as fontes disponíveis no *Web Service* são apenas ilustrativas de como funcionaria o acesso a essa listagem, pois nossa preocupação é com a disponibilização das informações e não com seu conteúdo. Isso se deve também ao fato de não termos uma pesquisa fundamentada de quais fontes RSS (*web sites, blogs*, etc) poderiam estar disponíveis no nosso *Web Service*, uma vez que existe uma dezena de milhares de fontes disponíveis na Internet, pela crescente utilização e aceitação de arquivos RSS por parte dos sites e blogs existentes.

O código java da classe `ImportWSList` que implementa essa listagem é visto na Figura 33. A parte principal do código encontra-se na terceira linha que mostra a definição de uma variável do tipo vetor chamada `vecFeedRSS` que contém a lista de `FeedRSS` disponível no servidor, que será retornada ao cliente solicitante, sendo preenchida pelo método `getList` encontrado na classe.

```
1 import java.util.*;
2 public class ImportWSList {
3   private Vector vecFeedRSS = new Vector();
4   public ImportWSList(){}

   public Vector getList() {
        vecFeedRSS.addElement(
                  new FeedRSS("Title 1 WS List",
                              "Description Title 1 WS List",
                              "http://localhost:8080/site1.rss") );
        vecFeedRSS.addElement(
                  new FeedRSS("Title 2 WS List",
                              "Description Title 2 WS List",
                              "http://localhost:8080/site2.rss") );
        vecFeedRSS.addElement(
                  new FeedRSS("Title 3 WS List",
                              "Description Title 3 WS List",
                              "http://localhost:8080/site3.rss") );
        vecFeedRSS.addElement(
                  new FeedRSS("Title 4 WS List",
                              "Description Title 4 WS List",
                              "http://localhost:8080/site4.rss") );

        vecFeedRSS.addElement(
                  new FeedRSS("Title 5 WS List",
                              "Description Title 5 WS List",
                              "http://localhost:8080/site5.rss") );

        return vecFeedRSS;
    }
}
```

Figura 33. Código Java da classe ImportWSList

Uma vez o usuário escolhendo seus `FeedRSS`, digamos que ele queira buscar pelas informações contidas nessas fontes. Para isso, o usuário fará uma requisição ao *Web Services* passando a listagem escolhida para o método `searchNews`, que estudaremos a seguir.

searchNews

O método `searchNews` cria uma instância da classe `SearchNews` e, por meio dessa faz a busca pelas fontes solicitadas e processa as informações colhidas, retornando as mesmas fontes solicitadas contendo agora as informações encontradas nos documentos RSS pesquisados. Internamente, a classe `SearchNews` faz uso da classe `RSSNews`. Essa última, por sua vez, contém os métodos de busca e processamento das informações encontradas nas fontes `FeedRSS` analisadas.

Lembramos que as informações estão contidas em documentos RSS que têm sua estrutura baseada no padrão XML. Logo, a busca pelas informações contidas na listagem `FeedRSS` definida pelo cliente é feita através de um processamento num arquivo XML. Como estamos

tratando de um cliente que é um dispositivo móvel, a velocidade de processamento desses arquivos XML está diretamente ligada ao tempo gasto para retornar a resposta ao dispositivo cliente, por isso é preciso usar uma API Java capaz de ser tão rápida quanto possível para essa tarefa. Dessa forma, a API escolhida para esta tarefa foi a SAX [39].

O SAX utiliza uma abordagem baseada em eventos Assim, à medida que a mensagem é lida, eventos são disparados, tornando seu processo rápido com uso mínimo de memória da máquina servidora. Nessa abordagem, a aplicação necessita implementar tratadores de eventos para manipular cada tipo de evento gerado. A classe `DefaultHandler` provê uma implementação para tratar todos os eventos gerados pelo parser.

Alguns dos principais eventos gerados pelo SAX são:

- *Start document*: Disparado no início do documento;
- *Start element*: Disparado a cada novo elemento. Apenas o nome do elemento está disponível;
- *Characters*: Conteúdo de cada elemento;
- *End element*: Disparado no final de cada elemento;
- *End document*: Disparado ao final do documento.

No nosso contexto, foi criada a classe `XmlHandler` que estende da classe `DefaultHandler`. A Figura 34 mostrar os trechos de códigos da classe `RSSNews` e a Figura 35 da classe `Xmlhandler`.

```
1 public class RSSNews {
2   private Vector vcNewsTitle = new Vector();
3   private Vector vcNewsDescription = new Vector();
4 ...
5   public void getNews(String _link){
6       String arquivo;
7       arquivo = _link;
8       try {
9          // início do parser //
10         // cria a factory e o parser
11         SAXParser parser =
12                   SAXParserFactory.newInstance().newSAXParser();
13         // abre a conexão com o arquivo
14         InputSource input = new InputSource(arquivo);
15         // inicia o parsing
16         parser.parse(input, new XMLHandler(this));
17     }
18 ...
19 }
```

Figura 34. Trecho do código da classe RSSNews

No caso da classe `RSSNews`, pode-se observar a definição nas linhas 2 e 3 de dois vetores *`vcNewsTitle`* e *`vcNewsDescription`* os quais conterão os títulos e os resumos das notícias encontradas nos arquivos RSS, respectivamente. No método `getNews`, responsável pela inicialização da busca dos arquivos RSS escolhidos pelo usuário, criamos (linha 11) um objeto da classe *`SAXParse`* que conterá o link para arquivo RSS naquele momento e a classe que será responsável pelo processamento do mesmo. Na linha 14, referenciamos o link em questão e na linha 16 iniciamos o parser para processar o documento RSS, informando a classe `XMLHandler` programada especialmente para processá-lo.

Já a classe `XMLHandler`, uma vez marcada como `Handler` do parser em questão começa a trabalhar com eventos. Para cada início de tag XML encontrada no arquivo RSS, a classe dispara um evento, para cada final ela dispara outro evento e assim por diante. O processamento do arquivo pode ser visto pelo trecho de código da classe na Figura 35.

```
class XMLHandler extends DefaultHandler {

   /** o galho atual */
   private StringBuffer galhoAtual = new StringBuffer(200);
   /** o valor da tag atual */
   private StringBuffer valorAtual = new StringBuffer(100);
...
   /** final da tag */
   public void endElement(String uri, String localName, String tag)
   {
       if( !isItem && galhoAtual.toString().startsWith("item") ) {
         isItem = true;
       }
       if ( isItem ){
          if ( galhoAtual.toString().startsWith("title") ) {
            varRSSNews.addTitleNews(valorAtual.toString().trim());
          }
          if ( galhoAtual.toString().startsWith("description") ) {
            varRSSNews.
                  addDescriptionNews(valorAtual.toString().trim());
          }
      }
      ...
   }
...
}
```

Figura 35. Trecho de código da classe XMLHandler

A Figura 35 mostra apenas um dos quatro métodos que devemos sobrescrever da classe `DefaultHandler`, sendo no nosso caso, o método mais importante o `endElement`. O atributo `galhoAtual` serve para marcar em qual tag XML o *parser* do arquivo se encontra no momento, enquanto o atributo `valorAtual` contém a informação contida entre *tags*, como por exemplo: *<tag>informação contida no valorAtual</tag>*. Como estamos buscando apenas as informações contidas na *tag* `<item>` do documento RSS, analisamos se a *tag* atual é um `item` e, posteriormente, analisamos a *tag* seguinte, e caso seja um `title` ou `descrpition`, gravamos seu conteúdo no vetor de títulos e descrições contidos no objeto `RSSNews`, respectivamente.

No final do processamento, o vetor de títulos e descrições de cada RSS encontrado e processado da listagem de `FeedRSS` solicitadas pelo cliente estarão preenchidos com as informações colhidas, sendo retornadas ao dispositivo solicitante. A visualização dessas informações no cliente e as mensagens transmitidas pelo cliente e servidor serão relacionados na Seção de Interface Gráfica ainda neste Capítulo.

writeNews

O terceiro método disponível no *Web Service*, é o `writeNews`. Vale lembrar que esse método é responsável por atualizar um documento RSS informado pelo usuário do aplicativo cliente através do objeto `FeedRSS`, passado como parâmetro através do SOAP. Assim que invocado,

ele instancia um objeto da classe `WriteNews`, passando como parâmetro o `FeedRSS` recebido, nele teremos o link para o arquivo RSS que será atualizado, o título e o resumo da informação que serão gravadas.

Como pretendemos adicionar uma nova informação (um novo elemento `item`) ao documento RSS, suas informações antigas devem ser preservadas. Ou seja, será necessário criar um novo documento XML baseado no arquivo original incluindo as informações passadas. Para isso, a API SAX não se adequa, uma vez que seu funcionamento está baseado em eventos, não guardando nenhuma informação em memória, o que nos forçaria a percorrer todo o documento RSS, gravando todos os elementos numa estrutura em memória para depois adicionarmos a nova informação e, por fim, gravar o novo documento. Ao invés disso, usaremos uma outra API java que se adequa melhor a esta nova situação, o DOM. Usando DOM, ao lermos um documento XML, seu processamento guarda todos os elementos encontrados no documento em memória, deixando o acesso à sua estrutura bastante flexível. Obviamente há um maior processamento nesta API assim como num maior consumo de memória. Mas, como estamos processando do lado do servidor, memória não é um complicador e como estamos, no momento, tratando apenas de uma atualização (um arquivo RSS), o tempo de processamento não será elevado como se fossemos usar o DOM no método `searchNews`, por exemplo.

O trecho de código implementado no trabalho para criar um novo documento DOM, é visto na Figura 36.

```
1 private Document parseXmlFile(String link) {
2    try {
3       // Create a builder factory
4       DocumentBuilderFactory factory =
5                                 DocumentBuilderFactory.newInstance();
6       // Create the builder and parse the file
7       Document doc = factory.newDocumentBuilder().parse(link);
8       return doc;
9     }
10  ...
11  return null;
12 }
```

Figura 36. Trecho de código da classe writeNews (parseXmlFile)

Para tratar o documento RSS usando o DOM, primeiramente, como visto na linha 4, criamos uma fábrica de *parsers* DOM através da classe `DocumentBuilderFactory`. Em seguida validamos o documento RSS para garantir que o documento é bem formado e válido (linha 7), criamos um novo documento DOM através do método `newDocumentBuilder`. Por fim, retornamos uma nova instância de um documento DOM para ser processado.

Dessa forma, todo o documento RSS (nodos e filhos) estará em memória referenciado pela variável `doc`. Assim, podemos acessar, atualizar, adicionar ou apagar qualquer nodo ou filho que se queira. Uma vez feita as alterações desejadas, para se gravar este novo documento RSS alterado precisamos transformar a estrutura de memória do DOM num arquivo de texto a ser gravado, esta transformação pode ser visualizada na Figura 37 que mostra o trecho de código responsável por isso.

```
1 private String writeXmlFile(Document doc, String link) {
2   try{
3     Transformer transformer =
4                   TransformerFactory.newInstance().newTransformer( );
5     Source source = new DOMSource( doc );
6     StreamResult output = new StreamResult(
7                     new FileOutputStream(new File(link)));
8     transformer.transform( source, output );
9     return "sucesso";
10   }
```

Figura 37. Trecho de código da classe writeNews (writeXmlFile)

Uma transformação precisa de uma fonte e um destino. A fonte será um `DOMSource` que recebe o documento DOM que alteramos em memória como parâmetro (linha 5). O destino será um `StreamResult` que recebe um `File` onde será escrito o novo XML processado (linha 6 e 7). O objeto Java responsável pela transformação é o `Transformer`, sua instância é obtida do `TransformerFactory` (linhas 3 e 4) e a transformação ocorre após a chamada do método `transform` do objeto `Transformer` na linha 8.

Tanto o método `parseXmlFile` quanto o `writeXmlFile` , serão usados pelo método `updateNews` da classe `writeNews` para atualizar o arquivo RSS do usuário com a nova informação desejada. Lembre-se que um documento RSS pode conter vários elementos `item`, e precisamos adicionar um novo elemento que seja o primeiro no arquivo, uma vez que a ordem em que aparecem no documento, normalmente, reflete os mais atuais serem os primeiros. A Figura 38 mostra este novo método.

```
1 public String updateNews()
2 {
3   Document doc = parseXmlFile( link );
4   NodeList list = doc.getElementsByTagName("item");
5
6   Element root = doc.createElement("item");
7   Element title = doc.createElement("title");
8   Element description = doc.createElement("description");
9   title.appendChild(doc.createTextNode(this.title));
10  description.appendChild(doc.createTextNode(this.description));
11  root.appendChild(doc.createTextNode("\n\t"));
12  root.appendChild(title);
13  root.appendChild(doc.createTextNode("\n\t"));
14  root.appendChild(description);
15  root.appendChild(doc.createTextNode("\n\t"));
16
17  Element element = (Element)list.item(0);
18  element.getParentNode().insertBefore(root , element);
19
20  return writeXmlFile(doc, link);
21 }
```

Figura 38. Código do método updateNews da classe writeNews

Pela Figura 38, podemos analisar a criação de um novo elemento `item`. A criação do novo elemento item, título e resumo (linhas 6, 7 e 8); o preenchimento deles com as novas informações desejadas pelo usuário do dispositivo cliente (linhas 9 e 10); a inserção do novo item anterior ao primeiro elemento `item` encontrado no arquivo (linhas 17 e 18) e o retorno do resultado da gravação desse novo elemento no mesmo documento RSS pesquisado (linha 20).

4.4.2 Registrando o Serviço Web

Nota-se que toda implementação até agora, foi feita livre de preocupações com o registro do *Web Services* no servidor de aplicações. A total preocupação, até o momento, foi desenvolver as classes e os métodos Java necessários para o seu funcionamento e o Axis se encarrega de transformar os dados das requisições e respostas em mensagem SOAP e transmitir via HTTP.

Para registrar o *Web Service* no Apache Axis, podemos fazer de duas formas. A primeira consiste em um registro rápido de serviços conhecido por *Simples Classes Java* que são arquivos Java com extensão .JWS ao invés de .JAVA. Neste caso os detalhes do serviço disponibilizados, suas estruturas e métodos ficam a cargo do próprio Axis, deixando o desenvolvedor livre de aspectos mais técnicos, o que acarreta numa perda de flexibilidade e de gerenciamento. Já a segunda forma de registro favorecido por este pacote, é através de um *Web Services deployment descritor* (WSDD, descritor de organização de serviços web) baseado em XML que proporciona ao provedor de serviços fornecer os detalhes do serviço ao ambiente Axis em tempo de execução como definir quais serviços serão disponibilizados, controladores e objetos de transporte a serem organizados ao mecanismo deste pacote.

Por isso, vamos utilizar um arquivo WSDD para registrar nosso *Web Service*. O descritor de organização de nosso *Web Service* - *deploy.wsdd* - é mostrado na Figura 39.

```
1 <deployment xmlns="http://xml.apache.org/axis/wsdd/"
2         xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
3
4     <service name="urn:RSSAnywhereService" provider="java:RPC">
5         <parameter name="className" value="RSSAnywhereService"/>
6         <parameter name="allowedMethods" value="*"/>
7         <beanMapping qname="myNS:FeedRSS"
8                      xmlns:myNS="urn:RSSAnywhereService1"
8                      languageSpecificType="java:FeedRSS"/>
10    </service>
11 </deployment>
```

Figura 39. Descritor de organização do Web Service RSSAnywhereService

O elemento raiz de um documento WSDD pode conter o elemento `<deployment>` que notifica ao mecanismo Axis para organizar um novo serviço controlador ou pode conter o elemento `<undeployment>` para informar ao mesmo para não organizar um determinado serviço. Pela linha 1 da Figura 39, vemos que o elemento mais externo é o `<deployment>`, informando ao Axis que este arquivo trata de uma organização de serviço. O elemento `<service>` especifica o nome do serviço e o provedor para o controle do serviço (linha 4). O nome do serviço, `urn:RSSAnywhereService`, é o que estará disponível publicamente para os demais sistemas que queiram acessar os métodos e funcionalidades por ele implementados. Já o controlador é do tipo `java:RPC`, para indicar que o serviço é do tipo Java e que só será acessado por meio de chamada a procedimentos remotos, o qual é implementado pelo mecanismo do Axis através da classe `org.apache.axis.providers.RPCProvider`.

Para garantir que o serviço crie um instância correta de suas funcionalidades, além de definir um nome para o serviço em questão, o elemento `<service>` deve conter o elemento `<parameter>` a fim de especificar o nome da classe que efetivamente implementa o *Web Service* (linha 5). Podemos ainda constatar através da Figura 39 que todos os métodos contidos na classe `RSSAnywhereService` são incluídos no registro (linha 6). Ainda analisando o descritor, podemos ver o mapeamento de um tipo SOAP `myNS:FeedRSS` para um tipo Java `java:FeedRSS` (linhas 7, 8 e 9). Isto permite que o Axis mapeie corretamente um tipo no

outro que não se encontra no mapeamento padrão dele.

O Axis, oferece ainda uma ferramenta, *org.apache.axis.client.AdminClient*, para registrar um *Web Services* que usa um descritor de organização. Para isso, executamos a ferramenta passando como parâmetro de entrada o arquivo *deploy.wsdd* a partir do comando no *prompt* do *windows*, como exemplificado na Figura 40.

```
> java org.apache.axis.client.AdminClient deploy.wsdd
Processing file deploy.wsdd
<Admin>Done processing</Admin>
```

Figura 40. Processo de deploy do Web Service no Axis

Pode-se conferir os *Web Services* registrados no Axis pela url: http://localhost:8080/axis/servlet/AxisServlet. Para se ter acesso ao arquivo WSDL gerado, podemos obtê-lo automaticamente acessando a url http://localhost:8080/axis/services/urn:RSSAnywhereService?wsdl. Lembre-se que o arquivo WSDL é usado por qualquer agente que queira ter acesso ao nosso *Web Service,* encontrando no mesmo, todos os métodos, parâmetros e respostas definidos.

4.4.3 RSSAnywhere

No cliente encontramos o aplicativo RSSAnywhere que faz acesso ao *Web Services* criado e disponibilizado no servidor *web*. As classes e suas associações que compõem o aplicativo em questão podem ser vistos no diagrama de classes ilustrado na Figura 41, omitindo seus atributos e métodos.

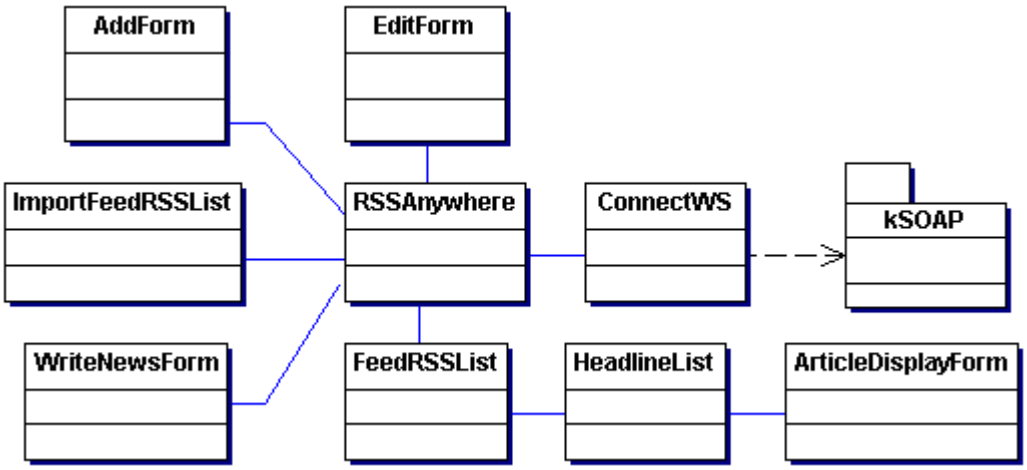


Figura 41. Diagrama de classe RSSAnywhere

Pela Figura 41, vemos as diversas classes que compõem o aplicativo RSSAnywhere. Focando na integração e comunicação do aplicativo com o *Web Service*, não entraremos em detalhes em todas as classes, uma vez que, por exemplo, a classe `AddForm`, responsável por adicionar um novo `FeedRSS` na lista do usuário, não está relacionada diretamente com a comunicação com o *Web Services*.

Vimos anteriormente que no servidor precisamos alterar a classe `FeedRSS` para ser transportada pelo mecanismo do Axis através do SOAP, implementando a interface `Serializable`. No caso do cliente acontece algo semelhante, a diferença está no fato da classe

FeedRSS do cliente, ter que implementar a interface kvmSerializable e quatro métodos que serão acessado pelo kSOAP, como mencionamos no Capítulo 2.

A estrutura inicial da classe FeedRSS é igual ao esquema da Figura 30 do servidor. Na qual definimos os atributos e métodos *get* e *set* para acessá-los. Lembrando que serão esses objetos que serão transportados entre as aplicações cliente-servidor empacotados com SOAP. Porém, a nova classe FeedRSS encontrada no cliente, implementando a interface kvmSerializable, é indicada na Figura 42.

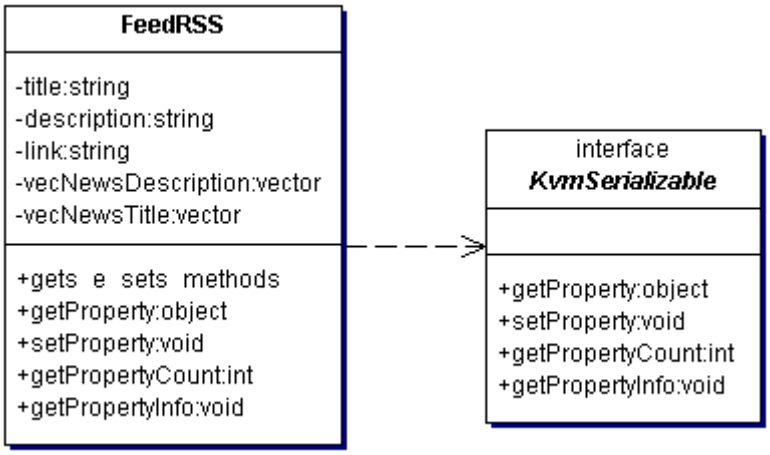


Figura 42. Classe cliente FeedRSS com a interface kvmSerializable

No kSOAP, todas as variáveis do objeto a ser transferido têm que ser definidas como objetos do tipo PropertyInfo na aplicação. A Figura 43 mostra o trecho de código de como isso é feito na classe FeedRSS.

O primeiro parâmetro do construtor de PropertyInfo é o nome do parâmetro a ser usado pelo kSOAP e o segundo é o tipo Java a que este parâmetro será mapeado (linhas 10 a 19). Este segundo parâmetro pode ser qualquer um dos tipos que o kSOAP pode mapear diretamente (ver Tabela 2). Para qualquer classe que implemente a interface KvmSerializable, todos os objetos de PropertyInfo necessitam ser agrupados em uma estrutura de modo que o kSOAP possa se manter a par de todos os parâmetros existentes. No nosso caso, a estrutura criada foi a de um vetor (linha 21) no qual todos os objetos de PropertyInfo são adicionados ao mesmo quando o construtor da classe é chamado (linhas 28 a 32). A variável PROP_COUNT contem a quantidade de objetos PropertyInfo no vetor.

```
1 public class FeedRSS implements KvmSerializable {
2    private String title, description, link;
3    private Vector vecNewsTitle, vecNewsDescription;
4
5    /*
6    *  Used by KSOAP Implementation methods
7    */
8    private static int PROP_COUNT = 5;
9
10   private static PropertyInfo PI_title =
11            new PropertyInfo("title", ElementType.STRING_CLASS);
12   private static PropertyInfo PI_description =
13            new PropertyInfo("description", ElementType.STRING_CLASS);
14   private static PropertyInfo PI_link =
15            new PropertyInfo("link", ElementType.STRING_CLASS);
16   private static PropertyInfo PI_vecNewsTitle =
17            new PropertyInfo("vecNewsTitle", ElementType.VECTOR_CLASS);
18   private static PropertyInfo PI_vecNewsDescription =
19      new PropertyInfo("vecNewsDescription", ElementType.VECTOR_CLASS);
20
21   private Vector PI_Vector = new Vector();
22
23   public FeedRSS(String title, String description, String link)
24   {
25      this.title = title;
26      this.description = description;
27      this.link = link;
28      this.PI_Vector.addElement(PI_title);
29      this.PI_Vector.addElement(PI_description);
30      this.PI_Vector.addElement(PI_link);
31      this.PI_Vector.addElement(PI_vecNewsTitle);
32      this.PI_Vector.addElement(PI_vecNewsDescription);
33   }
34 ...
```

Figura 43. Objetos PropertyInfo da classe cliente FeedRSS

Uma vez definidos os objetos de `PropertyInfo`, sobrescrevemos os métodos (ver Tabela 3) para acessá-los. O trecho de código que mostra tais métodos implementados na classe cliente `FeedRSS` é ilustrado na Figura 44.

Todo o processo interno do kSOAP para serializar e desserializar os dados transmitidos, não será abordado neste trabalho, uma vez que esta etapa de transformar os dados em bytes é feito internamente pelo pacote de forma transparente ao programador e usuário no momento que o kSOAP está empacotando ou desempacotando a mensagem SOAP, o que foge do escopo desta monografia.

```
/*
* Implement Logic for kSOAP
*/
public Object getProperty(int param){
    if (param == 0){
       return getTitle();
    } else if (param == 1){
       return getDescription();
    } else if (param == 2){
       return getLink();
    } else if (param == 3){
       return getVecNewsTitle();
    } else if (param == 4){
       return getVecNewsDescription();
    } else {
       return null;
    }
}
public void setProperty(int param, Object obj){
    if (param == 0){
       setTitle((String) obj);
    } else if (param == 1){
       setDescription((String) obj);
    } else if (param == 2){
       setLink((String) obj);
    } else if (param == 3){
       setVecNewsTitle((Vector) obj);
    } else if (param == 4){
       setVecNewsDescription((Vector) obj);
    } else {}
}
public int getPropertyCount(){
    return this.PI_Vector.size();
}
public void getPropertyInfo(int param,
                            PropertyInfo propertyInfo){
    PropertyInfo PI = (PropertyInfo) PI_Vector.elementAt(param);
    propertyInfo.name = PI.name;
    propertyInfo.nonpermanent = PI.nonpermanent;
    propertyInfo.copy(PI);
}
```

Figura 44. Implementação dos métodos da interface kvmSerializable

Assim como o Axis precisa fazer o mapeamento de tipos SOAP para tipos Java através do descritor de organização, *deploy.wsdd*, o kSOAP também precisa definir este mapeamento. O mapeamento no kSOAP é feito através da classe `ClassMap` e um trecho de código é apresentado na Figura 45.

```
ClassMap classMap = new ClassMap()
classMap.addMapping("urn:RSSAnywhereService1",
                    "FeedRSS",
                    new FeedRSS().getClass())
```

Figura 45. Mapeamento entre tipo SOAP e tipo Java no kSOAP

Veremos mais adiante onde este mapeamento será inserido no código dos métodos que se comunicam com o *Web Service*. Uma vez implementada a interface do kSOAP e feito o

mapeamento dos objetos transmitidos, a classe `FeedRSS` esta pronta para ser.

importWSList

O acesso do cliente ao método `importWSList` do *Web Service* segue o roteiro:

1. O aplicativo é iniciado pela classe `RSSAnywhere`;
2. Uma vez o comando de importar lista do servidor acionado pelo usuário da aplicação, a classe cria uma instância da classe `ImportFeedRSSList`. A classe `ImportFeedRSSList` é responsável por criar uma estrutura interna para guardar os `FeedRSS` retornados pelo serviço e gerar uma listagem para o usuário do aplicativo;
3. Uma vez a classe `ImportFeedRSSList` instanciada, esta chama o método `importWSList` da classe `ConnectWS`. A classe `connectWS` é responsável em criar a chamada ao *Web Service*, criar os parâmetros e mapeamentos necessários e processar a resposta recebida pelo servidor.

Das três classes necessárias, a Figura 46 mostra o trecho de código da classe `ConnectWS` destacando o método `importWSList`.

Nas linhas 1 a 3, importamos as classes do kSOAP necessárias para a comunicação com o *Web Services*: `HttpTransport` – o protocolo de rede utilizado, `SoapObject` – que será a mensagem enviada/recebida empacotada por SOAP e `ClassMap` – para mapear nosso tipo `my:FeedRSS` SOAP no tipo `FeedRSS` Java que definimos. Em seguida declaramos algumas variáveis que serão usadas durante o método, destacando a string *`SERVICE_URL`*, que indica a url onde o serviço está hospedado (linha 5).

Já no método (linha 10), definimos na variável de transporte a url do *Web Service* em questão (linha 14), a ação SOAP que será realizada (linha 15), e definimos o depurador do protocolo como ativo, para que, em caso de falhas durante o processo de envio/recebimento da mensagem possamos verificar as possíveis causas para as mesmas. Logo após, mapeamos os tipos criados e adicionamos ao protocolo de transporte (linhas 18 a 22) e por fim, um objeto Soap (`SoapObject`) é instanciado, chamado `client`, com o intuito de criar a mensagem SOAP.

Após a chamada ao serviço através do comando `ht.call(client)` (linha 29) recebemos como resposta um objeto SOAP enviado pelo *Web Service*. A partir desta resposta, temos acesso a todos os elementos `FeedRSS` retornados do servidor. Perceba que é neste momento que os métodos implementados anteriormente na classe `FeedRSS` para ser transportada pelo kSOAP são utilizadas, para encontrar a quantidade de variáveis transmitidas (linha 30), e para se ter acesso aos dados transmitidos (linhas 31 a 34). Para cada elemento encontrado, criamos um similar no aplicativo (linha 35 e 36) e este é adicionado a classe `ImportFeedRSSList` que cuidará de sua visualização para o usuário do aplicativo cliente (linhas 38 e 39).

Através da classe `ImportFeedRSSList`, o usuário poderá ter acesso ao nome, descrição e link para acesso de qualquer fonte de informação importada do servidor, assim como, poderá guardá-la na sua lista de `FeedRSS` favoritos. Após feita suas escolhas e retornando ao menu principal do aplicativo, a lista dos `FeedRSS` importada é liberada da memória pela mesma classe `ImportFeedRSSList`.

```
1  import org.ksoap.ClassMap;
2  import org.ksoap.SoapObject;
3  import org.ksoap.transport.HttpTransport;
4  ...
5  private String SERVICE_URL = "http://localhost:8070/axis/services";
6  private HttpTransport ht = new HttpTransport();
7  private Vector auxVector = new Vector();
8  protected ImportFeedRSSList lsImportFeedRSS = null;
9  ...
10 public void importWSList(ImportFeedRSSList _lsImportFeedRSS) {
11
12  lsImportFeedRSS = _lsImportFeedRSS;
13
14  ht.setUrl(SERVICE_URL);
15  ht.setSoapAction("urn:RSSAnywhereService#importWSList");
16  ht.debug = true;
17
18  ClassMap classMap = new ClassMap();
19  classMap.addMapping("urn:RSSAnywhereService1",
20                      "FeedRSS",
21                      new FeedRSS().getClass());
22  ht.setClassMap(classMap);
23
24  SoapObject client = new  SoapObject("urn:RSSAnywhereService",
25                                      "importWSList");
26  SoapObject resp = null;
27  auxVector.removeAllElements();
28  try {
29      resp = (SoapObject) ht.call(client);
30      for (int i = 0 ; i < resp.getPropertyCount(); i++){
31        String title = ( (FeedRSS) resp.getProperty(i) ).getTitle();
32        String description = ( (FeedRSS) resp.getProperty(i) ).
33                                                    getDescription();
34        String link = ( (FeedRSS) resp.getProperty(i) ).getLink();
35        auxFeedRSS = new FeedRSS(title, description, link);
36        auxVector.addElement(auxFeedRSS);
37      }
38      lsImportFeedRSS.setVectorFeedRSS(auxVector);
39      lsImportFeedRSS.showList();
40   }
41 }
```

Figura 46. Trecho de código da classe ConnectWS do método importWSList

searchNews

O acesso ao método `searchNews` do *Web Service* segue o roteiro:

1. O aplicativo é iniciado pela classe `RSSAnywhere`;
2. O usuário acessa sua lista de `FeedRSS` preferidos, e escolhe de quais deseja obter por informações naquele momento;
3. Uma vez o comando de procurar notícias acionado pelo usuário da aplicação, a classe cria uma instância da classe `FeedRSSList`. A classe `FeedRSSList` é responsável em criar um estrutura interna para guardar os `FeedRSS` retornados pelo serviço, as informações encontradas e gerar uma listagem para o usuário do aplicativo;
4. Uma vez a classe `FeedRSSList` instanciada, esta chama o método `searchNews` da classe `ConnectWS`. A classe `connectWS` é responsável em criar a chamada ao

Web Services, criar os parâmetros e mapeamentos necessários e processar a resposta recebida pelo servidor.

A Figura 47 mostra o trecho de código da classe `ConnectWS` contendo o método `searchNews`.

A principal diferença quanto ao método anterior, `importList`, está no fato da função `searchNews` no *Web Service,* receber como parâmetro de entrada um vetor de `FeedRSS` do aplicativo cliente. Logo, devemos adicionar à chamada do método SOAP este parâmetro através do método `addProperty`, encontrado no objeto SOAP instanciado no método (visto na linha 18 da Figura 47). O primeiro parâmetro do método `addProperty` é o mesmo nome da variável que receberá o valor enviado no *Web Services*, enquanto o segundo é a variável local que estamos querendo transmitir.

```
1 public void searchNews(Vector vecFeedRSS, FeedRSSList _lsFeedRSS) {
2
3   auxVector = vecFeedRSS;
4   lsFeedRSS = _lsFeedRSS;
5
6   ht.setUrl(SERVICE_URL);
7   ht.setSoapAction("urn:RSSAnywhereService#searchNews");
8   ht.debug = true;
9
10  ClassMap classMap = new ClassMap();
11  classMap.addMapping("urn:RSSAnywhereService1",
12                      "FeedRSS",
13                      new FeedRSS().getClass());
14  ht.setClassMap(classMap);
15
16  SoapObject client = new  SoapObject("urn:RSSAnywhereService",
17                                      "searchNews");
18  client.addProperty("temp",auxVector);
19
20  SoapObject resp = null;
21   try {
22      resp = (SoapObject) ht.call(client);
23      for (int i = 0 ; i < resp.getPropertyCount(); i++){
24        String title = ( (FeedRSS) resp.getProperty(i) ).getTitle();
25        String description = ( (FeedRSS) resp.getProperty(i) ).
26                                                    getDescription();
27        String link = ( (FeedRSS) resp.getProperty(i) ).getLink();
28        Vector newsTitle = new Vector();
29        Vector newsDesc = new Vector();
30        newsTitle = ((FeedRSS) resp.getProperty(i)).getVecNewsTitle();
31        newsDesc = ((FeedRSS) resp.getProperty(i)).
32                                              getVecNewsDescription();
33        auxFeedRSS = new FeedRSS(title, description, link);
34        auxFeedRSS.setVecNewsTitle(newsTitle);
35        auxFeedRSS.setVecNewsDescription (newsDesc);
36        lsFeedRSS.addElement(auxFeedRSS);
37     }
38     lsFeedRSS.showList();
39 }
```

Figura 47. Trecho de código da classe ConnectWS do método searchNews

Outra diferença encontrada é percebida nos dados recebidos pelo cliente, uma vez que os vetores de títulos e resumos estão preenchidos pelas informações encontradas na busca dos

FeedRSS escolhidos pelo usuário do aplicativo. Portanto, adicionamos a definição de dois vetores que conterão os títulos e descrições processados pelo servidor e os preenchemos com os dados retornados (linhas 28 a 32, respectivamente).

A seqüência do código é similar ao do método importWSList, a qual dispensa comentários.

writeNews

O acesso ao método writeNews do *Web Service* segue o roteiro:

1. O aplicativo é iniciado pela classe RSSAnywhere;
2. Uma vez o comando de escrever notícia acionado pelo usuário da aplicação, a classe cria uma instância da classe WriteNewsForm. A classe WriteNewsForm é responsável em criar um formulário que receberá os dados (título, resumo e url do documento RSS) que serão enviados ao *Web Service*. Após receber a resposta do servidor gera uma mensagem para o usuário do aplicativo;
3. A WriteNewsForm então, invoca o método writeNews da classe ConnectWS. A classe ConnectWS é responsável por criar a chamada ao *Web Services*, criar os parâmetros e mapeamentos necessários e processar a resposta recebida pelo servidor.

Pela Figura 48, analisamos o trecho de código da classe ConnectWS destacando o método writeNews.

```
1 public void writeNews(WriteNewsForm _fmWriteNews, FeedRSS varFeedRSS)
2 {
3
4   fmWriteNews = _fmWriteNews;
5
6   ht.setUrl(SERVICE_URL);
7   ht.setSoapAction("urn:RSSAnywhereService#writeNews");
8   ht.debug = true;
9
10  ClassMap classMap = new ClassMap();
11  classMap.addMapping("urn:RSSAnywhereService1",
12                      "FeedRSS",
13                      new FeedRSS().getClass());
14  ht.setClassMap(classMap);
15
16  SoapObject client = new  SoapObject("urn:RSSAnywhereService",
17                                      "writeNews");
18  client.addProperty("paramFeedRSS",varFeedRSS);
19
20  String resp = null;
21  try {
22     resp = (String)ht.call(client);
23     if (resp.equals("sucesso"))
24         fmWriteNews.show(true);
25     else
26         fmWriteNews.show(false);
27  }
28 }
```

Figura 48. Trecho de código da classe ConnectWS do método writeNews

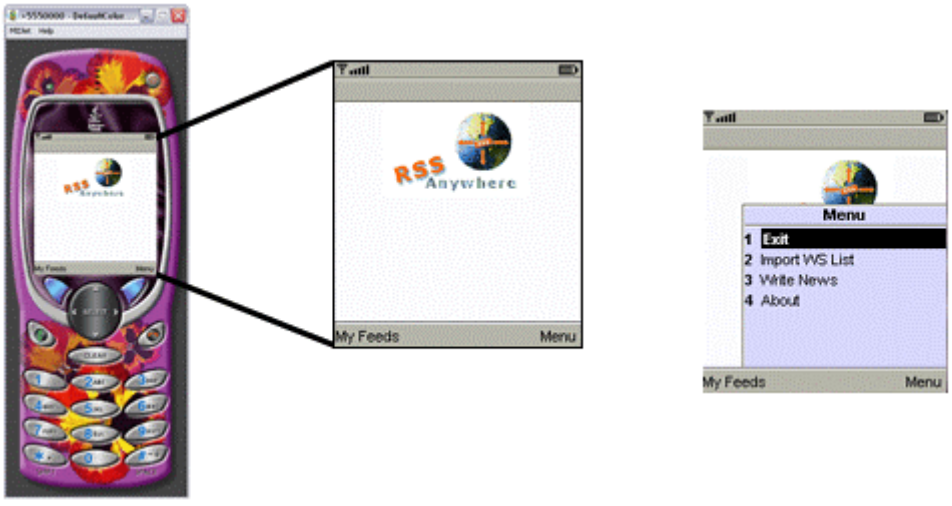
O início do código é similar aos outros dois métodos já vistos. Destacamos a semelhança com o segundo - `searchNews`, na parte referente do método `writeNews` encontrado no *Web Service*, por também receber um argumento de entrada. Só que neste momento, o parâmetro de entrada é apenas um elemento `FeedRSS` e não um vetor como no método anterior. Outra diferença quanto aos anteriores está no fato da resposta do *Web Service* deste ser apenas um texto puro (*string*) e não uma coleção de objetos FeedRSS, o que implica, que podemos mapear diretamente a resposta recebida numa variável do tipo String do método (linha 22). Após a chamada ao serviço através do comando `ht.call(client)` recebemos a resposta da operação e verificamos se ela foi bem sucedida ou não, retornando o resultado ao usuário da aplicação (linhas 23 a 26).

4.5 Interface Gráfica

Nessa Seção, mostramos algumas telas do aplicativo `RSSAnywhere` e as mensagens SOAP (Anexos A, B e C) trocadas com o *Web Services* pelos métodos `importWSList`, `searchNews` e `writeNews`, mostrando a facilidade para o usuário do dispositivo móvel, no caso celular, e a tecnologia por trás da interface.

4.5.1 RSSAnywhere

Lembramos que o emulador utilizado foi o WTK da Sun, com suas configurações padrões. Assim que iniciamos o aplicativo, uma tela é exibida (Figura 49.a), com detalhamento na tela central do celular.



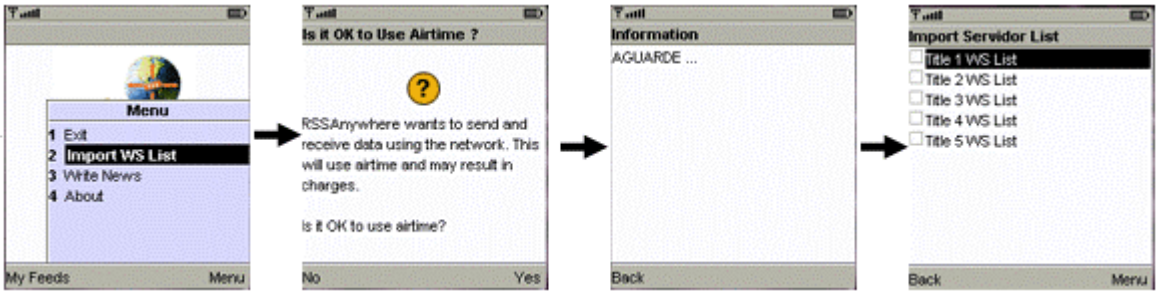
a) b)

Figura 49. a) tela inicial da aplicação, com detalhamento b) opções do menu principal

Pela Figura 49.a vemos o logo da aplicação, e dois botões: *MyFeeds* e *Menu*. A opção *MyFeeds*, quando acionada, lista as fontes gravadas no aplicativo do usuário, já a opção *Menu*, exibe mais quatro opções ao usuário como mostrado já em detalhe na Figura 49.b, sendo elas: *Exit*, *importWSList*, *writeNews* e *About*. A função *Exit*, fecha o aplicativo; a função *About* mostra informações sobre o aplicativo; já as funções *importWSList* e *writeNews* serão mencionadas a seguir.

4.5.2 importWSList

A opção *importWSList* no menu da aplicação tem como objetivo solicitar ao *Web Services* a lista mais recente de fornecedores disponíveis. A Figura 50 mostra todo esse processo na visão do usuário da aplicação cliente.



a) b) c) d)

Figura 50. Seqüência do processo do método importWSList

O processo começa quando o usuário escolhe a opção *importWSList* no menu principal - Figura 50.a. Neste ponto, como a aplicação procura acessar o protocolo de rede do celular (via http) uma tela de aviso é automaticamente exibida - Figura 50.b - com o título "*Is it OK to use Airtime?*" que seria algo como "Se o usuário está pronto para usar a rede?", exibindo duas opções: *Yes* (permite fazer a conexão com servidor) ou *No* (encerra o acesso e retorna a tela principal). Se a opção for positiva, aparece a tela seguinte ilustrada na Figura 50.c, que indica que a conexão foi estabelecida com o servidor, no caso o *Web Service*, e o dispositivo esta esperando uma resposta. Uma vez a resposta sendo retornada, a tela seguinte será exibida (Figura 50.d), listando os fornecedores disponíveis no servidor naquele momento (objetos FeedRSS).

As mensagens SOAP trocadas pelo cliente e o servidor, durante este processo, podem ser vistas no Anexo A. Neste Anexo na letra 'a', podemos ver a chamada ao método `importWSList` na mensagem enviada ao *Web Service*, e a resposta do mesmo ao cliente, sendo mostrado apenas uma parte da lista dos fornecedores retornados, no Anexo A letra 'b'.

Com a listagem na tela atual, o usuário pode ter acesso ao resumo (descrição) de um determinado fornecedor, bastando escolher um entre os listados, clicando no menu do lado direito e escolhendo a opção *See Description*, como visto na Figura 51.a. E se aquela fonte for interessante para o usuário, ele pode salvá-la em sua lista particular de Feeds, clicando no mesmo menu e agora escolhendo *Save* como opção, ver Figura 51.b, perceba que ao final dessa etapa o fornecedor arquivado não se encontra mais na lista dos fornecedores retornados pelo *Web Service*. O processo de buscar as informações contidas nesses fornecedores é detalhada no método *searchNews* adiante.

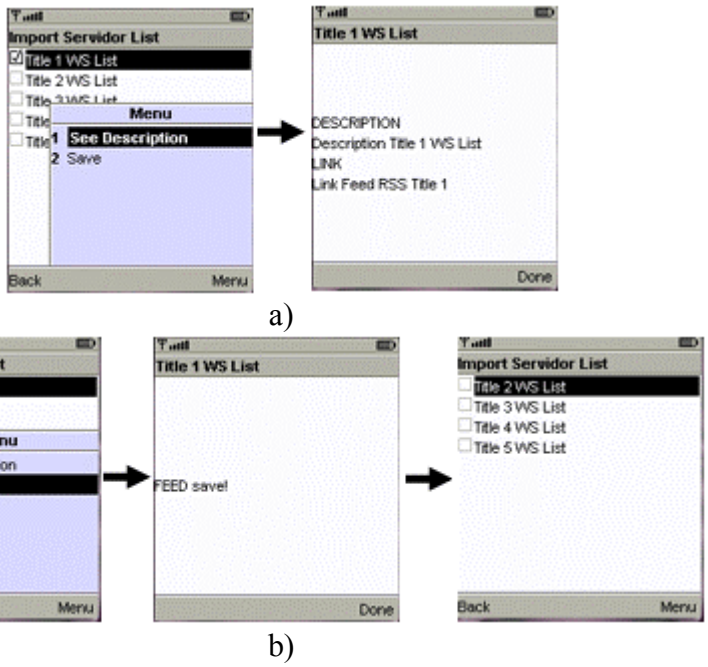


Figura 51. Telas da aplicação a) descrição do fornecedor b) salvar fornecedor

4.5.3 searchNews

Voltando para o menu principal da aplicação, a opção *MyFeeds* do lado esquerdo da tela, ao ser acionada mostra a listagem dos fornecedores gravadas do usuário, ver Figura 52.

a) b)

Figura 52. Tela MyFeeds a) lista gravada do usuário b) opções de gerenciamento

Note através da Figura 52.a, que a listagem atual já consta o *Feed Title 1 WS List*, que foi salva quando buscamos anteriormente a listagem do *Web Services*. Já a Figura 52.b mostra as demais opções disponíveis ao usuário, quando ele clica no item *Menu* do lado direito da tela, definidas por: *Back* - volta pra tela inicial principal do aplicativo; *Add* - adiciona um novo fornecedor à lista particular do usuário, neste caso o usuário terá que saber a url completa deste novo fornecedor; *Edit* - altera os dados de algum Feed, como título, descrição ou link do mesmo; *Delete* - apaga o fornecedor da lista; *Info* - visualiza a descrição (o resumo) do Feed selecionado.

Já o item *SearchNews* do lado esquerdo, é a opção que fará a requisição ao *Web Service* pela busca e processamento das informações contidas nos alimentadores escolhidos pelo usuário. A Figura 53 mostra todo o processo de busca na visão do usuário da aplicação cliente.

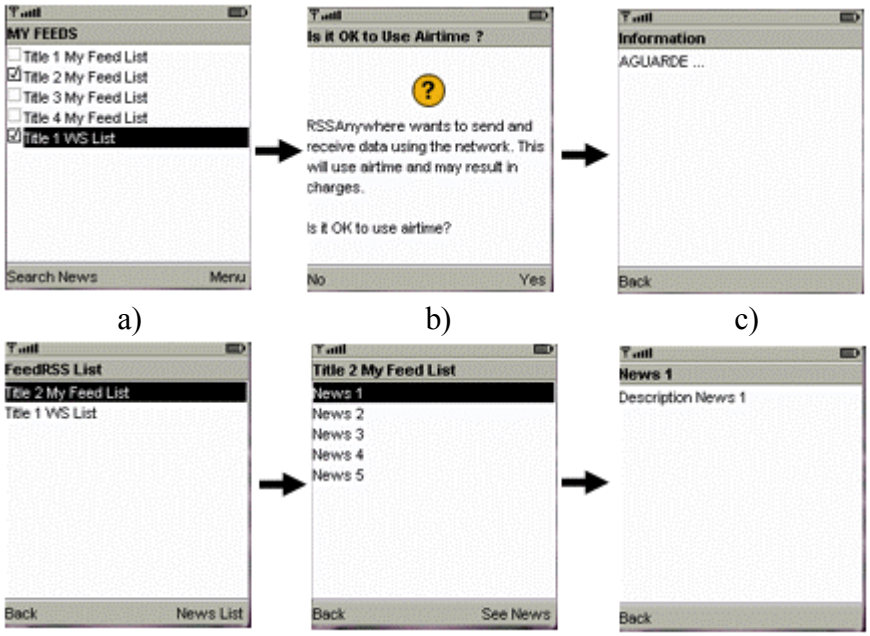


Figura 53. Seqüência do processo do método searchNews

Por exemplo, suponha que o usuário queira obter as informações contidas nas fontes *Title 2 My Feed List* e *Title 1 WS List*. Após selecioná-las, Figura 53.a, o usuário clica na opção *SearchNews*. Neste ponto, como no método *importWSList*, a aplicação está querendo acessar o protocolo de rede do celular (via http) e a tela de aviso novamente exibida. Se a escolha do usuário for positiva, aparece a tela seguinte exibida na Figura 53.c, que indica que a conexão foi estabelecida com o servidor e o dispositivo esta esperando uma resposta. Uma vez a resposta sendo retornada, a tela seguinte será a mostrada na Figura 53.d, que lista os mesmos *FeedRSS* escolhidos pelo usuário, com a diferença de ter em suas estruturas o conteúdo dos arquivos RSS encontrados na Internet. Uma vez o usuário tendo acesso a esta listagem, ele pode escolher uma fonte, Figura 53.d, e clicar na opção *NewsList* no menu direito da tela e as notícias referentes aquela fonte serão mostradas, como pode ser visto na Figura 53.e. Escolhendo a primeira notícia, *News 1*, por exemplo, o usuário terá acesso as informações que possivelmente estava procurando, sendo exemplificada na Figura 53.f.

As mensagens SOAP trocadas pelo cliente e o servidor durante o processo de busca das informações nos documentos RSS escolhidos, podem ser vistas no Anexo B.

Pelo Anexo B, podemos ver a chamada ao método *searchNews* na mensagem enviada ao servidor – letra ‘a’, e a resposta do mesmo, ao cliente, contendo as informações colhidas na busca dos alimentadores indicados pelo usuário – letra ‘b’.

4.5.4 writeNews

Ainda no menu principal da aplicação, a opção *writeNews* tem como objetivo atualizar um documento RSS particular com uma nova informação. A Figura 54 mostra essa funcionalidade na visão do usuário da aplicação cliente.

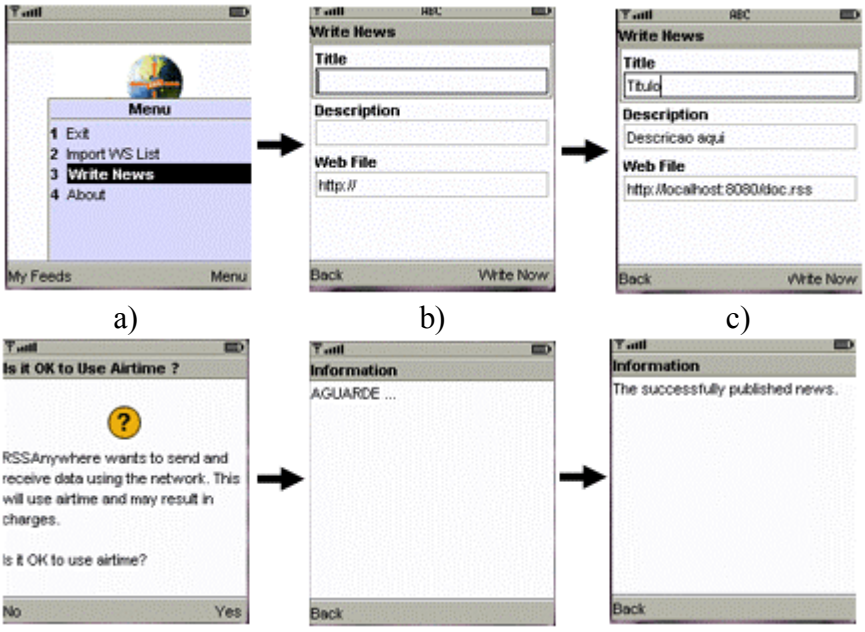


Figura 54. Seqüência do processo do método writeNews

O processo começa quando o usuário escolhe a opção *writeNews* no menu principal - Figura 54.a. Em seguida, ver Figura 54.b, é exibida um formulário solicitando ao usuário que insira o título e a descrição desejadas e por fim, o link (url) do documento RSS que será atualizado. No exemplo, os campos foram preenchidos respectivamente com "Título", "Descrição aqui" e "http://localhost:8080/doc.rss" (Figura 54.c). Ao final do preenchimento, o usuário clica na opção *Write Now* do lado direito da tela. Neste ponto, a tela de aviso padrão do aparelho é automaticamente exibida (Figura 54.d), optando pela ação positiva, a próxima tela será a indicada na Figura 54.e, mostrando que a conexão está sendo processada e o dispositivo está esperando uma resposta. Uma vez o retorno chegando ao cliente, a tela seguinte será a exibida na Figura 54.f, no caso, mostrando êxito na operação.

As mensagens SOAP trocadas pelo cliente e o servidor podem ser vistas no Anexo C. Através da letra ‘a’ do mesmo, podemos ver a chamada ao método *writeNews* na mensagem enviada ao servidor, e a resposta, letra ‘b’, do servidor ao cliente, contendo a resposta da operação realizada pelo *Web Service*.

Capítulo 5

Conclusões

O desenvolvimento de aplicações cada vez mais tem buscado a simplicidade e objetividade para seus usuários aliado às tecnologias e padrões que permitam uma interatividade maior entre aplicações diferentes. Diante desse cenário, os *Web Services* mostram- se como soluções viáveis para essa necessidade. O objetivo deste trabalho foi mostrar essa tecnologia sendo integrada na arquitetura J2ME através de um serviço útil ao usuário final da aplicação, buscando informações de arquivos RSS disponibilizados na Internet, com o auxílio de diversas ferramentas durante seu desenvolvimento.

Observa-se que a compreensão de todos os conceitos teóricos sobre *Web Services* estipulados ao longo do trabalho é de grande importância a fim de permitir que se entenda as principais características dessa tecnologia - padronização e interoperabilidade - e os passos necessários para criação de um serviço em qualquer linguagem de programação desejada.

Também através do estudo sobre a tecnologia J2ME, seus conceitos e características, demonstra-se que a união entre os dispositivos móveis e *Web Services*, é uma forma de aumentar "as funcionalidades" destes equipamentos, tornando-os úteis para satisfazer necessidades tecnológicas de usuários finais ou mesmo de empresas. Durante o decorrer do trabalho constatou-se, porém, que as aplicações *wireless* para dispositivos sem fio apresentam ainda muitos desafios aos profissionais da computação, pois existem ainda muitas limitações destas em relação às aplicações comumente realizadas para PC. Existem, inclusive, desafios no campo social, como o fato de que as atividades realizadas com um celular ainda são fundamentalmente diferentes das realizadas com um PC e a resistência dos usuários pelo fato de que a interação com o aplicativo no celular (como telas e teclado) é muito mais limitada que num PC. Além disso, há os desafios técnicos como a arquitetura limitada e a disponibilidade dos dados para transmissão, ainda dependente de local e operadora de telefonia celular.

Já a escolha do padrão RSS como elemento central da comunicação entre os módulos cliente-servidor, deve-se principalmente ao fato dele ser cada vez mais aceito e utilizado nos *web sites*, *blogs*, e outros sistemas encontrados na Internet. Seu formato muitas vezes simples, ou extensível, baseado em XML, lhe proporciona flexibilidade e poder para tratar com várias aplicações diferentes.

As ferramentas de desenvolvimento utilizadas mostraram-se adequadas e os módulos desenvolvidos neste trabalho comprovaram, através de seu funcionamento, terem atingido seus principais objetivos, permitindo ao usuário ter acesso às diversas informações disponíveis hoje na Internet através de arquivos RSS, e ainda podendo atualizar um arquivo RSS de propriedade dele na Internet. Nesse contexto, demonstrou-se um sistema com potencial de uso real na busca por

informações através de telefones celulares, uma vez que estar sempre atualizado em qualquer lugar que se esteja a qualquer hora do dia é um fator bastante relevante para pessoas que precisam estar sempre 'conectadas' com os últimos acontecimentos.

Outras ferramentas, como JBuilder ou Eclipse, poderiam ser usadas a fim de aumentar a velocidade de desenvolvimento dos arquivos e dos módulos escritos em Java, uma vez que esta característica já é comprovada em diversos meios de comunicação.

A tecnologia dos *Web Services* deve ser levada em conta quando se necessita de uma camada de fácil acesso entre aplicações *web*, mesmo que sejam desenvolvidas em plataformas diferentes. A união da arquitetura J2ME com os conceitos de *Web Services* se mostra uma solução de implementação viável, pois esta arquitetura possui tecnologias e APIs adicionais que permitem a integração de forma rápida e fácil. Unindo essas características e vantagens, em especial a interoperabilidade pelo uso de serviços *Web*, o projeto mostra-se preparado para a maioria das situações a que ele se destina.

5.1 Trabalhos Relacionados

Percebendo a necessidade de softwares específicos para aparelhos móveis, algumas empresas e equipes de desenvolvedores criaram programas para disponibilizar os arquivos RSS a qualquer usuário desses dispositivos com alguns pré-requisitos a depender de cada programa.

Os softwares estudados levaram em consideração as tecnologias similares aos do projeto realizado, assim como pela finalidade de disponibilizar um sistema/serviço para celular. Por este motivo, os programas escritos para PDAs não serão mencionados. Os aplicativos estudados são:

1. **MobileRSS**
2. **RSS News Reader**
3. **JRss Center**

1. MobileRSS

O MobileRSS [33] foi criado em meados de julho de 2003, esse serviço traz um formato diferente de buscar as informações nos arquivos RSS Ao invés de instalar um programa no PC ou no PDA como se vê mais comumente, o serviço MobileRSS disponibiliza as informações colhidas diretamente numa página *web* dentro do seu próprio *web site*. Essa página *web* atualiza a lista de RSS que o usuário cadastrou previamente, e lista as informações no *browser* do aparelho, incluindo aqui, aparelhos celulares.

Esse serviço é gratuito, porém é necessário um registro prévio no site do mesmo, para definir, ou editar, quais arquivos são de interesse do usuário. Também é necessário ter instalado no dispositivo móvel um *browser* para visualizar páginas na Internet, uma vez que a visualização de seus arquivos RSS prediletos é feito *on-line* diretamente no *site* do serviço. Este detalhe não é vislumbrado no aplicativo desenvolvido (RSSAnywhere), uma vez que o próprio programa é encarregado de exibir as informações transmitidas.

Um exemplo de visualização deste serviço num aparelho celular qualquer, disponibilizado na página do mesmo pode ser vislumbrado pela Figura 55. Nos nossos testes, não foi possível reproduzir algo semelhante, mesmo cadastrando-se no serviço citado, uma vez que não obtivemos informações suficientes pelo *site* para criar a página *web* com as informações dos documentos RSS que havíamos cadastrado.

Figura 55. MobileRSS - Tela de visualização de notícias num celular

Esse serviço possui como vantagem mais importante, o fato de centralizar a lista de RSS favoritos em um único lugar, uma área restrita dentro do *site* do serviço. É um serviço gratuito, mesmo depois do registro de um novo usuário e pode ser aplicável tanto para dispositivos celulares, como para PDAs. O ponto fraco deste serviço é ter que estar conectado à Internet até ler todas as informações desejadas; não ter como cadastrar um novo alimentador RSS através do dispositivo móvel, sendo possível apenas pelo *web site* do MobileRSS; não poder definir que arquivos RSS escolher para fazer a busca na Internet pelas informações desejadas, pois todos os sites cadastrados na área restrita do usuário são usados na busca das informações. Diferentemente, no serviço apresentado neste trabalho mostramos que o usuário pode cadastrar os *feeds* preferidos através do próprio aparelho, podendo escolher dentre eles, quais deseja obter as informações mais atuais nos arquivos RSS que eles oferecem.

2. RSS News Reader

O RSS News Reader [16] é um software de código aberto, disponível pela Nokia desde novembro de 2003, que tem a finalidade de buscar os documentos RSS através de telefones celulares.

Não existe um executável do programa, mas a Nokia disponibiliza todo o código para que o desenvolvedor crie um leitor de RSS no celular desejado, podendo alterar e melhorar o código da forma que quiser.

A diferença para o MobileRSS, além do fato desse ser um serviço e o RSS News Reader ser um aplicativo, é que toda busca e processamento dos arquivos RSS do serviço MobileRSS é feita pelo mesmo, deixando apenas a visualização das informações a cargo do aparelho. Já o RSS News Reader, como aplicativo criado pelo próprio usuário, deixa a cargo do dispositivo toda a busca e o processamento dos arquivos RSS, exigindo do aparelho seu processamento, memória e quaisquer bibliotecas adicionais. Comparando com o serviço criado neste trabalho, a principal diferença está no fato da busca e do processamento ficar a cargo do servidor *web*, restando ao sistema do aparelho celular realizar a solicitação e posterior exibição das informações colhidas pelo *Web Service*. Reduzindo, deste modo, o processamento e o armazenamento de informações adicionais para outras tarefas.

Outro fato importante, é que a busca das notícias através do RSS News Reader é realizada uma vez para cada documento RSS que se queira acessar, ou seja, se o usuário do aplicativo deseja obter informações de dois *sites* diferentes (dois arquivos RSS distintos), ele terá que

acessar, processar e ler as informações de um para, posteriormente, poder acessar, processar e ler as informações do outro. De forma diferente, no serviço RSSAnywhere criado neste trabalho, o usuário pode escolher quantos sites quiser para obter as notícias que tem interesse, e toda a busca e processamento é realizado apenas uma única vez pelo servidor, deixando o usuário livre para ler as informações colhidas no momento que quiser.

A forma de não ser um aplicativo "executável", faz com que o aplicativo RSS News Reader se torne mais um referência aos desenvolvedores J2ME do que aplicável aos usuários finais de celulares, que possuem pouco ou nenhum conhecimento de tais tecnologias e não saberiam "transformá-lo" no programa executável propriamente dito.

3. JRss Center

O JRss Center [13], da mesma forma que o RSS News Reader, não é um aplicativo pronto. O JRss Center foi criado num Trabalho de Conclusão de Curso de Mike Matsumoto, com o tema de "*Web Services* na Arquitetura J2EE" [13] cujo resultado final foi esse aplicativo.

O JRss Center é um software cliente-servidor desenvolvido na plataforma J2EE, que tem seu código fonte distribuído sob licença *BSD License [21]*. A parte servidora funciona um *Web Service* que busca e processa os arquivos RSS desejados pelo cliente. O Cliente acessa o serviço informando que *web sites* deseja obter por informações RSS e recebe as notícias encontradas. Tanto o servidor quanto o cliente funcionam em máquinas *desktop*. Sentimos falta de maiores detalhes de como implantar o software proposto por Matsumoto, e depois de algumas tentativas não obtivemos sucesso em executar o aplicativo; desta forma, não colhemos mais detalhes sobre seu funcionamento.

O enfoque no trabalho de Matsumoto, foi na tecnologia *Web Services* e na plataforma J2EE. Mostrando uma visão geral das tecnologias envolvidas como SOAP, UDDI, WSDL; vantagens e desvantagens da plataforma J2EE, aplicações multi-camadas, e ferramentas e padrões utilizados, como Eclipse, JBoss, Struts, Hibernate, etc. Mas um estudo mais detalhado com aspectos mais específicos quanto as tecnologias escolhidas, dentre elas o padrão RSS, por exemplo, ficou a desejar. Por esse motivo, mostramos no presente trabalho não só o aplicativo pronto e as ferramentas para sua confecção, mas também detalhes (história, formatos, características) das tecnologias que utilizamos para que isto se tornasse possível.

5.2 Contribuições

Neste trabalho, desenvolveu-se um software com uma arquitetura cliente-servidor voltado para aparelhos celulares, capaz de buscar as informações contidas em arquivos RSS na Internet e mostrar seu conteúdo de forma simples e objetiva no dispositivo, tornado-se a melhor opção para buscas de informações em aparelhos celulares.

Para isso, utilizou tecnologias de *Web Services* para inserir no servidor formas de padronização e interoperabilidade entre sistemas distintos, caso o servidor venha a ser acessado por um cliente diferente, como PC ou PDA, por exemplo. E o uso da plataforma J2ME para desenvolver o aplicativo no dispositivo móvel, focado em celular, detalhando a integração dessas duas tecnologias para o melhor funcionamento da aplicação.

Vale lembrar que nem o RSS News Reader nem o JRss Center são aplicativos executáveis. Suas disponibilizações estão nos seus códigos fontes encontrados publicamente, necessitando que o próprio usuário crie todo o ambiente e sistemas para que os mesmos possam a vir a funcionar. Outra diferença fundamental ao trabalho realizado nesta monografia está no fato de se estar trabalhando com um cliente sendo um dispositivo móvel, e não um *desktop* como no

caso do JRss Center. O que fez que detalhes de programação até então não abordados tivessem que ser planejados neste trabalho para um total sucesso na nossa integração.

Por fim, a realização deste trabalho trouxe muitas contribuições, principalmente no sentido dos assuntos estudados terem acrescentado conhecimentos; por ter demonstrado novas formas de integração de tecnologias para coletar e tratar dados; por se tornar uma fonte de referência na língua portuguesa, uma vez que documentos sobre este tema, em específico a integração realizada, são escassos e na maioria encontrados em línguas estrangeiras; assim como no exemplo de criação de sistemas que dispõem de interoperabilidade, tudo isso dentro da área da computação.

5.3 Trabalhos Futuros

Com o que foi apresentado, podemos afirmar que os objetivos deste trabalho foram alcançados em sua totalidade. Apesar disso, existem várias contribuições que ainda podem ser feitas.

Como sugestão de trabalhos futuros, teríamos a implementação no módulo cliente, dentre outras, novas funcionalidades tais quais:

- Salvar no aparelho as notícias pesquisadas na Internet;
- Escrever mais de uma notícia para ser atualizada;
- Ter um controle maior das fontes cadastradas, como a divisão em categorias.

No módulo servidor, podemos sugerir:

- Uma pesquisa mais detalhada dos fornecedores mais acessados na Internet para serem disponibilizados;
- Implementar uma logística na busca para encontrar as notícias desejadas de forma mais rápida;
- O uso de banco de dados para guardar as últimas notícias encontradas dos últimos fornecedores pesquisados, acelerando assim, a resposta do servidor.

Outros trabalhos que podem dar continuidade a esta monografia, podem ser:

- Comparativo entre *Web Services* em Java e .NET;
- Outras tecnologias para acesso a *Web Services* via dispositivos móveis;
- Atom, o novo formato de RSS;
- Extensões para RSS;
- Ofuscadores - comparações, reduções e impactos.

Bibliografia

[1] Beck, K. e Fowler, M. Planning Extreme Programming. Addison-Wesley Professional, 2000.
[2] Graham, S. *et al*. Building Web Services with Java™: Making Sense of XML, SOAP, WSDL, and UDDI. Sams Publishing, 2001.
[3] Hammersley, B. Content Syndication with RSS. O'Reilly, 2003.
[4] Hendricks, M. *et al*. Profissional Java Web Services. Alta Books, 2002.
[5] Irani, R. e Basha, S. J. Axis: Next Generation Java SOAP. Peer Information, 2002.
[6] Muchow, J. W., *Core J2ME Technology & MIDP*. Prentice Hall PTR, 2001.
[7] Newcomer, E. Understanding Web Services: XML, WSDL, SOAP, and UDDI. Addison-Wesley Professional, 2002.
[8] PIROUMAN, V. Wireless J2ME Platform Programming, 2002.
[9] TOPLEY, K., *J2ME in a Nutshell*, O'Reilly, 2002.
[10] Tull, Chris. WAP 2.0 Development. Que, 2002.
[11] Cezar, G. Mobilidade em alta velocidade busca padronização, http://computerworld.uol.com.br/AdPortalV5/adCmsDocumentShow.aspx?GUID=42893EEE-7F2E-4EFF-BC5F-480D2C232394&ChannelID=20, acessado em 28/05/2005.
[12] Lima, L. B. F. Comparativo entre trocas de mensagens J2ME e J2EE. Escola Politécnica de Pernambuco – curso de engenharia da computação, 2005.
[13] Matsumoto, M. S. Web Services na Arquitetura J2EE. Centro Universitário do Triângulo - Pró-Reitoria de ensino de graduação - curso de ciência da computação, 2004.
[14] McHugh, J. Low Bandwidth SOAP, 2003, http://webservices.xml.com/pub/a/ws/2003/08/19/ksoap.html, acessado em 22/06/2005.
[15] Neto, A. C. Web Services em Java com Axis - Teoria e Prática, www.guj.com.br/content/articles/webservices/ws.pdf, acessado em 21/01/2005.
[16] RSS News Reader, http://www.corej2me.com/DeveloperResources/Nokia/J2ME_RSS_News_Read_v1_0_en.pdf, acessado em 25/07/2005.
[17] SUN. Java 2 Plataform, Micro Edition (J2ME). [S.l.],[2003], http://java.sun.com/j2me, acesso em 20/06/2005.
[18] Agregadores - http://pt.wikipedia.org/wiki/Agregador, acessado em 15/05/2005.
[19] Apache Axis, http://ws.apache.org/axis, acessado em 21/01/2005.
[20] Apache Tomcat, http://jakarta.apache.org/tomcat/, acessado em 21/01/2005.
[21] BSD license, http://en.wikipedia.org/wiki/BSD_License, acessado em 28/07/2005.
[22] Channel Definition Format (CDF), http://www.w3.org/TR/NOTE-CDFsubmit.html, acessado em 20/05/2005.
[23] Code Conventions for the Java Programming Language, http://java.sun.com/docs/codeconv/, acessado em 11/05/2004.
[24] Dave Winer Info Site, http://blogs.law.harvard.edu/dave/, acessado em 22/05/2005.
[25] Document Object Model (DOM), http://www.w3.org/DOM, acessado em 22/06/2005.
[26] General Public License (GPL), www.gnu.org/copyleft/gpl.html/, acessado em 28/07/2005.

[27] Hands , www.hands.com.br, acessado em 15/05/2005.
[28] HyperText Markup Language (HTML) Home Page, http://www.w3.org/MarkUp/, acessado em 15/05/2005.
[29] JPluck , jpluck.sourceforge.net/, acessado em 25/06/2005.
[30] kSOAP, ksoap.objectweb.org/, acessado em 25/07/2005.
[31] kXML, kxml.objectweb.org/, acessado em 25/07/2005.
[32] Meta Content Framework Using XML (MCF), http://www.w3.org/TR/NOTE-MCF-XML-970624/, acessado em 20/05/2005.
[33] MobileRSS, http://www.mobilerss.net/, acessado em 25/07/2005.
[34] Overview of SGML Resources (SGML), http://www.w3.org/MarkUp/SGML/, acessado em 15/05/2005.
[35] Resource Description Framework (RDF), http://www.w3.org/RDF/, acessado em 20/05/2005.
[36] Retrologic. http://www.retrologic.com/, acessado em 26/07/2005.
[37] Scripting News, http://www.scripting.com/, acessado em 22/05/2005.
[38] Scripting News in XML, http://davenet.scripting.com/1997/12/15/scriptingNewsInXML, acessado em 22/05/2005.
[39] Simple API for XML (SAX), sax.sourceforge.net/, acessado em 22/06/2005.
[40] SOAP 1.1 Specification, http://www.w3.org/TR/SOAP, acessado em 13/07/2005.
[41] Steve Jobs Info Site, http://www.geocities.com/franktau/, acessado em 22/05/2005.
[42] Sun Java Wireless Toolkit, http://java.sun.com/products/j2mewtoolkit/, acessado em 10/02/2005.
[43] Syndic8, http://www.syndic8.com, acessado em 10/11/2005.
[44] Tim Bray Info Site, http://www.tbray.org/ongoing/, acessado em 22/05/2005.
[45] UDDI, http://www.uddi.org, acessado em 14/07/2005.
[46] WSDL 1.1 Specification, http://www.w3.org/TR/wsdl, acessado em 14/07/2005.
[47] XML: Padrão regido pelo W3C - World Wide Web Consortium, http://www.w3c.org, acessado em 15/05/2005.
[48] XML Namespaces, http://www.w3.org/TR/REC-xml-names/, acessado em 15/08/2005.

Apêndice A

Mensagens SOAP do Método importList

```
<SOAPENV:Envelope xmlns:n0="urn:RSSAnywhereService1"
      xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:SOAPENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAPENV:Body
      SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<importWSList xmlns="urn:RSSAnywhereService" id="o0" SOAPENC:root="1" />
</SOAPENV:Body>

</SOAPENV:Envelope>
```

a)

```
<soapenv:Envelope
            xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>
  <ns1:importWSListResponse
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns1="urn:RSSAnywhereService">
   <ns1:importWSListReturn href="#id0"/>
  </ns1:importWSListResponse>
  <multiRef id="id0" soapenc:root="0"
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xsi:type="ns2:Vector"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns2="http://xml.apache.org/xml-soap">
   <item href="#id1"/>
   <item href="#id2"/>
   <item href="#id3"/>
   <item href="#id4"/>
   <item href="#id5"/>
  </multiRef>
  <multiRef id="id1" soapenc:root="0"
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
```

```
        xsi:type="ns3:FeedRSS" xmlns:ns3="urn:RSSAnywhereService1"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <title xsi:type="xsd:string">
             Title 1 WS List</title>
    <description xsi:type="xsd:string">
             Description Title 1 WS List</description>
    <link xsi:type="xsd:string">
              http://localhost:8080/site1.rss</link>
    <vecNewsDescription xsi:type="ns4:Vector" xsi:nil="true"
             xmlns:ns4="http://xml.apache.org/xml-soap"/>
    <vecNewsTitle xsi:type="ns5:Vector" xsi:nil="true"
             xmlns:ns5="http://xml.apache.org/xml-soap"/>
  </multiRef>
  <multiRef id="id2" soapenc:root="0"
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xsi:type="ns6:FeedRSS" xmlns:ns6="urn:RSSAnywhereService1"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
    <title xsi:type="xsd:string">
             Title 2 WS List</title>
    <description xsi:type="xsd:string">
             Description Title 2 WS List</description>
    <link xsi:type="xsd:string">
              http://localhost:8080/site2.rss </link>
    <vecNewsDescription xsi:type="ns4:Vector" xsi:nil="true"
             xmlns:ns7="http://xml.apache.org/xml-soap"/>
    <vecNewsTitle xsi:type="ns5:Vector" xsi:nil="true"
             xmlns:ns8="http://xml.apache.org/xml-soap"/>
  </multiRef>
    ...
 </soapenv:Body>

</soapenv:Envelope>
```

b)

Apêndice B

Mensagens SOAP do Método searchList

```
<SOAP-ENV:Envelope xmlns:n0="urn:RSSAnywhereService1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAP-ENV:Body
      SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <searchNews xmlns="urn:RSSAnywhereService" id="o0" SOAP-ENC:root="1">
  <temp xmlns="" xsi:type="SOAP-ENC:Array"
                 SOAP-ENC:arrayType="xsd:anyType[2]">
     <item xsi:type="n0:FeedRSS">
     <title xsi:type="xsd:string">
            Title 2 My Feed List</title>
     <description xsi:type="xsd:string">
            Description Title 2 My Feed List</description>
     <link xsi:type="xsd:string">
            http://localhost:8080/site2.rss</link>
     <vecNewsTitle xsi:type="SOAP-ENC:Array"
            SOAP-ENC:arrayType="xsd:anyType[0]" />
     <vecNewsDescription xsi:type="SOAP-ENC:Array"
            SOAP-ENC:arrayType="xsd:anyType[0]" />
    </item>
    <item xsi:type="n0:FeedRSS">
     <title xsi:type="xsd:string">
            Title 1 WS List</title>
     <description xsi:type="xsd:string">
            Description Title 1 WS List</description>
     <link xsi:type="xsd:string">
            http://localhost:8080/site1.rss</link>
     <vecNewsTitle xsi:type="SOAP-ENC:Array"
            SOAP-ENC:arrayType="xsd:anyType[0]" />
     <vecNewsDescription xsi:type="SOAP-ENC:Array"
            SOAP-ENC:arrayType="xsd:anyType[0]" />
    </item>
  </temp>
  </searchNews>
```

```
</SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

a)

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>
  <ns1:searchNews
 soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:ns1="urn:RSSAnywhereService">
   <ns1:searchNews href="#id0"/>
  </ns1:searchNewsResponse>
  <multiRef id="id0" soapenc:root="0"
       soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xsi:type="ns2:Vector"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
       xmlns:ns2="http://xml.apache.org/xml-soap">
   <item href="#id1"/>
   <item href="#id2"/>
  </multiRef>
  <multiRef id="id1" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xsi:type="ns3:FeedRSS" xmlns:ns3="urn:RSSAnywhereService1"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <title xsi:type="xsd:string">
Title 2 WS List</title>
   <description xsi:type="xsd:string">
Description Title 2 WS List</description>
   <link xsi:type="xsd:string">
 http://localhost:8080/site2.rss</link>
    <vecNewsTitle xsi:type="ns6:Vector" xsi:nil="true"
xmlns:ns5="http://xml.apache.org/xml-soap"/>News 1
   <vecNewsTitle xsi:type="ns7:Vector" xsi:nil="true"
xmlns:ns5="http://xml.apache.org/xml-soap"/>News 2
    ...
   <vecNewsDescription xsi:type="ns4:Vector" xsi:nil="true"
xmlns:ns4="http://xml.apache.org/xml-soap"/>Description News 1
<vecNewsDescription xsi:type="ns5:Vector" xsi:nil="true"
xmlns:ns4="http://xml.apache.org/xml-soap"/>Description News 2
    ...
  </multiRef>
  <multiRef id="id2" soapenc:root="0"
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
       xsi:type="ns6:FeedRSS" xmlns:ns6="urn:RSSAnywhereService1"
       xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
   <title xsi:type="xsd:string">
Title 1 WS List</title>
   <description xsi:type="xsd:string">
Description Title 1 WS List</description>
   <link xsi:type="xsd:string">
 http://localhost:8080/site1.rss </link>
   <vecNewsDescription xsi:type="ns4:Vector" xsi:nil="true"
xmlns:ns7="http://xml.apache.org/xml-soap"/>
   <vecNewsTitle xsi:type="ns5:Vector" xsi:nil="true"
xmlns:ns8="http://xml.apache.org/xml-soap"/>
  </multiRef>
   ...
```

```
 </soapenv:Body>

</soapenv:Envelope>
```

b)

Apêndice C

Mensagens SOAP do Método writeNews

```
<SOAPENV:Envelope xmlns:n0="urn:RSSAnywhereService1"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:SOAPENC="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:SOAPENV="http://schemas.xmlsoap.org/soap/envelope/">

<SOAPENV:Body
  SOAPENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<writeNews xmlns="urn:RSSAnywhereService" id="o0" SOAPENC:root="1">
   <paramFeedRSS xmlns="" xsi:type="n0:FeedRSS">
    <title xsi:type="xsd:string">Defesa da Monografia</title>
    <description xsi:type="xsd:string">
            Hoje as 08:00 na Sala 3 do Bloco I na Escola Politécnica
            de Pernambuco - POLI. Compareçam!</description>
    <link xsi:type="xsd:string">
            http://localhost:8080/doc.rss </link>
    <vecNewsTitle xsi:null="true" />
    <vecNewsDescription xsi:null="true" />
   </paramFeedRSS>
  </writeNews>
 </SOAPENV:Body>
</SOAPENV:Envelope>
```

a)

```
<soapenv:Envelope
      xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<soapenv:Body>
 <ns1:writeNewsResponse
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="urn:RSSAnywhereService">
    <ns1:writeNewsReturn
            xsi:type="xsd:string">Sucesso</ns1:writeNewsReturn>
 </ns1:writeNewsResponse>
</soapenv:Body>
```

```
</soapenv:Envelope>
```

b)

Apêndice D

História e Versões do RSS

O Início

Em 1995, Ramanathan V. Guha desenvolveu uma linguagem chamado de *Meta Content Framework* (MCF) [32] algo como "Estrutura de Informações de Índice", tendo como base trabalhos na área de Inteligência Artificial, em sistemas de representação do conhecimento, com o objetivo de descrever objetos, seus atributos, e os relacionamentos entre eles.

Na época, a Apple financiava esse projeto de pesquisa experimental (MCF), que posteriormente veio a se chamar ProjectX, e, logo em seguida, HotSauce. Em 1996, cerca de cem sites na Internet já criavam arquivos MCF que se descreviam, e o HotSauce permitiu que qualquer usuário visualizasse essas representações de MCF em terceira dimensão (3D). "O projeto tornou-se popular, mas ainda experimental, e quando Steve Jobs [41] retornou ao cargo de CEO (*Chief Executive Officer*) da Apple em 1997, ele extinguiu várias atividades de pesquisa que existiam, sendo a Hotsauce uma delas, e Guha teve que deixar a Apple e ingressar na Netscape.", comenta Ben Hammersley [3].

Já na Netscape, Guha conheceu e começou a trabalhar com Tim Bray [44] um dos pioneiros na criação do XML e juntos transpuseram a linguagem MCF para o formato baseado nessa tecnologia (XML). Mais tarde, esse trabalho transformou-se no formato de "Estrutura de Descrição de Recursos", *Resource Description Framework* (RDF) [35], em inglês, e segundo o W3C, RDF é "uma linguagem de propósito geral para representar informações na Internet".

Surgimento das versões

Em 1997, a Microsoft, percebendo o sucesso do projeto HotSauce, uniu-se com outras companhias e juntas desenvolveram com base na descrição da MCF o formato *Channel Definition Format* (CDF) [22], que seria "Formato de Definição de Canais", submetido ao W3C no mesmo ano. Apesar de padrões diferentes, os objetivos eram os mesmos: disponibilizar sumários de informações freqüentemente atualizadas.

Percebendo a concorrência ao seu redor, a Netscape acelerou o lançamento de um portal de serviços chamado de *My Netscape Network* – "Minha Rede na Netscape", incluindo, pela primeira vez em um *web site*, um pequeno arquivo de resumo baseado no RDF, que chamou de *RDF Site Summary* (RSS), no ano de 1999.

Esse primeiro resumo RSS, permitiu que o portal da Netscape indicasse trechos e *links* de outros *sites*, e um usuário poderia então personalizar sua rede contendo as notícias de todos os *web sites* que o interessassem, tudo dentro da mesma página *My Netscape*. "Basicamente, era uma versão de tudo que o HotSauce e o CDF tinham se tornado, o que logo fez um enorme sucesso", exalta Ben Hammersley [3].

Este primeiro formato, nomeado de RSS 0.90, era inteiramente baseado no modelo de dados RDF, e algumas pessoas acharam tal formato muito complicado para criar os sumários de informações de seus *web sites*.

Dave Winer [24], fundador da Userland Software Inc e um dos primeiros usuários de RSS, notando também essa dificuldade em criar arquivos RSS no formato apresentado e com um experiência de já ter criado um formato particular para atualização de seu *blog*, conhecido por *Scriptnews* [37][38], resolveu criar um novo padrão para o RSS, mais simples que a versão 0.90, pois acreditava que a simplicidade daria aos usuários uma adaptação mais fácil e rápida para criarem esses arquivos. Esse novo formato tornou-se a versão 0.91, totalmente livre da versão RDF – notadamente mais complexa, e a chamou de *Really Simple Syndication* (RSS).

Pouco depois, o projeto *My Netscape* foi fechado, dando início a duas vertentes:

- A primeira, liderada por Rael Dornfest da O'Reilly, quis introduzir algumas características de tornar o padrão mais extensível, proporcionando a habilidade de possuir novas características, como modularização, no qual módulos adicionais pudessem ser utilizados no arquivo. Para isso, viu-se a necessidade de continuar com a base mais complexa, no padrão RDF;
- Já a segunda vertente, liderada por Dave Winer, temia que essa 'continuação' adicionasse um nível de complexidade muito alto para os usuários e acreditava que manter o RSS tão simples quanto possível, seria o melhor caminho.

Em dezembro de 2000, a versão RSS 1.0 foi apresentada por Dornfest. Enriquecida pelo uso de módulos, XML *namespaces*, e baseada no modelo de dados RDF. Duas semanas depois, Dave Winer publicou a versão RSS 0.92, como alternativa a versão 1.0, buscando a simplicidade já encontrada na versão 0.91. E, assim, permaneceram por cerca de dois anos: o padrão 0.92 como a especificação mais simples e o padrão 1.0 como sendo a mais complexa, contendo mais características, extensões e flexibilidade.

Após esse período, no ano de 2002, um novo formato surgiu, a versão 2.0, dando continuidade às versões 0.91 e 0.92. Porém, esee novo formato estava um pouco mais robusto que ambas as anteriores, mas ainda resistindo a alta complexidade que o modelo RDF da versão 1.0 oferecia. E, desde então, as versões 1.0 e 2.0 são os formatos mais atuais e recomendados para a criação de arquivos RSS.

Mais recentemente, um grupo de desenvolvedores discute uma versão definitiva aos formatos, que visa unir as melhores qualidades das versões existentes até o momento. Tal formato é conhecido por Atom, e não entraremos em detalhes sobre o mesmo uma vez que as discussões ainda estão no início e fogem do objetivo deste trabalho.

Versões

Nessa Seção são apresentadas as principais versões de RSS e as características de cada uma.

RSS 0.91

É a versão mais antiga e a que continua em uso até hoje. RSS 0.91 é uma combinação do formato 0.90 criado por um grupo de desenvolvedores da Netscape liderado por Dan Libby, com um outro formato de Dave Winer, chamado *Scriptnews*.

Apesar de nenhum dos formatos originais estar sendo usado de maneira significativa hoje em dia, o resultado dos dois, a versão 0.91, continua como uma das mais populares das versões existentes. Até novembro de 2005, último acesso a página *web* Syndic8.com [43] – um dos mais famosos repositórios de *feeds* conhecidos no mundo, com quase 450 mil alimentadores cadastrados; em torno de 14% dos *feeds* encontrados nesse *web site* usavam ainda o formato RSS 0.91. Apesar do uso cada vez menor das versões 0.9x, esses continuam sendo ainda ótimos formatos para começar a criar arquivos RSS.

A Figura 56 mostra a representação em árvore da estrutura do padrão RSS 0.91, retirada do livro *Content Syndication with RSS* [3].

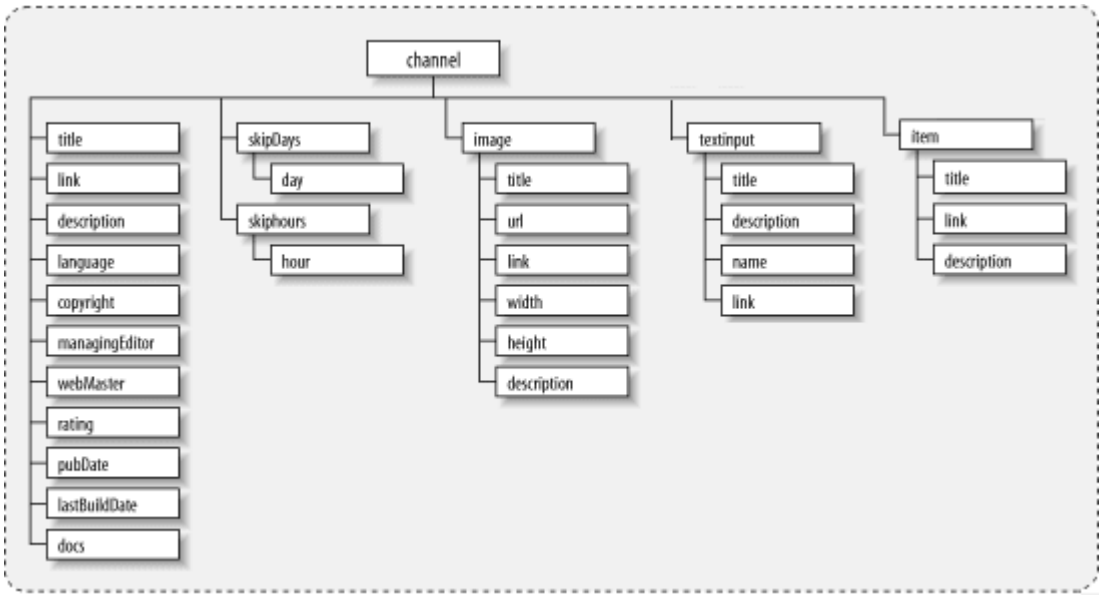


Figura 56. Representação em árvore do formato RSS 0.91

Como principais características dessa versão, podemos destacar [3]:

- Ser baseada em XML;
- Consiste em apenas um elemento *channel*, contendo no máximo 15 elementos *item;*
- Cada *item* possui um título, uma descrição e uma url;
- Metadados limitados, apenas usados no elemento *channel.*

RSS 0.92

É o padrão posterior ao 0.91 e teve como principais mudanças [3]:

- O elemento *channel*, poder ter infinitos elementos *item*;
- E cada *item* ter opção, a mais, de poder ser categorizada; poder ter uma versão em mídia (áudio ou vídeo) da informação contida no *item*.

Até novembro de 2005, último acesso a página *web* Syndic8.com [43], apenas cerca de 2% dos alimentadores cadastrados usavam esse formato, o que mostra que suas alterações à versão 0.91 não foram de grande impacto para comunidade de desenvolvedores em geral.

A representação em árvore da estrutura do padrão RSS 0.92, é bastante similar à versão 0.91, não sendo mencionada aqui.

RSS 2.0

Logo após o lançamento da versão RSS 2.0, Dave Winer declarou que não daria continuidade a criação de novas versões. Contudo, algumas pequenas atualizações foram lançadas para corrigir, melhorar ou esclarecer pontos específicos, "Mas para a criação de documentos RSS, a versão final a ser utilizada sempre seria a 2.0", afirma Hammersley [3].

Quando em novembro de 2005, acessamos a página *web* Syndic8.com pela última vez, a quantidade de alimentadores no formato 2.0 chegava perto dos 67% do total dos *feeds* cadastrados, concluindo-se que a simplicidade adotada por Dave Winer desde a versão 0.91, parece ter dado certo.

A Figura 57 mostra a representação em árvore da estrutura da versão 2.0, retirado do livro *Content Syndication with RSS* [3].

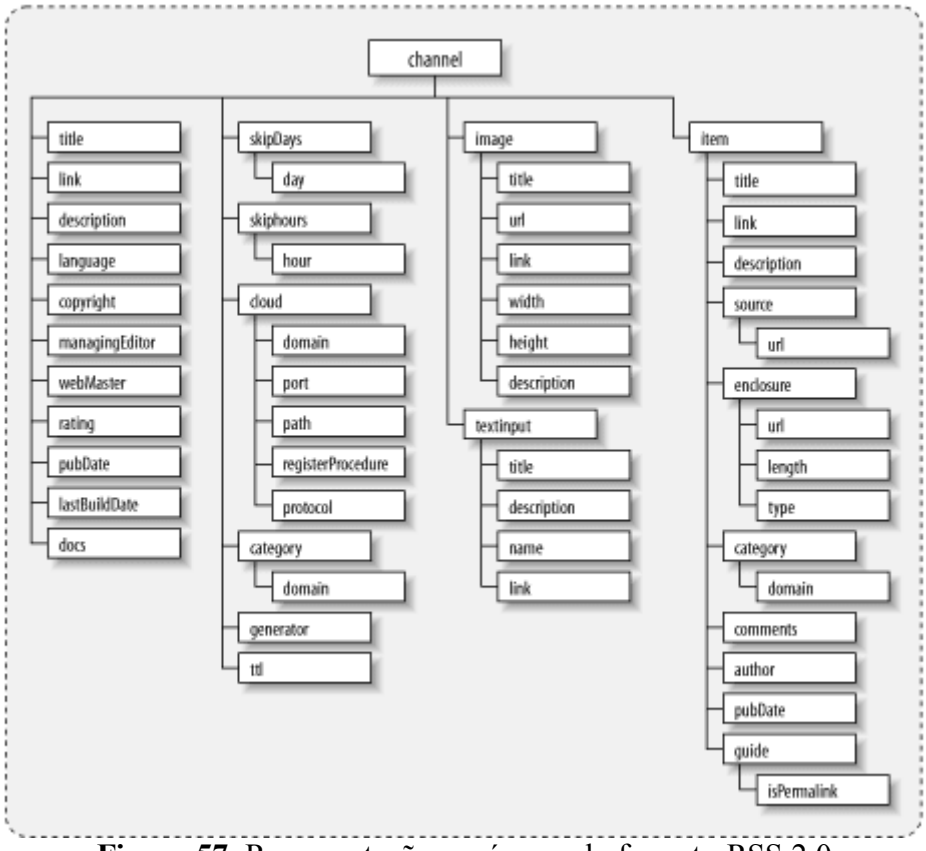


Figura 57. Representação em árvore do formato RSS 2.0

O resumo de suas características é listado a seguir, de acordo com [3]:

- Continua sendo baseado em XML, mas um pouco mais complexo do que as versões anteriores;

- É agora modularizado, oferecendo extensibilidade com um pouco de complexidade, através do uso de módulos baseados em *namespaces* XML;
- É o último das versões RSS simplificadas.

RSS 1.0

Baseado no formato RDF, *namespaces* XML e com suporte a módulos, essa última versão apresentada perde em simplicidade mas torna-se altamente extensível. Com a característica de modularidade inserida, existem, atualmente, mais de 14 conjuntos de elementos adicionais disponíveis fora da base da especificação 1.0 [3]. Dessa forma, temos uma listagem de objetos tão imensa e diversa que suprem as mais variadas possibilidades de sistemas no mundo real.

Analisando o comportamento dessa versão no *web site* Syndic8.com até novembro de 2005, cerca de 17% dos *feeds* cadastrados definiram esse formato de RSS, o padrão para disponibilizarem seus sumários de conteúdos. Dessa forma, percebemos que apesar das vantagens apresentadas por essa versão, a simplicidade e facilidade apresentada pelas versões anteriores são os maiores atrativos quando um *web site* quer criar um documento no formato RSS.

A Figura 58 mostra a representação em árvore da estrutura da versão 1.0, retirada do livro *Content Syndication with RSS* [3].

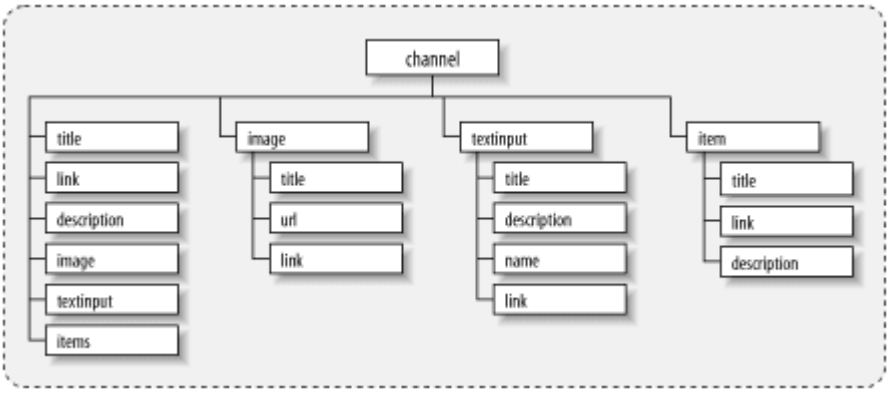


Figura 58. Representação em árvore do formato RSS 1.0

Ben Hammersley [3] destaca nessa versão:

- Ser baseada em XML, porém mais complexa do que as versões anteriores;
- Ser fundamentada em RDF, provendo riqueza de metadados;
- Ser modularizada, oferecendo uma grande diversidade de extensões.

A Figura 59 mostra o uso de módulos adicionais num pequeno trecho de código de um arquivo no formato RSS 1.0. Note que o módulo é adicionado através de *namespace*, que pode ser visto no início do documento pela inserção de um elemento *xmlns,* seguido do nome do módulo requerido e o *link* que indica a especificação do respectivo módulo, como mostrado nas linhas 3 a 7 da Figura 59. Para usar um módulo adicional, inserimos o código (*rdf*, módulo RDF, no exemplo) seguido de dois pontos (:), como visto na linha 11 no elemento em que queremos usá-lo.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <rdf:RDF
3   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
4   xmlns:dc="http://purl.org/dc/elements/1.1/"
5   xmlns:sy="http://purl.org/rss/1.0/modules/syndication/"
6   xmlns:co="http://purl.org/rss/1.0/modules/company/"
7   xmlns:ti="http://purl.org/rss/1.0/modules/textinput/"
8   xmlns="http://purl.org/rss/1.0/"
9  >
10 ...
11  <item rdf:about="http://c.moreover.com/click/here.pl?r123">
12   <title>XML: A Disruptive Technology</title>
13   <link>http://c.moreover.com/click/here.pl?r123</link>
14   <description>This the description of the article</description>
15   <dc:publisher>The O'Reilly Network</dc:publisher>
16   <dc:creator>Simon St.Laurent (mailto:simonstl@simonstl.com)</dc:creator>
17   <dc:rights>Copyright &#169; 2000 O'Reilly Associates, Inc.</dc:rights>
18   <dc:subject>XML</dc:subject>
19   <co:name>XML.com</co:name>
20  <co:market>NASDAQ</co:market>
21  <co:symbol>XML</co:symbol>
22 </item>
23 ...
24 </channel>
25 </rdf:RDF>
```

Figura 59. Trecho de código do formato RSS 1.0

Detalhes mais específicos sobre suas restrições e módulos adicionais, sobre cada formato mencionado, 0.91, 2.0, 1.0, podem ser encontradas nas suas respectivas especificações.