UNIVERSIDADE FEDERAL DE SANTA CATARINA

Um framework para Web Services através do Short Message Service.

BRUNO DE MEDEIROS LEDESMA

Florianópolis – SC

2007 – 1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Um framework para Web Services através do Short Message Service.

BRUNO DE MEDEIROS LEDESMA

Trabalho de conclusão de curso
apresentado como parte dos
requisitos para obtenção do grau
de Bacharel em Ciências da Computação

.

Florianópolis – SC
2007 – 1

BRUNO DE MEDEIROS LEDESMA

Um framework para Web Services através do Short Message Service.

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador: Dr. João Bosco Mangueira Sobral

Banca Examinadora

Fernando Augusto da Silva Cruz

João Bosco A. P. Filho

Lista de Figuras

Lista de Tabelas

Índice

Resumo

O presente trabalho é uma implementação de um framework cujo propósito é tratar recebimento e envio de mensagens SMS, onde o conteúdo e a lógica de negócio são solicitados à Web Services e respondidos para o cliente SMS. Para isso, foi criado um modelo abstrato de servidor, o qual foi estendido para implementação concreta de um servidor de Web Services e um servidor de mensagens SMS. Por fim, é descrita a implementação de um serviço, que atende aos usuários de uma rede de transporte público, para que eles consigam consultar os horários das linhas que passarão no ponto onde os mesmos estão localizados.

Palavras-chave: **SMS, mobilidade,web services, SOA, aplicações móveis.**

Abstract

The present work is a framework implementation which purpose is to handle Short Messages send and receiving actions and provide business logic and content with Web Services. For that was made an abstract server model that has been extended in a concrete implementation of a Web Services Server and a Short Messages Service Server. As conclusion, it is described a service implementation that attends public transport network users and give to them the power to query the bus-linhe schedule for that point.

Keywords: **SMS, mobility, web services, SOA, mobile applications.**

1. Introdução

Em maio de 2007 a Anatel divulgou que chegou a 105 milhões o número de aparelhos de telefones celulares no Brasil. Este dado nos da uma noção do alto nível de utilização do telefone celular no dia a dia da população brasileira. [1]

Este alto número de aparelhos celulares nos dá a perspectiva de uma inclusão digital acontecendo longe dos tradicionais computadores pessoais (PCs), e ela está intimamente relacionada a mobilidade e ao baixo custo desses aparelhos. Com isso o serviço SMS, o mais popular serviço de mensagens nos aparelhos celulares de todo o mundo, ganha o potencial de maior cliente de requisições de serviços no Brasil. Não só pelos números constatados, mas por ser uma das funcionalidades mais usadas no aparelho celular e de fácil aprendizado. O uso de navegadores WAP e aplicativos nos telefones celulares ainda são restritos a uma parcela reduzida da população que possui o conhecimento e o poder econômico para usá-los. Estes aplicativos ainda são limitados no que se refere a solucionar efetivamente problemas no cotidiano de seus usuários. O serviço de mensagens SMS pode ser usado por quase todos os modelos de celulares existentes e uma grande parcela dos seus usuários tem domínio e poderio econômico para utilizá-lo.

Atualmente, o SMS tem um alto índice de uso por consumidores, mas um baixo nível de utilização para soluções de negócio. Em outras palavras, enquanto bilhões de mensagens SMS são trocadas entre consumidores individuais todo mês, o seu uso como parte da estratégia de negócio das empresas é relativamente baixo. Apesar de outras mídias como rádio, internet e televisão estarem disponíveis para publicidade, o SMS ainda não é amplamente utilizado. Este serviço, segundo um relatório da empresa

Clickatell [WATERMEYER, 2003], tem as seguintes características: alto alcance, baixo custo, bom poder de retenção na memória dos seus usuários (ver tabela 1). Tais características fazem do SMS uma potencial ferramenta de marketing para as organizações.

Tabela 1 – Comparação entre mídias de comunicação padrão

Meio de comunicação	**Alcance**	**Custo**	**Retenção na memória**
Televisão	**Um dos maiores**	Muito Alto	Bom
Radio	Médio	Médio	Ruim
Internet (banners)	Alto	Médio	Em queda
E-mail	Alto	Baixo	Muito baixo
Telefone	Médio	Alto	Médio
Mídia móvel	Médio	Alto	Médio
Correio	Alto	Alto	Médio
Interação Pessoal	Baixo	Alto	Alto
SMS	Alto	Baixo	Alto

Fonte: WATERMEYER, 2003

1.1 Tema

Integração de sistemas e serviços é o maior desafio para todos os analistas e desenvolvedores da área tecnológica nos próximos anos. Esforços consideráveis estão sendo feitos pelas maiores organizações da área da tecnologia da informação para que arquiteturas susceptíveis a integração sejam concebidas. O maior desafio para esta concepção é a elaboração de arquiteturas e frameworks de alto nível integração e de baixo nível de acoplamento entre sistemas e serviços. Estas duas características são necessárias um baixo custo de manutenção dos projetos de software e para que eles possam evoluir de forma organizada e robusta.

Neste contexto, temos dois atores principais que possuem suas necessidades e que devem ser respeitados na concepção dos sistemas, o usuário e a equipe de desenvolvimento. O papel cujos riscos são maiores, logicamente, é de quem desenvolve. Este deve estar apto a lidar com sistemas com um nível de complexidade cada vez maior e com uma necessidade de integração crescente. No entanto, ele não pode esquecer que todo seu trabalho tem como objetivo final aplicações com a maior variedade de serviços para seus usuários. Estes serviços devem estar disponíveis nos momentos necessários a quem usa independentemente da sua localização geográfica.

Levando em consideração os dois atores e seus papéis, a aplicação do serviço de mensagens SMS (Short Message Service) para acesso a serviços e aplicações quaisquer, torna-se uma alternativa para acesso fácil a serviços de negócio de qualquer natureza, mas principalmente em consultas a serviços de informação. Logicamente esta aplicação não soluciona todo e qualquer problema de acessibilidade e usabilidade que um sistema e seus serviços podem apresentar. Contudo, ela cumpre com estes requisitos em uma variedade imensa de situações.

1.2 Delimitação do Tema

A proposta apresentada pelo presente trabalho é viabilizar a implementação de um framework para aplicações com finalidade de integrar Web Services e mensagens SMS. Esta integração consiste em projetar e escrever código na linguagem de programação Java para que estas mensagens gerem requisições à Web Services e sejam recebam respostas a partir do conteúdo provido pelo mesmo.

O trabalho não pretende aprofundar-se nas tecnologias envolvidas, mas sim dar uma breve explicação sobre elas e propor um modelo de utilização e integração das mesmas através do framework desenvolvido.

1.3 Objetivos Gerais

Para facilitar e padronizar o desenvolvimento de soluções SMS, este trabalho terá como objetivos gerais projetar e a implementar inicialmente um framework de acesso à Web Services via mensagens SMS.

1.3 Objetivos Específicos

Prover código Java que facilite sua extensão e modele as aplicações cujo propósito é enviar e tratar o recebimento de mensagens SMS para responder e atender requisições à Web Services.

Objetivos Principais:

Fazer um estudo de caso de uma aplicação que utilize este framework para uma solução de negócio.

Objetivos Secundários:

Analisar a proposta de solução de negócio e a viabilidade de implementação da mesma.

2. Revisão de Literatura

Este capítulo trata de definições utilizadas no trabalho, revisando a literatura a respeito das tecnologias utilizadas na construção do framework. Tais tecnologias abordam desde o Short Message Service, o protocolo SMPP e os protocolos utilizados na construção do framework.

2.1 Short Message Service

A sigla SMS significa Short Message Service e descreve um serviço sem fio que proporciona aos seus usuários a possibilidade de transmitir mensagens alfanuméricas entre usuários de telefones móveis e sistemas externos, como correio eletrônico, *paging* e sistemas de correio de voz[2]. O serviço SMS foi criado como parte do padrão GSM fase 1, e acredita-se que a primeira mensagem foi enviada em dezembro de 1992[3]. A mensagem, que pode ser formada de palavras, números ou uma combinação alfanumérica, possui limitação de comprimento de 160 caracteres quando são utilizados alfabetos latinos e de 70 caracteres quando são usados alfabetos não latinos, como Chinês e Árabe. O SMS provê um mecanismo para transmitir mensagens curtas para e de terminais sem fio. O serviço faz uso de uma Short Message Service Center (SMSC) que atua como elemento de repasse e armazenamento das mensagens. A rede sem fios realiza o transporte das mensagens entre a central SMSC e os terminais sem fio. Um diferencial entre o SMS e outros serviços de transmissão de mensagens de texto, como *paging*, é que os elementos que compõem o SMS são desenvolvidos com o objetivo de garantir a entrega das mensagens até o destino. Também, outra característica importante, é que o dispositivo móvel está apto a receber e enviar mensagens a qualquer hora, independente de estar ou não com uma chamada de voz ou uma transmissão de dados em progresso, e,

quando são identificadas falhas temporárias, a mensagem fica armazenada na rede até que o destino volte a estar disponível.

Em relação à economia de recursos de rede, o serviço SMS é caracterizado pelo envio de pacotes sem consumo de banda e pelas transferências de mensagens de pequena largura de banda, que são dois fatores que tem relativa importância devido ao grande número de aplicações possíveis. Inicialmente, as aplicações para o SMS estavam focadas no objetivo de permitir o envio de mensagens de propósito geral nos dois sentidos e por estarem disponíveis alguns serviços de notificação. Com a consolidação da tecnologia e das redes, novos serviços como integração com correio eletrônico, fax, *paging* e serviços de informação, como boletins de previsão do tempo, surgiram e ampliaram o número de aplicações disponíveis aos usuários de terminais móveis.

2.1.1 Elementos de rede e topologia

A estrutura de rede básica do serviço SMS e a topologia de seus componentes serão descritos a seguir, e ilustrados na figura 1, para demonstrar seu funcionamento.

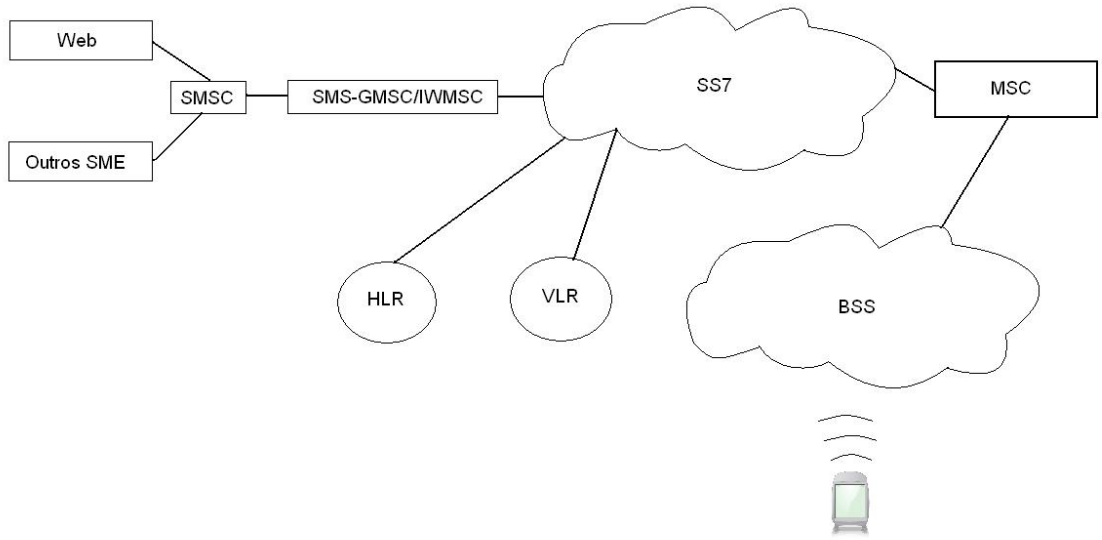

Figura 1 - Elementos da rede básica do serviço SMS

- **BSS – Base Station System**

Todas as funcionalidades relativas a transmissão e recepção via rádio são realizadas no Base Station System (BSS). O sistema BSS consiste de Base-Station Controlers (BSCs) e as Base-Transceiver Stations (BTSs). Ele é o principal responsável em transmitir tráfego de voz e dados entre as estações móveis

- **SME – Short Mesaging Entities**

A entidade SME pode enviar ou receber mensagens SMS, podendo estar localizada em uma rede fixa, uma estação móvel ou outro centro de serviços.

- **SMSC – Short Message Service Center**

SMSC é a central responsável pela retransmissão e armazenamento temporário de uma mensagem curta entre uma entidade SME e uma estação móvel. Dependendo do tipo de plataforma de SMSC e do protocolo utilizado pela aplicação emissora do SM temos a seguintes funcionalidades

1. Um prazo de armazenamento das mensagens de vários dias, com mecanismo inteligente de reenvio. Desta forma, diferentemente do Pager, a mensagem não é perdida se o assinante não estiver disponível na hora da primeira tentativa de envio. A mensagem sofrerá um atraso mas será encaminhada.
2. A possibilidade de priorizar as mensagens, estabelecendo grau de prioridade. Isto significa que uma mensagem mais urgente será entregue primeira.
3. A notificação de entrega para a fonte emissora.

- **HLR – Home Location Register**

O registro HLR é uma base de dados usada para armazenamento permanente e gerenciamento de assinaturas e serviços de perfil. Quando interogado pela central SMSC, o HLR provê a informação de roteamento para o asinante indicado. O registro HLR também informa a central SMSC, que não teve suceso anteriormente ao tentar despachar a mensagem, e que a estação móvel agora está acessível.

- **MSC – Mobile Switching Center**

Em português significa Central de Comutação e Controle. A MSC realiza o chaveamento do sistema e controla chamadas para e de outro telefone e sistema de dados.

- **MS-GMSC – SMS-Gateway Mobile Switching Center**

SMS-GMSC é um gateway capaz de receber uma mensagem de uma central SMS, interrogar um banco de dados HLR para obter informações de roteamento, e despachar a mensagem para a central de chaveamento (MSC) onde se encontra a estação móvel receptora.

- **SMS-IWMSC - SMS-Interworking MobileSwitching Center**

A central de interconexão SMS-IWMSC é capaz de receber uma mensagem da rede de dispositivos móveis e enviá-la para a central SMS apropriada.

- **VLR – Visitor Location Register**

O Visitor Location Register (VLR) pode ser definido como uma base de dados que contém informações temporárias sobre assinantes. Esta informação é necessária à central

de chaveamento (MSC) para o serviço a assinantes visitantes.

- **MS – Mobile Station**

Uma estação móvel (Mobile Station) é um terminal sem fio capaz de receber e originar mensagens curtas bem como chamadas de voz.

- **SS7 – Signaling System 7**

O sistema SS7 fornece a informação necessária para estabelecer e manter chamadas telefônicas envolvendo redes separadas, e não apenas na própria rede do telefone que originou a chamada.

2.1.2 O envio de Short Messages

Normalmente, uma mensagem demora cerca de quatro segundos para percorrer seu caminho desde o aparelho celular emissor até o receptor. O aparelho emissor estará no raio de alcance de uma estação base BTS (Base Transceiver Station) mais próxima do terminal emissor, que por sua vez encaminha a mensagem para uma estação de controle, que é denominada BSC(Base Station Controler).A BSC comunica-se com o comutador, designado pela sigla MSC (Móbile Switching Center) que por sua vez transfere a informação ao STP (Signalling Transfer Point), que se encarrega de transmitir ao SMSC(Short Message Service Center), que é o centro de processamento de mensagens da operadora.Neste ponto , a mensagem já está com seu processo realizado pela metade, e faz em seguida o percurso inversão até chegar à estação Base (BTS) mais próxima do receptor, que se encarrega de descarregar no terminal receptor, .

Todo este processo é executado quando lidamos com mensagens dentro de uma mesma operadora. No caso de mensagens entre operadoras diferentes, o SMSC fica encarregado de enviar a mensagem para um gateway que transmite os dados para a rede de destino que se encarrega de encaminhar a mensagem para o destinatário. Tudo leva quatro segundos e é exatamente o mesmo percurso de uma chamada de voz quando e estabelecida entre dois usuários de redes móveis.[4]

2.2 O Protocolo SMPP

O SMPP (Short Message Peer to Peer) é um protocolo aberto, desenvolvido para proporcionar uma interface para a comunicação de dados flexível, para a transferência de short messages entre um Short Message Center (SMSC), GSM USSD (Unstructured Supplementary Services Data) ou outro tipo qualquer de centro de mensagens, e uma aplicação SMS, como por exemplo, uma plataforma de Voice Mail, servidor de E-mail, Servidor Proxy WAP ou outra gateway de mensagens qualquer.

O protocolo SMPP utiliza o termo SMSC (Short Message Service Center) quando se refere à entidade servidora da conexão SMPP. No caso da entidade cliente da conexão SMPP, o nome adotado pelo protocolo é ESME (External Short Message Entity).

Funcionalidades do Protocolo SMPP

- Associar um tipo de serviço para cada mensagem.
- Transmissão de mensagens de uma ESME para um único ou múltiplos destinos via SMSC;
- Possibilitar que uma ESME receba mensagens de um terminal móvel através do SMSC;
- Envio de mensagens com confirmação de recebimento;
- Cancelamento ou reposição de mensagens;
- Consulta ao status de entrega de uma determinada mensagem;
- Agendamento de entrega de mensagens, selecionando a data e a hora de entrega;

- Seleção do modo de transmissão da mensagem, i.e. datagrama ou *store and forward;*
- Definição de tipo de codificação dos dados da mensagem;
- Definir um período de validade para a mensagem;
- Definição de prioridade de entrega para as mensagens;

A versao 5.0 do SMPP suporta o servido de short messaging para qualquer tecnologia de telephone móvel e tem aplicações especificas para tecnologias como:

- ANSI-136 (TDMA)
- iDEN
- IS-95 (CDMA)
- CDMA2000 (1xRTT & 3xRTT)
- GSM
- UMTS

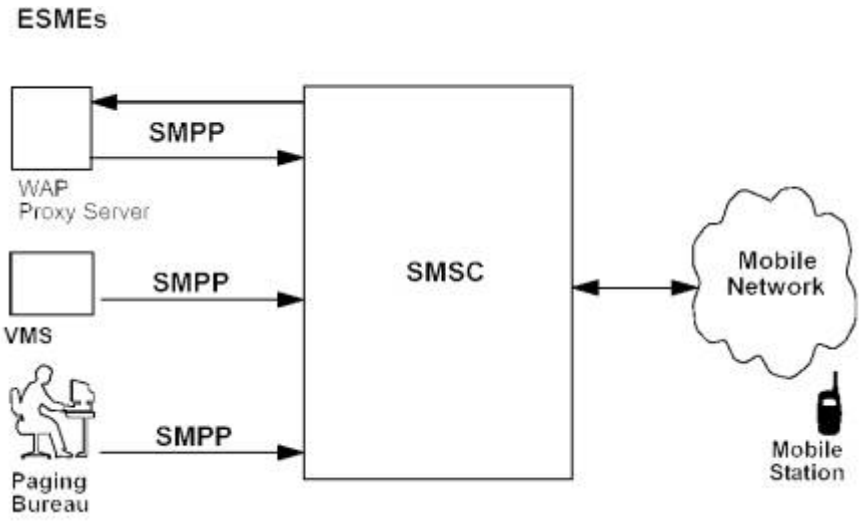

Figura 2 - Exemplos de aplicações que utilizam SMPP
Fonte: SMPP – Protocolos e aplicações, por João Bosco Silvino Júnior

Este protocolo está sendo utilizado recentemente para permitir a troca de SMS entre operadoras. Para este intuito foram desenvolvidas gateways para trabalhar com este protocolo, convertendo-o de uma tecnologia para outra (e.g. de TDMA para GSM). Este tipo de conversão se faz bastante necessário no cenário brasileiro dada a diversidade de tecnologias adotadas pelas operadoras.

Definições do protocolo SMPP

O protocolo SMPP pode trafegar sobre a pilha TCP/IP ou X.25. Em linhas gerais, a filosofia do protocolo SMPP consiste em abrir sessões específicas, permanentes, semi-permanentes ou dinâmicas, entre cada entidade. Por estas sessões são enviados os pacotes ou PDU (Protocol Data Unit), contendo as informações daquela operação SMPP específica. Fazendo uma analogia, a seção seria como uma rodovia por onde trafegam os caminhões, neste caso PDU's com as suas respectivas cargas, ou operações.

Da mesma forma que uma rodovia, as sessões SMPP podem ser unidirecionais ou bidirecionais.

O SMPP utiliza um sistema de troca de mensagens e confirmações de recebimento para garantir a confiabilidade das transações. O protocolo SMPP define:

- Um conjunto de operações para a troca de mensagens entre a ESME e o SMSC;
- Os dados que uma entidade pode trocar com a outra, durante uma operação SMPP.

Todas as operações de envio de mensagens SMPP devem ser seguidas de uma mensagem de resposta. A única exceção à esta regra é no caso da mensagem de ALERT_NOTIFICATION, que não requer resposta.

Os três grupos distintos de transações de mensagens SMPP são os seguintes:

a) mensagens enviadas a partir da ESME para o SMSC;

b) mensagens enviadas a partir do SMSC para a ESME;

c) Mensagens trocadas entre a ESME e o SMSC simultaneamente;

A figura 3 mostra os três tipos de seção possíveis dentro do protocolo SMPP. Note o sentido de envio das mensagens.

1. Conexão Transmitter;

2. Conexão Receiver;

3. Conexão Transceiver ;

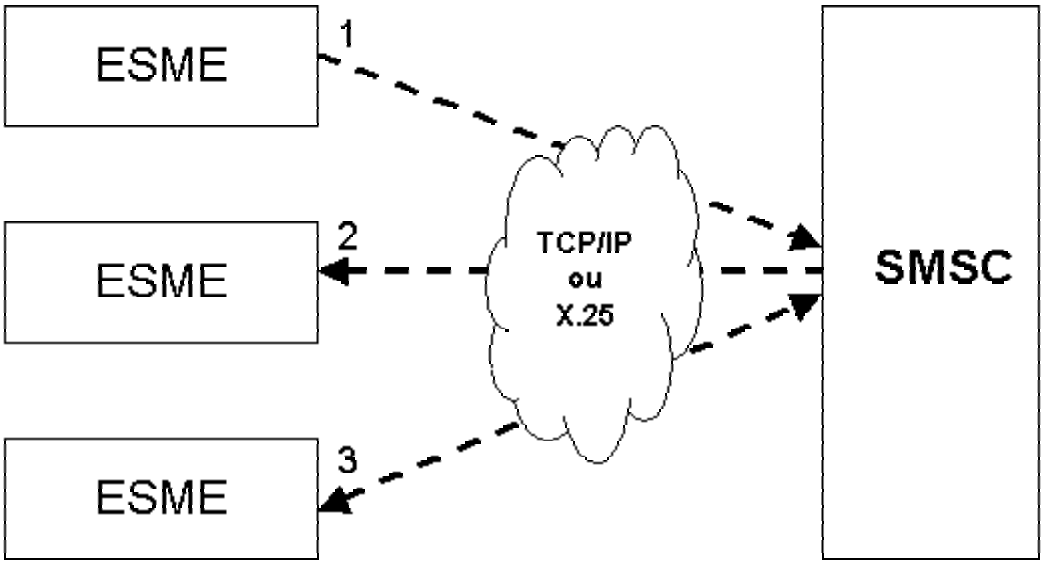

Figura 3 - Tipos de conexão SMPP

Fonte: SMPP – Protocolos e aplicações, por João Bosco Silvino Júnior

2.3 Aplicações típicas SMS

A variedade de aplicações via mensagens, particularmente SMS para qual o SMPP é aplicado, é quase infinita. Sistemas que suportam operações wireless e qualquer corporação que tenha como objetivo atingir usuários em grande escala e com um custo relativamente baixo, precisam deste tipo de aplicações. Alguns serviços de informações e aplicações que podem ser usados utilizando a tecnologia de Short Message Services serão descritos a seguir.

Serviços de *paging* tanto numéricos como alfanuméricos. Com um telefone com

suporte a SMS, não existe a necessidade de possuir um pager e um telefone. Por exemplo, uma aplicação que possibilita aos seus assinantes fazer consultas sobre preços e taxas de uma base dados ou da internet e mostra o resultado em seu telefone.

Serviços de informação. Por exemplo, um assinante requisite informações sobre restaurantes de uma região. O operador lista os restaurantes disponíveis e envia uma mensagem SMS para quem ligou.

Serviços baseados em localização. Isso inclui aplicações que utilizam dados da localização do aparelho celular obtida pela estação rádio base a partir da triangularização do sinal. Estas mensagens são enviadas para um ESME onde estes dados são utilizados para gerenciar serviços como a rastreamento de veículos roubados, chamadas de táxi e controle de logística.

Aplicações de telemetria. Por exemplo, um medidor embarcado que transmite mensagens SMS para uma companhia que faz o armazenamento e processamento da cobrança do uso do equipamento pelo seu usuário.

Aplicações de segurança como sistemas de alarme que podem usar o serviço SMS para acesso remoto e para alertar sobre eventuais problemas. Por exemplo, um pai recebe uma mensagem SMS da companhia de segurança para informar que sua filha chegou à sua casa e digitou seu código de acesso.

Bate-papos e jogos através de SMS. Usuários móveis podem interagir uns com os outros através de um servidor central (ESME) e usa desta interação para fundamentar jogos sem fio, encontros ou serviços de bate-papo através de SMS, similar aos conceitos de mensagem instantânea e de salas de bate-papo. Estes serviços já existem na forma da SMS-TV e do serviço SMS-Radio.

Notificações MMS. Em um Sistema de Mensagens Multimídia, uma mensagem

SMS pode informar a seus usuários que ele possui uma mensagem multimídia disponível no Centro de Mensagens Multimídia.

Serviços de transmissão para celulares. Aplicações cujo propósito é alertar os seus assinantes sobre o trafego de uma determinada localidade ou emergências quaisquer de uma região.

2.4 XML

A descrição e comunicação de Web Services utilizam da linguagem XML. XML (eXtensible Markup Language) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais.É um subtipo de SGML (acrônimo de Standard Generalized Markup Language, ou Linguagem Padronizada de Marcação Genérica) capaz de descrever diversos tipos de dados. Seu propósito principal é a facilidade de compartilhamento de informações através da Internet. Entre linguagens baseadas em XML incluem-se XHTML (formato para páginas Web), RDF, SMIL, MathML (formato para expressões matemáticas), NCL XBRL e SVG (formato gráfico vetorial).

Estimulado pela insatisfação com os formatos existentes (padronizados ou não), um grupo de empresas e organizações que se autodenominou World Wide Web Consortium (W3C) começou a trabalhar em meados da década de 1990 em uma linguagem de marcação que combinasse a flexibilidade da SGML com a simplicidade da HTML. O principio do projeto era criar uma linguagem que pudesse ser lida por software, e integrar-se com as demais linguagens. Sua filosofia seria incorporada por vários princípios importantes:

- Separação do conteúdo da formatação
- Simplicidade e Legibilidade, tanto para humanos quanto para computadores
- Possibilidade de criação de tags sem limitação
- Criação de arquivos para validação de estrutura (DTDs ou XML-SCHEMA)
- Interligação de bancos de dados distintos
- Concentração na estrutura da informação, e não na sua aparência

O XML é considerado um bom formato para a criação de documentos com

dados organizados de forma hierárquica, como se vê frequentemente em documentos de texto formatados, imagens vetoriais ou banco de dados.

2.5 Web Services

Web Service [5] é uma tecnologia projetada para ser uma solução utilizada na integração de sistemas e na comunicação entre diferentes aplicações. Com esta tecnologia é possível que novas aplicações possam interagir com aquelas que já existem e que sistemas desenvolvidos em diferentes plataformas sejam compatíveis. Os *Web Services* são componentes que permitem às aplicações enviar e receber dados em formato XML. Cada aplicação pode ter a sua própria "linguagem", que é traduzida para uma linguagem universal, o formato *XML*.

As empresas se beneficiam do uso dos *Web Services* porque eles podem trazer agilidade para os processos e eficiência na comunicação entre cadeias de produção ou de logística. Toda e qualquer comunicação e interação entre sistemas passa a ser dinâmica e principalmente segura, pois não há nenhuma intervenção humana.

O W3C e o OASIS são as instituições responsáveis pela padronização e regulamentação dos *Web Services*. Muitas empresas de grande renome no mundo todo da tecnologia da informação como IBM e Microsoft, apóiam o desenvolvimento deste padrão.

2.5.1 Tecnologias

Para construir-se um *Web service* temos como base os padrões XML e SOAP. O transporte dos dados é realizado, geralmente, via protocolo HTTP (o padrão não determina o protocolo de transporte). Os dados são transferidos em formato XML, encapsulados pelo protocolo SOAP.

2.5.2 Segurança

Muitas empresas tinham receio, no passado, prover funcionalidades na Internet devido ao medo de deixar seus dados expostos para seus concorrentes. Mas com advento dos *Web Services* elas podem publicar serviços de forma simples e clara, sem nenhum acoplamento com sua base de dados, garantindo a segurança das transações.

2.5.3 Integração de sistemas

É inegável que os *Web Services* corrigem um grande problema da informática: a integração de sistemas. Os *Web Services* permitem que a integração de sistemas seja realizada de maneira compreensível, reutilizável e padronizada. É uma tentativa de organizar um cenário de caos gerado pela grande variedade de diferentes aplicativos, fornecedores e plataformas.

2.5.4 O futuro dos Web Services

No cenário atual temos uma tendência de que as empresas listem seus *Web Services* em diretórios públicos (UDDI). Estes diretórios possibilitam o intercâmbio de serviços com outras empresas, instituições ou usuários comuns.

2.5.5 Tecnologias envolvidas

A representação e estruturação dos dados nas mensagens recebidas/enviadas é feita utilizando o XML (eXtensible Markup Language). As chamadas às operações, inclusive os parâmetros de entrada/saída, são codificadas no protocolo SOAP (Simple Object Access Protocol)(baseado em XML). Os serviços (operações, mensagens,

parâmetros, etc.) são descritos usando a linguagem WSDL (*WebServices Definition Language*). O processo de publicação/pesquisa/descoberta de *Web Services* utiliza o protocolo UDDI (*Universal Description, Discovery and Integration*).

2.6 SOAP

O SOAP (acrônimo do inglês *Simple Object Access Protocol*) é um protocolo para troca de mensagens entre programas de computador. SOAP é um dos protocolos utilizados na criação de *Web Services*. Na maioria das vezes os servidores SOAP são implementados utilizando-se servidores HTTP pré-existentes, embora isto não restrinja a utilização e o funcionamento do protocolo. As mensagens do protocolo SOAP são documentos XML que aderem a uma especificação fornecida pelo órgão W3C.

Segundo a definição do W3C[6]:

> SOAP Version 1.2 (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It uses XML technologies to define an extensible messaging framework providing a message construct that can be exchanged over a variety of underlying protocols. The framework has been designed to be independent of any particular programming model and other implementation specific semantics.

Porém, a flexibilidade desta técnica também traz problemas quando o fator desempenho é considerado. Por usar XML (linguagem totalmente baseada em texto), SOAP introduz um processamento adicional na construção e na interpretação das mensagens.

2.7 Frameworks

Um framework pode ser definido como um modelo conceitual de solução para uma determinada classe de problema. Quando se trata de frameworks orientados a objetos utilizados na construção de softwares, um framework pode ser definido como: um conjunto de classes que incorpora um projeto abstrato para a solução de problemas relacionados.[7]

O objetivo principal de um framework orientado a objeto é: ser estendível pela

criação de subclasse e parametrizado pela aceitação de objetos de outras classes [8]. Tais características estão presentes em padrões de projeto como o "Strategy" e o "Template Method". Padrões de projeto, no contexto da programação orientada a objetos, são descrições de objetos e classes para resolver um determinado problema [9]. Estes padrões são parte integrante dos frameworks, que utiliza normalmente diversos padrões no seu projeto.

3. O framework desenvolvido

Este capítulo trás as especificações técnicas do framework que foi desenvolvido, incluindo também um exemplo de aplicação para usuários de uma rede de transporte público.

3.1 Especificação

O framework provê classes escritas na linguagem de programação Java que através do mecanismo de herança e implementação de suas interfaces, facilitam o atendimento de requisições a serviços web feitos por mensagens SMS. No entanto, para que em uma re-fatoração futura seja possível o framework abstrai o tipo do cliente. O núcleo do mesmo é interfaceador das requisições aos serviços e tratará o processo de *parsing* da mensagem recebida, e de *parsing* da resposta a ser enviada.

A figura 4 ilustra como deverá funcionar este mecanismo.

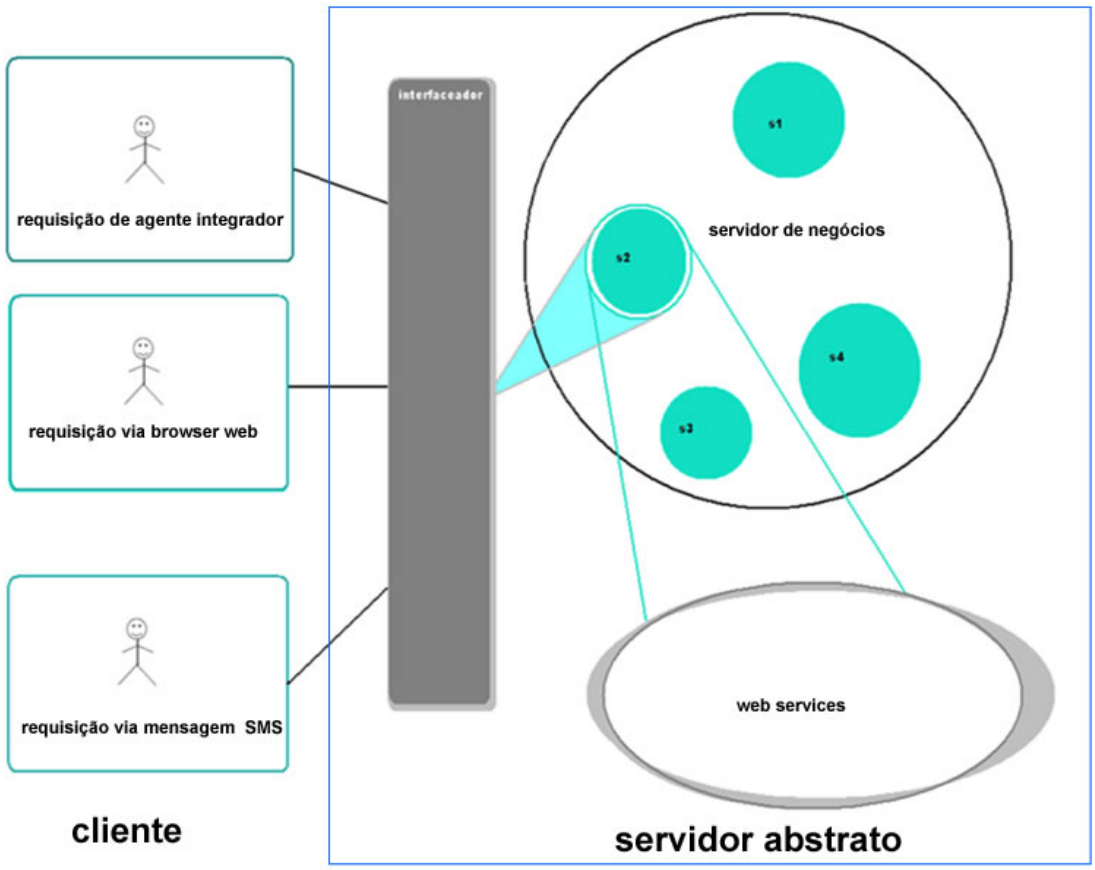

Figura 4 - Abstração da modelagem do framework

A figura 4 se divide em dois contextos. O lado do cliente que deve ser abstraído da implementação SMS proposta pelo framework e o lado de um servidor abstrato. O servidor abstrato trabalhará com o conceito de imagem única, abstraindo o cliente de seus componentes internos. Fará parte do framework uma camada que proverá as funcionalidades com a finalidade de requisitar os *Web Services* e responder corretamente.

3.2 Propósito

O framework tem como propósito inicial indicar uma especificação técnica para a criação de serviços de informação que utilizem mensagens SMS como cliente. Os Web Services serão uma camada isolada que proverá as informações com baixo acoplamento dos componentes de software e com alto grau independência da implementação.Para isto foram projetadas e desenvolvidas as classes e interfaces descritas neste trabalho, bem como foi criada uma aplicação para exemplificar a utilização do mesmo.

3.3 Projeto e classes

O framework desenvolvido tem o seu núcleo de utilização baseado em duas interfaces, como pode ser visto na figura 5. A interface *IServer* que descreve o comportamento padrão de um servidor para o framework. E a interface *IResponseHandler* que define o comportamento de uma classe que fará o *parsing* das mensagens e fará a conexão com o servidor de respostas para prover a resposta ao usuário do servidor.

3.3.1 Diagrama de Classes

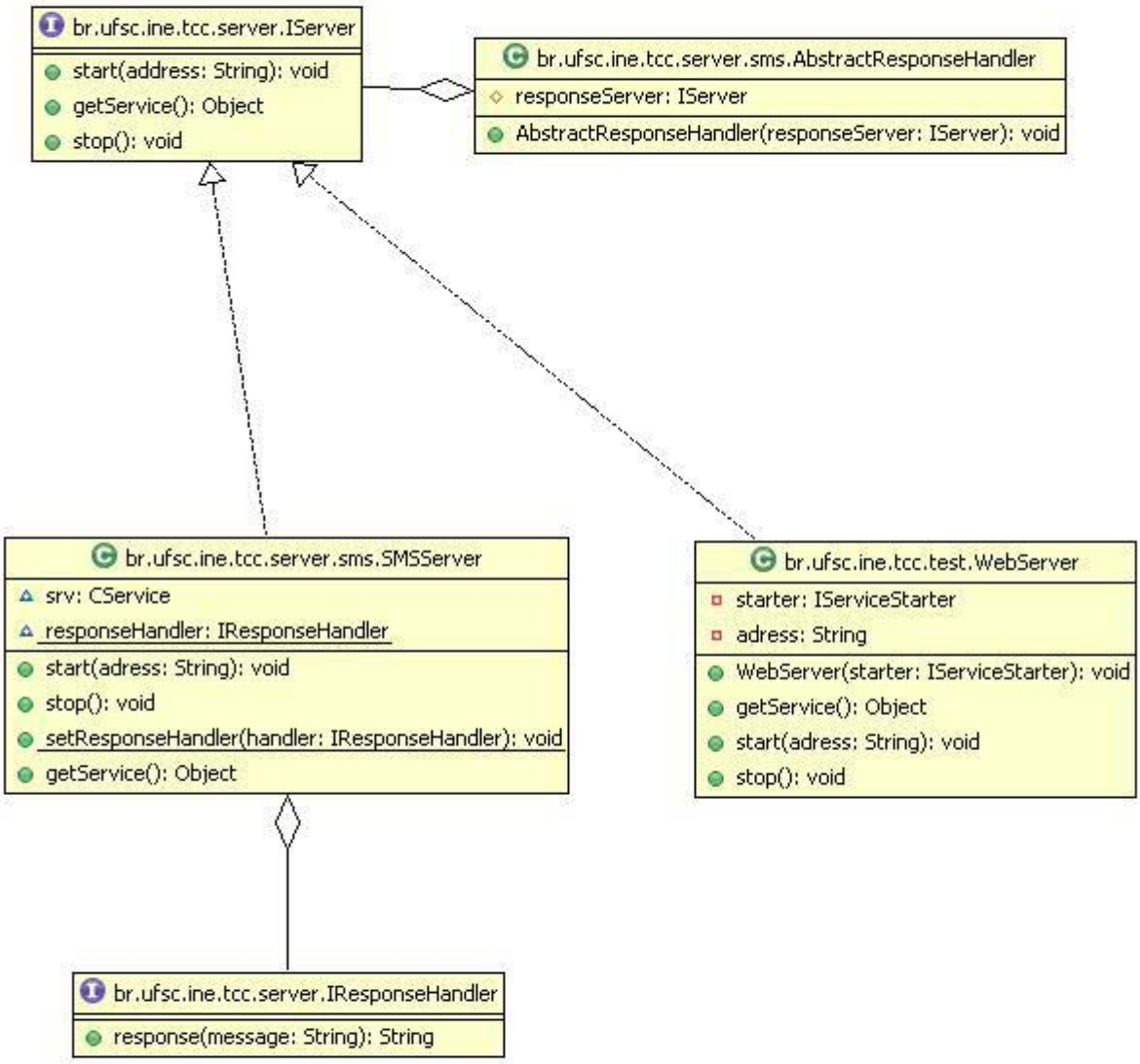

Figura 5 - Diagrama de classes do framework

3.4 Utilização

Para demonstrar a utilização do framework em uma aplicação real, teremos que utiliza-lo definindo algumas implementações:

1) A implementação do serviço de negócio a ser utilizado como exemplo.

2) A implementação do Servidor Web Service para responder as requisições do serviço;

4) A implementação do Servidor SMS para atender a necessidade de envio e recebimento de mensagens SMS;

5) Implementação da classe Java que tratará mensagens *(Handler)*, e executará o processo de *parsing* da mesma e invocação do serviço em utilização.

3.5 Aplicação proposta

Teremos como aplicação proposta para utilização do framework desenvolvido neste projeto a consulta dos horários previstos das linhas de ônibus em um determinado ponto de uma malha de transporte público. Para que tal aplicação seja possível tem-se como premissa:

- Cada ponto de ônibus deve ter um identificador único exposto ao seu usuário para que ele possa fazer a consulta;
- Uso de uma sintaxe simples que esteja explicita ao usuário ;
- Existe um limite de 160 caracteres na resposta do sistema;

No único serviço proposto por nossa aplicação teremos a seguinte sintaxe de consulta:

onibus <identificador do ponto>

Por exemplo, se o usuário se encontra no ponto de ônibus 234, e quer saber quais serão as linhas que passarão neste ponto e em que horário, ele enviara uma mensagem SMS para um determinado número, o número do servidor SMS, contendo o seguinte texto: "**onibus** 234".

O sistema conectará ao Web Service responsável pela resposta de tal consulta, e responderá algo como: 21h30 Linha 1; 21:45 Linha 2;

O número de linhas que o serviço irá listar é relativo, dependendo do número de caracteres da descrição da linha de ônibus, teremos mais ou menos estimativas passadas ao usuário. Com isso, devemos-nos preocupar com a construção da resposta para que ele efetivamente cumpra seu papel de informar o usuário tendo o tamanho de 160 caracteres.

3.5.1 Web Service

1. Interface do serviço

O classe LinhaOnibus é implementadora do serviço especificado pela Interface ILinhaOnibus. É única classe de serviço implementada neste trabalho e foi configurada para ser um Web Service através da JSR-181 [10]que utiliza o recurso *annotations* da linguagem Java, como explicado anteriormente neste trabalho. Para a construção da aplicação proposta foi implementado o método desta classe denominado getProximasLinhasByIdPonto que recebe como parâmetro o identificador da linha de ônibus e retorna a previsão dos horários dos próximos ônibus para este ponto. LinhaOnibus.java tem um atributo do tipo LinhaOnibusDao, que implementa o padrão de projetos DAO, responsável pela coleta dos dados da nossa aplicação.

2. O padrão Data Acess Object

O padrão de projeto DAO ou Data Access Object, tem o intuito de separar o código que depende de características especificas dos recursos de acesso aos dados, do código que programa as regras de negócio. Isso dificulta a manutenção da aplicação quando vamos alterar ou adicionar funcionalidades relacionadas ao modelo de dados da sua aplicação. O padrão de projeto Data Access Object :

- Separa a interface cliente dos recursos de dados dos mecanismos de acesso aos dados;
- Adapta a API de acesso a recursos de dados a uma interface cliente genérica;

O padrão DAO permite o que o acesso aos dados seja alterado independentemente do

código que usa os dados.

3. Persistência dos dados

Para a camada de acesso aos dados utilizando o padrão de projeto DAO, utilizou-se o banco de dados orientado a objeto db4o.

O db4o, é um banco de dados orientado a objetos que trabalha com o conceito de persistência via linguagem de programação.Isso quer dizer que o desenvolvedor deve conhecer simplesmente a linguagem de programação de sua preferência, sem a necessidade do aprendizado de uma linguagem nativa do banco de dados.Ele possui implementações para diversas linguagens, como Java e .Net.

Este banco de dados tem ampla aceitação no mercado corporativo. Possui usuários e clientes de cerca 170 países, da Albânia até o Zimbábue, e de empresas líderes de mercado em seus segmentos como a Boeing, Bosch, Intel, entre outros.

A motivação do seu uso no presente trabalho foi a simplicidade na programação da camada de persistência dos objetos do domínio de negócio, e o baixo grau de complexidade das consultas que a aplicação desenvolvida requeria. Para consultas mais complexas o db4o perde um pouco uma de suas características mais marcantes, que é a produtividade de desenvolvimento.

4. Servidor Web Services

Critérios de escolha

Na escolha de uma implementação Java para Web Services, o presente trabalho de deparou em uma escolha entre a implementação da “Apache Software Foundation”, o

Apache Axis e a implementação da "The Codehaus - Opensource Software Community" o XFire.

O principal critério utilizado para escolha da implementação foi a facilidade de criação de um Web Service a partir de uma classe Java qualquer. Também foi levado em conta a documentação e o nível de atividade das comunidades de usuários dos respectivos projetos. Diante disso, a primeira premissa do estudo foi que a implementação da JSR 181 seria pré-requisito no processo de escolha. Ambas as implementações dão suporte a JSR 181, que utiliza de um recurso contido na linguagem Java na versão 5, as anotations, para a criação de Web Services. Porém, a implementação do XFire é mais madura e possui melhor documentação que a do Apache Axis, que só implementa em sua versão 1.2 (Axis2), uma das mais recentes dentro de seu histórico. Um comparativo na documentação dos projetos e no suporte à duvidas dentro de fóruns e artigos de sites especializados na tecnologia deixou a convicção de que o XFire traria maior produtividade e rapidez para a implementação de Web Services em Java.

Desenvolvimento

Para explicar o desenvolvimento do servidor Web Services utilizado como exemplo de utilização do framework, devemos entender primeiro como funciona a API XFire. Segue abaixo a definição das principais classes deste framework.

XFireFactory

Esta classe que segue o padrão de projeto Factory, fabrica instancias do XFire, classe que concentra quase todas as funcionalidades do XFire.

ServiceFactory

Esta classe que segue o padrão de projeto Factory, fabrica instancias dos serviços do XFire, ou seja Web Services.

Service

Representa um serviço do XFire.

Criando um serviço a partir de uma classe Java

No exemplo abaixo, o serviço estará modelado como um objeto Java simples (pertencente à classe YourService) que será mapeado para um Web Service.

```
XFire xfire = XFireFactory.newInstance().getXFire();
ServiceFactory factory = new ObjectServiceFactory(xfire.getTransportManager(), null);
Service service = factory.create(YourService.class);
```

Isto vai criar um documento SOAP 1.1 para sua Classe. Se esta classe possui tipos complexos(ex: que não sejam Strings, Inteiros ou boleanos), o XFire vai automaticamente tentar serializar eles para você. Depois de criado, o serviço deve ser registrado:

```
xfire.getServiceRegistry().register(service);
```

Porém está forma de implementação fará com que o desenvolvedor da aplicação tenha que configurar alguns arquivos XML e fazer configurações no servidor HTTP a ser utilizado. Para abstrair problemas como este usa-se a implementação da JSR-181 que o XFire provê. Para diminuir a complexidade utiliza-se o recurso de inicializar um servidor Http chamado Jetty internamente a aplicação.

Primeiramente inicializamos as mesmas entidades do exemplo anterior porém agora instanciamos uma AnnotationServiceFactory. Está classe criará Web Services a partir de uma classe Java que esteja "anotada" como tal.

```
XFire xfire = XFireFactory.newInstance().getXFire();
AnnotationServiceFactory factory = new
AnnotationServiceFactory(xfire.getTransportManager());
Service service = factory.create(YourService.class);
xfire.getServiceRegistry().register(service);
```

A classe Java que sera transformada em Web Service deve possuir a anotação @WebService para que o XFire a reconheça como uma classe que está apta a ser transformada seguindo a JSR-181. Abaixo segue exemplo de tal anotação

```
@WebService(endpointInterface = "br.ufsc.ine.tcc.services.ILinhaOnibus")
public class LinhaOnibus implements ILinhaOnibus
```

Para inicializar o Jetty basta simplesmente instanciar um objeto da classe XFireHttpServer e invocar o método start() do mesmo. Com isso tem-se toda a infra-estrutura de código Java necessária para a instanciação de um servidor Web que responderá as requisições à Web Services.

3.5.2 Servidor SMS

A implementação do servidor responsável pelo envio e recebimento de mensagens SMS foi feita com a utilização de uma biblioteca que permite o estas funcionalidades através de um modem GSM ou um telefone celular conectado a estação de desenvolvimento.

1. Critérios de escolha

Durante o processo de pesquisa da ferramenta que auxiliaria no envio e recebimento de mensagens SMS tivemos como critério de escolha:

- Abstração da complexidade das tecnologias envolvidas;

- Baixa curva de aprendizado e baixo custo de implementação;

Seguindo esta linha, a pesquisa teve como candidatos finais a biblioteca de código aberto SMSLib e a implementação do protocolo SMPP para Java, OpenSMPP. No entanto, a segunda teve como característica eliminatória a necessidade de acesso a um SMSC. Tal requisito necessita a contratação de um serviço de acesso e utilização de um SMSC de uma operadora de telefonia.

Sendo assim, o presente trabalho escolheu biblioteca SMSLib , mas vale ressaltar a necessidade da utilização de uma implementação utilizando o protocolo SMPP para aplicações de grande porte. A biblioteca SMSLib permite o envio de cerca de 7 mensagens por minuto. Tal taxa de envio seria insatisfatória para a criação de aplicações de grande porte. O protocolo SMPP e o uso da infraestrutura disponível a partir de um SMSC atendem a demanda de uma taxa de envio de mensagens maior.

2. SMSLib

SMSLib é uma biblioteca que permite ao desenvolvedor criar aplicações que enviem e recebam mensagens SMS via um modem GSM. Você pode usar a SMSLib com um modem GSM dedicado ou com um telefone celular suportado por ela.

Características do SMSLib.

SMSLib trabalha com o protocolo PDU. SMSLib para Java (v2.1.o ou mais recente) também suporta protocolos TEXT, com o alfabeto GSM/HEX.

- Suporta o envio de mensagens WAP PUSH SI
- Suporta o envio e recebimento de mensagens 7-bit, 8-bit ou UCS2.
- Suporta mensagens multi-part
- Provê informações do modem GSM ou telefone celular: fabricante, modelo, nível de bateria e de sinal etc.

A biblioteca SMSLib comunica com um telefone móvel ou modem GSM através da porta serial. Ela usa um conjunto de comandos 3GPPAT, que é similar ao seu ancestral o Hayes AT, para controlar o modem. Estes comandos são enviados e recebidos através da porta serial.

Para a utilização da bilblioteca tem-se como pre-requisito a conexao via porta serial do seu modem GSM com o computador que possuirá o código a ser executado. Estes modems podem estar conectdos via cabos seriais. Mas alguns podem ser conectar via *Bluethooth*, IrDA ou cabos USB. No entanto, estas camadas de conexão devem

estar mapeadas para portas seriais através de *drivers* do dispositivo instalados no computador que excuta o código da aplicação.

4. Conclusões

O trabalho propôs e implementou uma alternativa de integração de duas tecnologias que suprem algumas necessidades no contexto de desenvolvimento de sistemas nos dias de hoje. O SMS que tem uma larga base de usuários e os *Web Services* tem como propósito facilitar a interoperabilidade entre sistemas distintos com baixo nível de acoplamento.

Verificou-se também a viabilidade de utilização destas tecnologias na criação de soluções de negócio, principalmente para servidores de informação. O conceito de arquiteturas orientadas a serviço (SOA) regeu o desenvolvimento da aplicação e da construção do framework, facilitando assim a convergência na utilização de serviços de diferentes tipos de rede.

A viabilidade comercial do uso de tais tecnologias e de tal framework para soluções de negócio ainda depende de um estudo mais detalhado dos custos de implantação deste modelo em um sistema de grande porte. Para viabilização de tais soluções, alguns fatores como acordos entre empresas e operadoras de telefonia e cooperação entre grupos de empresas que necessitam de tais soluções, são relevantes e necessitam ser levados em conta. Porém, mostrou-se que tecnicamente existe mais do que uma alternativa de implementação de tais soluções de negócio. Cabe ao contexto de utilização da solução, escolher as estratégias de implementação que suprem a sua necessidade.

5. Trabalhos Futuros

O trabalho desenvolvido poderá ser lapidado e melhorado em muitos aspectos. O tratamento de exceções, para mensagens digitadas incorretamente pode ser um desses melhoramentos, bem como a implementação de algum mecanismo de segurança baseado em um módulo de autenticação. Esse módulo pode ser a extensão de algum framework de autenticação já desenvolvido para este propósito.

A especificação de outras classes de aplicação do framework e a integração com o protocolo SMPP também é uma alternativa de extensão deste trabalho. Ao especificar outras classes de aplicação e implementar outras alternativas de recebimento de mensagens SMS, poderemos efetivamente consolidar este framework como uma alternativa viável para criação de servidores de informação SMS.

6. Anexos

Anexo 1 – Artigo

Um framework para Web Services através do Short Message Service.

Bruno de Medeiros Ledesma

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina
Caixa Postal 15.064 CEP 88040-900 - Campus Universitário – Florianópolis/SC

led@inf.ufsc.br

Abstract. *This paper is a about a framework implementation which purpose is to handle Short Messages send and receiving actions and provide business logic and content with Web Services. For that was made an abstract server model that has been extended in a concrete implementation of a Web Services Server and a Short Messages Service Server.*

Resumo. *Este artigo descreve a implementação de um framework cujo propósito é tratar recebimento e envio de mensagens SMS, onde o conteúdo e a lógica de negócio são solicitados à Web Services e respondidos para o cliente SMS. Para isso, foi criado um modelo abstrato de servidor, o qual foi estendido para implementação concreta de um servidor de Web Services e um servidor de mensagens SMS.*

1. Introdução

Em maio de 2007 a Anatel divulgou que chegou a 105 milhões o número de aparelhos de telefones celulares no Brasil. Este dado nos da uma noção do alto nível de utilização do telefone celular no dia a dia da população brasileira. [1]

Este alto número de aparelhos celulares nos dá a perspectiva de uma inclusão digital acontecendo longe dos tradicionais computadores pessoais (PCs), e ela está intimamente relacionada a mobilidade e ao baixo custo desses aparelhos. Com isso o serviço SMS, o mais popular serviço de mensagens nos aparelhos celulares de todo o mundo, ganha o potencial de maior cliente de requisições de serviços no Brasil. Não só pelos números constatados, mas por ser uma das funcionalidades mais usadas no aparelho celular e de fácil aprendizado. O uso de navegadores WAP e aplicativos nos telefones celulares ainda são restritos a uma parcela reduzida da população que possui o conhecimento e o poder econômico para usá-los. Estes aplicativos ainda são limitados no que se refere a solucionar efetivamente problemas no cotidiano de seus usuários. O

[1] http://www.anatel.gov.br. Acesso em: 18 de maio de 2007.

serviço de mensagens SMS pode ser usado por quase todos os modelos de celulares existentes e uma grande parcela dos seus usuários tem domínio e poderio econômico para utilizá-lo.

Atualmente, o SMS tem um alto índice de uso por consumidores, mas um baixo nível de utilização para soluções de negócio. Em outras palavras, enquanto bilhões de mensagens SMS são trocadas entre consumidores individuais todo mês, o seu uso como parte da estratégia de negócio das empresas é relativamente baixo. Apesar de outras mídias como rádio, internet e televisão estarem disponíveis para publicidade, o SMS ainda não é amplamente utilizado. Este serviço, segundo um relatório da empresa Clickatell [WATERMEYER, 2003], tem as seguintes características: alto alcance, baixo custo, bom poder de retenção na memória dos seus usuários (ver tabela 1). Tais características fazem do SMS uma potencial ferramenta de marketing para as organizações.

Tabela 2 – Comparação entre mídias de comunicação padrão

Meio de comunicação	**Alcance**	**Custo**	**Retenção na memória**
Televisão	**Um dos maiores**	Muito Alto	Bom
Radio	Médio	Médio	Ruim
Internet (banners)	Alto	Médio	Em queda
E-mail	Alto	Baixo	Muito baixo
Telefone	Médio	Alto	Médio
Mídia móvel	Médio	Alto	Médio
Correio	Alto	Alto	Médio
Interação Pessoal	Baixo	Alto	Alto
SMS	Alto	Baixo	Alto

Fonte: WATERMEYER, 2003

2. Proposta de um framework

2.1 Especificação

O framework proposto provê classes escritas na linguagem de programação Java que através do mecanismo de herança e implementação de suas interfaces, facilitam o atendimento de requisições a serviços web feitos por mensagens SMS. No entanto, para

que em uma re-fatoração futura seja possível o framework abstrai o tipo do cliente. O núcleo do mesmo é interfaceador das requisições aos serviços e tratará o processo de *parsing* da mensagem recebida, e de *parsing* da resposta a ser enviada.

A figura 1 ilustra como deverá funcionar este mecanismo.

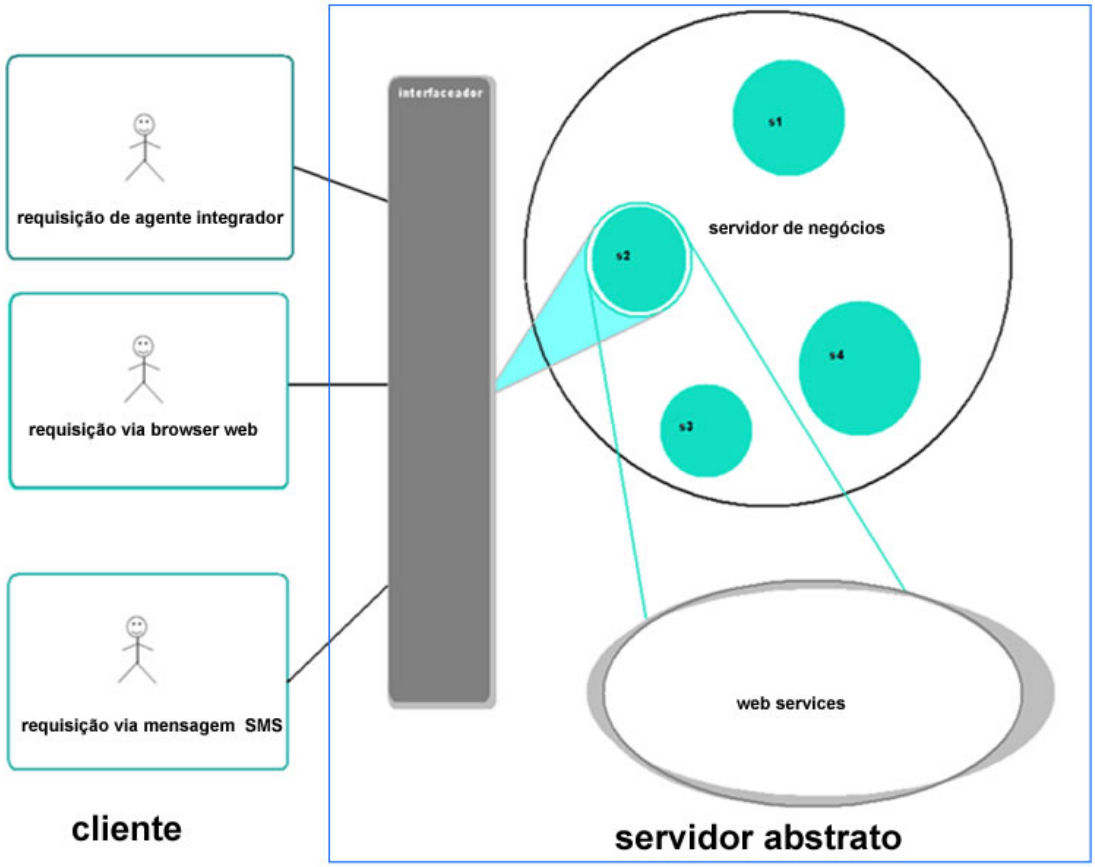

Figura 1 - Abstração da modelagem do framework

A figura 1 se divide em dois contextos. O lado do cliente que deve ser abstraído da implementação SMS proposta pelo framework e o lado de um servidor abstrato. O servidor abstrato trabalhará com o conceito de imagem única, abstraindo o cliente de seus componentes internos. Fará parte do framework uma camada que proverá as funcionalidades com a finalidade de requisitar os *Web Services* e responder corretamente.

2.2 Propósito

O framework tem como propósito inicial indicar uma especificação técnica para a criação de serviços de informação que utilizem mensagens SMS como cliente. Os Web Services serão uma camada isolada que proverá as informações com baixo acoplamento dos componentes de software e com alto grau independência da implementação.Para isto

foram projetadas classes e interfaces descritas neste artigo, bem como foi projetada uma aplicação para exemplificar a utilização do mesmo.

2.3 Projeto e classes

O framework desenvolvido tem o seu núcleo de utilização baseado em duas interfaces, como pode ser visto na figura 2. A interface *IServer* que descreve o comportamento padrão de um servidor para o framework. E a interface *IResponseHandler* que define o comportamento de uma classe que fará o *parsing* das mensagens e fará a conexão com o servidor de respostas para prover a resposta ao usuário do servidor.

2.3.1 Diagrama de Classes

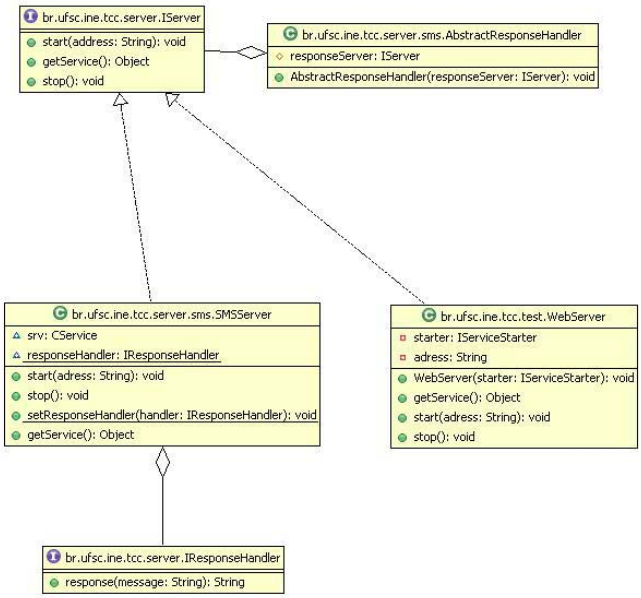

Figura 2 - Diagrama de classes do framework

2.4 Utilização

Para demonstrar a utilização do framework em uma aplicação real, teremos que utiliza-lo definindo algumas implementações:

1) A implementação do serviço de negócio a ser utilizado como exemplo.
2) A implementação do Servidor Web Service para responder as requisições do serviço;
4) A implementação do Servidor SMS para atender a necessidade de envio e recebimento de mensagens SMS;
5) Implementação da classe Java que tratará mensagens *(Handler)*, e executará o processo de *parsing* da mesma e invocação do serviço em utilização.

2.5 Aplicação proposta

Teremos como aplicação proposta para utilização do framework desenvolvido neste projeto a consulta dos horários previstos das linhas de ônibus em um determinado ponto de uma malha de transporte público. Para que tal aplicação seja possível tem-se como premissa:

- Cada ponto de ônibus deve ter um identificador único exposto ao seu usuário para que ele possa fazer a consulta;
- Uso de uma sintaxe simples que esteja explicita ao usuário ;
- Existe um limite de 160 caracteres na resposta do sistema;

No único serviço proposto por nossa aplicação teremos a seguinte sintaxe de consulta:

onibus <identificador do ponto>

Por exemplo, se o usuário se encontra no ponto de ônibus 234, e quer saber quais serão as linhas que passarão neste ponto e em que horário, ele enviara uma mensagem SMS para um determinado número, o número do servidor SMS, contendo o seguinte texto: "**onibus** 234".

O sistema conectará ao Web Service responsável pela resposta de tal consulta, e responderá algo como: 21h30 Linha 1; 21:45 Linha 2;

O número de linhas que o serviço irá listar é relativo, dependendo do número de caracteres da descrição da linha de ônibus, teremos mais ou menos estimativas passadas ao usuário. Com isso, devemos-nos preocupar com a construção da resposta para que ele efetivamente cumpra seu papel de informar o usuário tendo o tamanho de 160 caracteres.

2.5.1 Web Service

1. Interface do serviço

O classe LinhaOnibus é implementadora do serviço especificado pela Interface ILinhaOnibus. É única classe de serviço implementada neste artigo e foi configurada para ser um Web Service através da JSR-181 [2]que utiliza o recurso *annotations* da linguagem Java. Para a construção da aplicação proposta foi implementado o método desta classe denominado getProximasLinhasByIdPonto que recebe como parâmetro o identificador da linha de ônibus e retorna a previsão dos horários dos próximos ônibus para este ponto. LinhaOnibus.java tem um atributo do tipo LinhaOnibusDao, que implementa o padrão de projetos DAO, responsável pela coleta dos dados da nossa aplicação.

2. O padrão Data Acess Object

O padrão de projeto DAO ou Data Access Object, tem o intuito de separar o código que depende de características especificas dos recursos de acesso aos dados, do código que programa as regras de negócio. Isso dificulta a manutenção da aplicação quando vamos alterar ou adicionar funcionalidades relacionadas ao modelo de dados da sua aplicação. O padrão de projeto Data Access Object :

- Separa a interface cliente dos recursos de dados dos mecanismos de acesso aos dados;
- Adapta a API de acesso a recursos de dados a uma interface cliente genérica;

O padrão DAO permite o que o acesso aos dados seja alterado independentemente do código que usa os dados.

3. Persistência dos dados

Para a camada de acesso aos dados utilizando o padrão de projeto DAO, utilizou-se o banco de dados orientado a objeto db4o.

O db4o, é um banco de dados orientado a objetos que trabalha com o conceito de persistência via linguagem de programação.Isso quer dizer que o desenvolvedor deve conhecer simplesmente a linguagem de programação de sua preferência, sem a necessidade do aprendizado de uma linguagem nativa do banco de dados.Ele possui implementações para diversas linguagens, como Java e .Net.

Este banco de dados tem ampla aceitação no mercado corporativo. Possui usuários e clientes de cerca 170 países, da Albânia até o Zimbábue, e de empresas líderes de mercado em seus segmentos como a Boeing, Bosch, Intel, entre outros.

---

[2] *JSR 181: Web Services Metadata for the Java™ Platform***.** Disponível em: http://jcp.org/en/jsr/detail?id=181. Acesso em : 20 de junho de 2007.

A motivação do seu uso neste artigo foi a simplicidade na programação da camada de persistência dos objetos do domínio de negócio, e o baixo grau de complexidade das consultas que a aplicação desenvolvida requeria. Para consultas mais complexas o db4o perde um pouco uma de suas características mais marcantes, que é a produtividade de desenvolvimento.

4. Servidor Web Services

Na escolha de uma implementação Java para Web Services, o artigo teve que realizar uma uma escolha entre a implementação da "Apache Software Foundation", o Apache Axis e a implementação da "The Codehaus - Opensource Software Community" o XFire.

O principal critério utilizado para escolha da implementação foi a facilidade de criação de um Web Service a partir de uma classe Java qualquer. Também foi levado em conta a documentação e o nível de atividade das comunidades de usuários dos respectivos projetos. Diante disso, a primeira premissa do estudo foi que a implementação da JSR 181 seria pré-requisito no processo de escolha. Ambas as implementações dão suporte a JSR 181, que utiliza de um recurso contido na linguagem Java na versão 5, as anotations, para a criação de Web Services. Porém, a implementação do XFire é mais madura e possui melhor documentação que a do Apache Axis, que só implementa em sua versão 1.2 (Axis2), uma das mais recentes dentro de seu histórico. Um comparativo na documentação dos projetos e no suporte à duvidas dentro de fóruns e artigos de sites especializados na tecnologia deixou a convicção de que o XFire traria maior produtividade e rapidez para a implementação de Web Services em Java.
Para explicar o desenvolvimento do servidor Web Services utilizado como exemplo de utilização do framework, devemos entender primeiro como funciona a API XFire. Segue abaixo a definição das principais classes deste framework.

- **XFireFactory**

Esta classe que segue o padrão de projeto Factory, fabrica instancias do XFire, classe que concentra quase todas as funcionalidades do XFire.

- **ServiceFactory**

Esta classe que segue o padrão de projeto Factory, fabrica instancias dos serviços do XFire, ou seja Web Services.

- **Service**

Representa um serviço do XFire.

No exemplo abaixo, o serviço estará modelado como um objeto Java simples (pertencente à classe YourService) que será mapeado para um Web Service.

```
XFire xfire = XFireFactory.newInstance().getXFire();
ServiceFactory factory = new ObjectServiceFactory(xfire.getTransportManager(), null);
Service service = factory.create(YourService.class);
```

Isto vai criar um documento SOAP 1.1 para sua Classe. Se esta classe possui tipos complexos(ex: que não sejam Strings, Inteiros ou boleanos), o XFire vai automaticamente tentar serializar eles para você. Depois de criado, o serviço deve ser registrado:

```
xfire.getServiceRegistry().register(service);
```

Porém está forma de implementação fará com que o desenvolvedor da aplicação tenha que configurar alguns arquivos XML e fazer configurações no servidor HTTP a ser utilizado. Para abstrair problemas como este usa-se a implementação da JSR-181 que o XFire provê. Para diminuir a complexidade utiliza-se o recurso de inicializar um servidor Http chamado Jetty internamente a aplicação.
Primeiramente inicializamos as mesmas entidades do exemplo anterior porém agora instanciamos uma AnnotationServiceFactory. Está classe criará Web Services a partir de uma classe Java que esteja "anotada" como tal.

```
XFire xfire = XFireFactory.newInstance().getXFire();
AnnotationServiceFactory factory = new
AnnotationServiceFactory(xfire.getTransportManager());
Service service = factory.create(YourService.class);
xfire.getServiceRegistry().register(service);
```

A classe Java que sera transformada em Web Service deve possuir a anotação @WebService para que o XFire a reconheça como uma classe que está apta a ser transformada seguindo a JSR-181. Abaixo segue exemplo de tal anotação

```
@WebService(endpointInterface = "br.ufsc.ine.tcc.services.ILinhaOnibus")
public class LinhaOnibus implements ILinhaOnibus
```

Para inicializar o Jetty basta simplesmente instanciar um objeto da classe XFireHttpServer e invocar o método start() do mesmo. Com isso tem-se toda a infra-estrutura de código Java necessária para a instanciação de um servidor Web que responderá as requisições à Web Services.

2.5.2 Servidor SMS

A implementação do servidor responsável pelo envio e recebimento de mensagens SMS foi feita com a utilização de uma biblioteca que permite o estas funcionalidades através de um modem GSM ou um telefone celular conectado a estação de desenvolvimento.

a. Critérios de escolha

Durante o processo de pesquisa da ferramenta que auxiliaria no envio e recebimento de mensagens SMS tivemos como critério de escolha:

- Abstração da complexidade das tecnologias envolvidas;

- Baixa curva de aprendizado e baixo custo de implementação;

Seguindo esta linha, a pesquisa teve como candidatos finais a biblioteca de código aberto SMSLib e a implementação do protocolo SMPP para Java, OpenSMPP. No entanto, a segunda teve como característica eliminatória a necessidade de acesso a um SMSC. Tal requisito necessita a contratação de um serviço de acesso e utilização de um SMSC de uma operadora de telefonia.

Sendo assim, foi escolhida a biblioteca SMSLib , mas vale ressaltar a necessidade da utilização de uma implementação utilizando o protocolo SMPP para aplicações de grande porte. A biblioteca SMSLib permite o envio de cerca de 7 mensagens por minuto. Tal taxa de envio seria insatisfatória para a criação de aplicações de grande porte. O protocolo SMPP e o uso da infraestrutura disponível a partir de um SMSC atendem a demanda de uma taxa de envio de mensagens maior.

b. SMSLib

SMSLib é uma biblioteca que permite ao desenvolvedor criar aplicações que enviem e recebam mensagens SMS via um modem GSM. Você pode usar a SMSLib com um modem GSM dedicado ou com um telefone celular suportado por ela.

Características do SMSLib.

SMSLib trabalha com o protocolo PDU. SMSLib para Java (v2.1.o ou mais recente) também suporta protocolos TEXT, com o alfabeto GSM/HEX.

- Suporta o envio de mensagens WAP PUSH SI
- Suporta o envio e recebimento de mensagens 7-bit, 8-bit ou UCS2.
- Suporta mensagens multi-part
- Provê informações do modem GSM ou telefone celular: fabricante, modelo, nível de bateria e de sinal etc.

A biblioteca SMSLib comunica com um telefone móvel ou modem GSM através da porta serial. Ela usa um conjunto de comandos 3GPPAT, que é similar ao seu ancestral o Hayes AT, para controlar o modem. Estes comandos são enviados e recebidos através da porta serial.

Para a utilização da bilblioteca tem-se como pre-requisito a conexao via porta serial do seu modem GSM com o computador que possuirá o código a ser executado. Estes modems podem estar conectdos via cabos seriais. Mas alguns podem ser conectar via *Bluethooth*, IrDA ou cabos USB. No entanto, estas camadas de conexão devem estar mapeadas para portas seriais através de *drivers* do dispositivo instalados no computador que excuta o código da aplicação.

3. Conclusão

O artigo propôs uma alternativa de integração de duas tecnologias que suprem algumas necessidades no contexto de desenvolvimento de sistemas nos dias de hoje. O SMS que tem uma larga base de usuários e os *Web Services* tem como propósito facilitar a

interoperabilidade entre sistemas distintos com baixo nível de acoplamento.

Verificou-se também a viabilidade de utilização destas tecnologias na criação de soluções de negócio, principalmente para servidores de informação. O conceito de arquiteturas orientadas a serviço (SOA) regeu o desenvolvimento da aplicação e da construção do framework, facilitando assim a convergência na utilização de serviços de diferentes tipos de rede.

A viabilidade comercial do uso de tais tecnologias e de tal framework para soluções de negócio ainda depende de um estudo mais detalhado dos custos de implantação deste modelo em um sistema de grande porte. Para viabilização de tais soluções, alguns fatores como acordos entre empresas e operadoras de telefonia e cooperação entre grupos de empresas que necessitam de tais soluções, são relevantes e necessitam ser levados em conta. Porém, mostrou-se que tecnicamente existe mais do que uma alternativa de implementação de tais soluções de negócio. Cabe ao contexto de utilização da solução, escolher as estratégias de implementação que suprem a sua necessidade.

4. Referências

NAN, SHI. *Mobile Commerce Applications*. IGI Publishing, 2004. 358 pag.

ELKARRA, Naseean. *A Web Services Strategy for Móbile Phones***.**Disponível em:

http://webservices.xml.com/pub/a/ws/2003/08/19/mobile.html. Acesso em: 8 de fevereiro

de 2007.

SILVINO JR, JOÃO BOSCO. SMPP – *Protocolos e Aplicações***.** Disponível em:

http://www.inforede.net/Technical/Layer_1/Wireless_Mobile/SMPP_specs_(POR).pdf. Acesso em 10 de junho de 2007.

SMS FORUM. *Short Message Peer to Peer Protocol Specification v5.0 19-February-2003*. Disponível em: www.smsforum.net. Acesso em 10 de junho de 2007.

Dao Pattern. Disponível em: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.htm. Acesso em 10 de junho de 2007.

Documentação Banco de dados db4o. Disponível em: http://developer.db4o.com/docs/ . Acesso em 10 de junho de 2007.

Anexo 2 – Código fonte do framework

```
package br.ufsc.ine.tcc.server;
/**
 * Framework SMS_WEB_SERVICE
 * @author Bruno Ledesma
 *
 */
public interface IResponseHandler{

	public String response(String message);

}


package br.ufsc.ine.tcc.server;

import java.util.Map;

/**
 * Framework SMS_WEB_SERVICE
 * @author Bruno Ledesma
 *
 */
public interface IServer {

	public void start();

	public Object getService() throws Exception ;

	public void stop();

	public Map<String,String> getConfigurationMap();




}


package br.ufsc.ine.tcc.server.sms;
/**
 * Framework SMS_WEB_SERVICE
 * @author Bruno Ledesma
 *
 */
import br.ufsc.ine.tcc.server.IServer;

public abstract class AbstractResponseHandler {

	protected IServer responseServer;

	public AbstractResponseHandler(IServer responseServer) {
		super();
		this.responseServer = responseServer;
	}

}

package br.ufsc.ine.tcc.server.sms;

import br.ufsc.ine.tcc.server.IResponseHandler;
import br.ufsc.ine.tcc.server.IServer;
import br.ufsc.ine.tcc.server.sms.parser.LinhaOnibusParser;
import br.ufsc.ine.tcc.services.ILinhaOnibus;

public class HorariosByPontoSMSResponseHandler extends AbstractResponseHandler
implements IResponseHandler {

	LinhaOnibusParser parser = new LinhaOnibusParser();
	public HorariosByPontoSMSResponseHandler(IServer responseServer) {
		super(responseServer);
```

```
		// TODO Auto-generated constructor stub
	}

	/**
	 * Implementacao do parsing da mensagem e chamada do servico do responseServer;
	 */
	public String response(String message) {
		ILinhaOnibus linha = null;
		try {
			linha = (ILinhaOnibus) responseServer.getService();
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		return linha.getProximasLinhasByIdPonto(parser.parser(message));
	}



}

package br.ufsc.ine.tcc.server.sms;

import java.util.Map;

import org.smslib.CIncomingMessage;
import org.smslib.COutgoingMessage;
import org.smslib.CService;
import org.smslib.ISmsMessageListener;

import br.ufsc.ine.tcc.server.IResponseHandler;
import br.ufsc.ine.tcc.server.IServer;


/**
 * Implementacao do servidor SMS.
 * @author bruno
 *
 */
public class SMSServer implements IServer
{
	CService srv;
	private Map configurationMap;
	static IResponseHandler responseHandler;


	private static class ConcreteMessageListener implements ISmsMessageListener
	{



		public boolean received(CService service, CIncomingMessage message)
		{
			// Mostra a mensagem recebida
			System.out.println("[mensagem recebida!!!]====> msg: " +
message.getText());

			try
			{
				service.sendMessage(new
COutgoingMessage(message.getOriginator(), responseHandler.response(message.getText())));
			}
			catch (Exception e)
			{
				System.out.println("Could not send reply message!");
				e.printStackTrace();
			}

			// Return false para deixar a mensagem na memoria
			// true para deleta-la
			return false;
		}
	}
```

```
	public void start()
	{
		// Instancia CService com propriedades setadas no ConfigurarionMap.
Modelo SonyEricsson W200 utilzado no exemplo e baud rate 57600 é a padrao;
		srv = new
CService(getConfigurationMap().get("port").toString(),57600,"Sony Ericsson", "W200");

		// Esta é a classe listener que sera chamada para cada mensagem recebida.
		ConcreteMessageListener smsMessageListener = new
ConcreteMessageListener();

		System.out.println("  Using " + CService._name + " " +
CService._version);
		System.out.println();
		try
		{
			// Se o dispositivo GSM tem o PIN protegido , entre com o PIN
aquie. verifique o manual do seu dispostivo ou outra documentacao do fabricante do mesmo
			srv.setSimPin("0000");

			// Normalmente, vce seta o numero do SMSC, que esta no SIM Card de
dispositivos GSM
			srv.setSmscNumber("");
			// Conectando... alguma excecoes podem ser disparadas aqui.
			srv.connect();

			// Lets get info about the GSM device...
			System.out.println("Mobile Device Information: ");
			System.out.println("  Manufacturer  : " +
srv.getDeviceInfo().getManufacturer());
			System.out.println("  Model         : " +
srv.getDeviceInfo().getModel());
			System.out.println("  Serial No     : " +
srv.getDeviceInfo().getSerialNo());
			System.out.println("  IMSI          : " +
srv.getDeviceInfo().getImsi());
			System.out.println("  S/W Version   : " +
srv.getDeviceInfo().getSwVersion());
			System.out.println("  Battery Level : " +
srv.getDeviceInfo().getBatteryLevel() + "%");
			System.out.println("  Signal Level  : " +
srv.getDeviceInfo().getSignalLevel() + "%");



			// Seta o handler de mensagens
			srv.setMessageHandler(smsMessageListener);

			// Seta o intervalo de consulta ao dispostivo (pooling) em
segundos
			srv.setAsyncPollInterval(10);

			// Seta classe de mensagens a ser lidas.
			srv.setAsyncRecvClass(CIncomingMessage.MessageClass.Unread);

			// Muda para o modo POOL assincrono;
			srv.setReceiveMode(CService.ReceiveMode.AsyncPoll);


		//

		}
		catch (Exception e)
		{
			e.printStackTrace();
			try {
				//disconectando caso aconteca problemas
				srv.disconnect();
			} catch (Exception e1) {
				// TODO Auto-generated catch block
				e1.printStackTrace();
			}
		}
```

```
        }


        public void stop() {
                try {
                        srv.disconnect();
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

        }



        public static void setResponseHandler(IResponseHandler handler) {
                responseHandler = handler;
        }



        public Object getService() throws Exception {
                // TODO Auto-generated method stub
                return null;
        }



        public Map getConfigurationMap() {
                        return configurationMap;
        }



        public void setConfigurationMap(Map configurationMap) {
                this.configurationMap = configurationMap;
        }



}

package br.ufsc.ine.tcc.server.sms.parser;

import org.springframework.util.StringUtils;

public class LinhaOnibusParser {

        public String parser(String str){
                String x= "onibus";
                String[] ar = StringUtils.tokenizeToStringArray(str, x);


                return ar[0];
        }

}

package br.ufsc.ine.tcc.server.web.xfire;

import java.util.Map;

import org.codehaus.xfire.XFire;
import org.codehaus.xfire.XFireFactory;
import org.codehaus.xfire.annotations.AnnotationServiceFactory;
import org.codehaus.xfire.server.http.XFireHttpServer;
import org.codehaus.xfire.service.Service;

import br.ufsc.ine.tcc.server.IServer;
import br.ufsc.ine.tcc.services.ILinhaOnibus;
import br.ufsc.ine.tcc.services.LinhaOnibus;
import br.ufsc.ine.tcc.services.LinhaOnibus2;
import br.ufsc.ine.tcc.services.resources.Constantes;
```

```
public class AnnotationServiceStarter implements IServiceStarter
{
    XFireHttpServer httpServer;
       private IServer server;

    public void start() throws Exception
    {
       XFire xfire = XFireFactory.newInstance().getXFire();
       AnnotationServiceFactory factory = new
AnnotationServiceFactory(xfire.getTransportManager());
       Service service = factory.create(getServiceInstanceClass());
       xfire.getServiceRegistry().register(service);

        // Start the HTTP server
        httpServer = new XFireHttpServer();
        Map serverConfiguration = server.getConfigurationMap();
        String port = (String) serverConfiguration.get("port");
        String url = (String) serverConfiguration.get("url");

        httpServer.setPort(new Integer(port));
        httpServer.start();
    }

    public void stop() throws Exception
    {
        httpServer.stop();
    }

       public Class getServiceInstanceClass() {
              return LinhaOnibus.class;
       }

       public void setServer(IServer server) {
              this.server = server;

       }
}
```

```
package br.ufsc.ine.tcc.server.web.xfire;

import br.ufsc.ine.tcc.server.IServer;

/**
 * Framework SMS_WEB_SERVICE
 * @author Bruno Ledesma
 *
 */
public interface IServiceStarter {

       public abstract void start() throws Exception;

       public abstract void stop() throws Exception;

       public abstract Class getServiceInstanceClass();

       public abstract void setServer(IServer server);

}
```

```
package br.ufsc.ine.tcc.server.web.xfire;

import org.codehaus.xfire.XFire;
import org.codehaus.xfire.XFireFactory;
import org.codehaus.xfire.server.http.XFireHttpServer;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;
import org.codehaus.xfire.service.invoker.BeanInvoker;

import br.ufsc.ine.tcc.server.IServer;
import br.ufsc.ine.tcc.services.LinhaOnibus2;

import com.sun.xml.bind.v2.ClassFactory;

public class ServiceStarter implements IServiceStarter
{
```

```
    XFireHttpServer httpServer;
        private IServer server;


    /* (non-Javadoc)
         * @see br.ufsc.ine.tcc.xfire.IServiceStarter#start()
         */
    public void start() throws Exception
    {
        // Cria um servico XFire
        ObjectServiceFactory serviceFactory = new ObjectServiceFactory();
        Service service = serviceFactory.create(getServiceInstanceClass());
        service.setInvoker(new
BeanInvoker(ClassFactory.create(getServiceInstanceClass())));

        // Register the service in the ServiceRegistry
        XFire xfire = XFireFactory.newInstance().getXFire();
        xfire.getServiceRegistry().register(service);

        // Start the HTTP server
        httpServer = new XFireHttpServer();
               String port = (String) server.getConfigurationMap().get("port");
        httpServer.setPort(new Integer(port));
        httpServer.start();
    }

    /* (non-Javadoc)
         * @see br.ufsc.ine.tcc.xfire.IServiceStarter#stop()
         */
    public void stop() throws Exception
    {
        httpServer.stop();
    }

        public Class getServiceInstanceClass() {

               return LinhaOnibus2.class;
        }

        public void setServer(IServer server) {
               this.server = server;

        }
}

package br.ufsc.ine.tcc.server.web.xfire;

import java.net.MalformedURLException;
import java.util.Map;

import org.codehaus.xfire.client.XFireProxyFactory;
import org.codehaus.xfire.service.Service;
import org.codehaus.xfire.service.binding.ObjectServiceFactory;

import br.ufsc.ine.tcc.server.IServer;
import br.ufsc.ine.tcc.services.ILinhaOnibus;

public class WebServer implements IServer{
        private IServiceStarter starter ;
        private Map<String,String> configurationMap;


        // static IServiceStarter starter = new ServiceStarter();

        public WebServer(IServiceStarter starter){
               starter.setServer(this);
               this.starter = starter;
        }

        public Object getService() throws Exception {
               // Cria um service model para o cliente
               ObjectServiceFactory serviceFactory = new ObjectServiceFactory();
               Service serviceModel = serviceFactory.create(ILinhaOnibus.class);
               // Cria um proxy para o cliente
               XFireProxyFactory proxyFactory = new XFireProxyFactory();
```

```
		String serviceName = starter.getServiceInstanceClass().getSimpleName();
		ILinhaOnibus linha = null;
		String port = (String) configurationMap.get("port");
		String url = (String) configurationMap.get("url");
		try {
			linha = (ILinhaOnibus) proxyFactory.create(serviceModel,url+ port
+  "/" + serviceName);
		} catch (MalformedURLException e) {
			e.printStackTrace();
		}
		return linha;

	}


	public void start()  {
		try {
			starter.start();
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}


	public void stop() {
		try {
			starter.stop();
		} catch (Exception e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}
	}

	public Map<String,String> getConfigurationMap() {
		// TODO Auto-generated method stub
		return configurationMap;
	}

	public void setConfigurationMap(Map<String,String> configurationMap) {
		this.configurationMap = configurationMap;
	}



}
```

Anexo 3 – Código fonte da aplicação proposta

```
package br.ufsc.ine.tcc.dao;

import java.io.File;
import java.util.Date;



import br.ufsc.ine.tcc.services.resources.Constantes;

import com.db4o.Db4o;
import com.db4o.ObjectContainer;

/**
 * Implementacao de acesso DAO ao DB40.
 * @author bruno
 *
 */
public class BasicDao {

	com.db4o.ObjectContainer db ;
```

```
        boolean noPopulation = false;

        public BasicDao() {
                super();
                Db4o.configure().objectClass(Date.class).storeTransientFields(true);
                File file = new File(Constantes.DB_NAME);
                if (file.exists())
                        noPopulation = true;
                this.openConnection();

        }

        protected void openConnection(){
                db = Db4o.openFile(Constantes.DB_NAME);
        }

        public ObjectContainer getDB(){
                return db;
        }

        protected void closeConnection(){
                db.close();
        }



}

package br.ufsc.ine.tcc.dao;

import java.util.Collection;
import java.util.Set;

public interface ILinhaOnibusDao {

        public abstract Collection getProximasLinhasByIdPonto(String idPonto);

        public abstract Set getPontos();

        public abstract void populaBanco();

}

package br.ufsc.ine.tcc.dao;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Collection;
import java.util.GregorianCalendar;
import java.util.HashSet;
import java.util.Iterator;
import java.util.List;
import java.util.Set;

import br.ufsc.ine.tcc.pojo.EstimativaPontoPojo;
import br.ufsc.ine.tcc.pojo.LinhaOnibusPojo;
import br.ufsc.ine.tcc.services.resources.Constantes;

import com.db4o.ObjectSet;

public class LinhaOnibusDao extends BasicDao implements ILinhaOnibusDao {

        /**
         * Retorna as proximas linhas que passarao naquele ponto dado
         * @param idPonto: identifiacador do ponto
         */
        public Collection getProximasLinhasByIdPonto(String idPonto) {

         EstimativaPontoPojo p = new EstimativaPontoPojo();
         p.setIdPonto(idPonto);
         ObjectSet set = getDB().get(p);
         Iterator it = set.iterator();
         Calendar cal = GregorianCalendar.getInstance();
        Timestamp t = new Timestamp(Constantes.DATA_BASE_HORARIO.getTime());
```

```
        t.setHours(cal.get(cal.HOUR_OF_DAY));
        t.setMinutes(cal.get(cal.MINUTE));
         Collection resultado = new ArrayList();
         while (it.hasNext()){
                EstimativaPontoPojo e = (EstimativaPontoPojo) it.next();
                if (e.getHorarioPrevisto().getHours() > t.getHours())
                        resultado.add(e);
                else if (e.getHorarioPrevisto().getHours() == t.getHours() &&
e.getHorarioPrevisto().getMinutes() > t.getMinutes())
                        resultado.add(e);


         }

         return resultado;
        }

        /**
         * Pega os pontos de onibus cadastrados no banco.
         * @return Set de pontos de Onibus.
         */
        public Set getPontos(){

                 ObjectSet set = getDB().get(EstimativaPontoPojo.class);
                 Iterator it = set.iterator();

                 Set resultado = new HashSet();
                 while (it.hasNext()){
                        EstimativaPontoPojo est= (EstimativaPontoPojo) it.next();
                        resultado.add(est.getIdPonto());
                 }


                return resultado;
                }

/**
 * Metodo que popula o banco com valores aleatorios.
 */
        public void populaBanco() {
                if(noPopulation)
                        return;

                LinhaOnibusPojo linhaMorro = new LinhaOnibusPojo("Volta ao Morro");
                LinhaOnibusPojo linhaUFSC = new LinhaOnibusPojo("Ufsc SemiDireto");
                List pontos = new ArrayList();
                for (int i = 0 ;i < 10;i++){
                        long pt = (long) (System.currentTimeMillis() * Math.random());
                        pontos.add(String.valueOf(pt));
                        System.out.println(pt);
                }
                linhaMorro.setPontos(pontos);
                linhaUFSC.setPontos(pontos);
                int horainicial = 7;
                int minutoInicial = 0;

                db.set(linhaMorro);
                db.set(linhaUFSC);
                /**
                 * linha do morro soh as 7 horas, demorando 5 minutos de um ponto ao
outro
                 */
                for (int z = 0;z < 5 ;z++){
                        String idponto = (String) pontos.get(z);

                        Timestamp time = new
Timestamp(Constantes.DATA_BASE_HORARIO.getTime());
                        time.setHours(horainicial);
                        time.setMinutes(minutoInicial);
                        minutoInicial = minutoInicial + 5;
                        EstimativaPontoPojo estimativaPonto = new EstimativaPontoPojo();
                        estimativaPonto.setIdPonto(idponto);
                        estimativaPonto.setLinha(linhaMorro);
                        estimativaPonto.setHorarioPrevisto(time);
                        System.out.println("Ponto :: " + idponto + " horario:: " +
```

```
time.toString() + " LINHA:: " +linhaMorro.getNome());
                        db.set(estimativaPonto);

                }

                /**
                 * linha da ufsc soh as 22 horas, demorando 2 minutos de um ponto ao
outro
                 */
                horainicial = 22;
                minutoInicial = 2;
                for (int z = 0; z < 10 ;z++){
                        String idponto = (String) pontos.get(z);
                        Timestamp time = new
Timestamp(Constantes.DATA_BASE_HORARIO.getTime());
                        time.setHours(horainicial);
                        time.setMinutes(minutoInicial);
                        minutoInicial = minutoInicial + 2;
                        EstimativaPontoPojo estimativaPonto = new EstimativaPontoPojo();
                        estimativaPonto.setIdPonto(idponto);
                        estimativaPonto.setLinha(linhaUFSC);
                        System.out.println("Ponto :: " + idponto + " horario:: " +
time.toString() + " LINHA:: " +linhaUFSC.getNome());
                        estimativaPonto.setHorarioPrevisto(time);
                        db.set(estimativaPonto);
                }

                getDB().commit();

        }

}
```

```
package br.ufsc.ine.tcc.pojo;

import java.sql.Timestamp;
/**
 * Entidade que contem valores de estimativas das linhas para determinado ponto.
 * @author bruno
 *
 */
public class EstimativaPontoPojo {

        private LinhaOnibusPojo linha;
        private Timestamp horarioPrevisto;
        private String idPonto;


        public String getIdPonto() {
                return idPonto;
        }
        public void setIdPonto(String idPonto) {
                this.idPonto = idPonto;
        }
        public LinhaOnibusPojo getLinha() {
                return linha;
        }
        public void setLinha(LinhaOnibusPojo linha) {
                this.linha = linha;
        }
        public Timestamp getHorarioPrevisto() {
                return horarioPrevisto;
        }
        public void setHorarioPrevisto(Timestamp horarioPrevisto) {
                this.horarioPrevisto = horarioPrevisto;

        }

}


package br.ufsc.ine.tcc.pojo;

import java.util.Collection;
/**
 * Entidade que representa a linha de onibus,
```

```
 * atributos: nome da linha e uma colecao de pontos que compoe a linha
 * @author bruno
 *
 */
public class LinhaOnibusPojo {

	private String nome;
	private Collection pontos;
	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
	public Collection getPontos() {
		return pontos;
	}
	public void setPontos(Collection pontos) {
		this.pontos = pontos;
	}
	public LinhaOnibusPojo(String nome, Collection pontos) {
		super();
		this.nome = nome;
		this.pontos = pontos;
	}

	public LinhaOnibusPojo(String nome) {
		super();
		this.nome = nome;

	}
	public LinhaOnibusPojo() {
		// TODO Auto-generated constructor stub
	}


}



package br.ufsc.ine.tcc.test;

import br.ufsc.ine.tcc.server.sms.parser.LinhaOnibusParser;
import junit.framework.TestCase;

public class LinhaOnibusParserTester extends TestCase {
	LinhaOnibusParser parser = new LinhaOnibusParser();

	public void testParser() {
		String msg= "onibus 234";

		String ret = parser.parser(msg);

		assertEquals(ret, "234");
	}

}



package br.ufsc.ine.tcc.test;

import java.io.File;
import java.util.Iterator;
import java.util.Set;

import junit.framework.TestCase;
import br.ufsc.ine.tcc.services.LinhaOnibus;

public class LinhaOnibusTest extends TestCase {
	LinhaOnibus service;

	@Override
	protected void setUp() throws Exception {
		service = LinhaOnibus.createLinhaOnibus();
		super.setUp();
```

```
	}

	@Override
	protected void tearDown() throws Exception {
		// TODO Auto-generated method stub
		super.tearDown();
		service.finalize();
	}



	public void testLinha(){
		service.populaBanco();
		Set pontos = service.getPontos();
		Iterator it = pontos.iterator();
		while (it.hasNext()){
			String pt = (String) it.next();
			System.out.println("Ponto pesquisado: " + pt);
			System.out.println(service.getProximasLinhasByIdPonto(pt));
		}



	}


}

package br.ufsc.ine.tcc.test;

import java.util.HashMap;
import java.util.Map;

import br.ufsc.ine.tcc.server.IResponseHandler;
import br.ufsc.ine.tcc.server.sms.HorariosByPontoSMSResponseHandler;
import br.ufsc.ine.tcc.server.sms.SMSServer;
import br.ufsc.ine.tcc.server.web.xfire.AnnotationServiceStarter;
import br.ufsc.ine.tcc.server.web.xfire.WebServer;
import br.ufsc.ine.tcc.services.resources.Constantes;

public class ServerIntegrationTest {

	/**
	 * Metodo main para testar integracao entre os servidores
	 * @param args
	 */
	public static void main(String[] args) {

		/*
		 *  Instancia e inicia servidor web responsavel pelo atendimento de
requisicoes a web Services.
		 *  Este servidor web recebe um inicializador de servicos cuja a classe
provedora dos servicoes é
		 */
		WebServer webServer = new WebServer(new AnnotationServiceStarter());

		Map<String,String> webServerConf = new HashMap();
		webServerConf.put("port",Constantes.HTTP_WEB_SERVICE_PORT);
		webServerConf.put("url",Constantes.HTTP_WEB_SERVICE_HOST);

		webServer.setConfigurationMap(webServerConf);
		webServer.start();
		// Instancia handler para servicoes SMS com o servidor web anteriormente
instanciado.
		IResponseHandler handler = (IResponseHandler) new
HorariosByPontoSMSResponseHandler(webServer);
		SMSServer.setResponseHandler(handler);
		//Instancia e inicia servidor SMS responsavel pelo atendimento das
mensagens SMS.
		SMSServer smsServer = new SMSServer();

		Map smsServerConf = new HashMap();
		smsServerConf.put("port",Constantes.SMS_PORT);
```

```
                smsServer.setConfigurationMap(smsServerConf);
                smsServer.start();


        }

}

package br.ufsc.ine.tcc.services.resources;

import java.util.Date;



public class Constantes {

        public static final String HTTP_WEB_SERVICE_PORT = "8191";
        public static final String HTTP_WEB_SERVICE_HOST = "http://localhost:";
        public static final String DB_NAME = "f:\\bancoOnibus.yap";

        public static final String SMS_PORT = "COM6";

        public static final Date DATA_BASE_HORARIO = new Date("01/01/1900");

}

package br.ufsc.ine.tcc.services;

import java.util.Collection;
import java.util.Iterator;
import java.util.Set;

import javax.jws.WebService;

import br.ufsc.ine.tcc.dao.LinhaOnibusDao;
import br.ufsc.ine.tcc.pojo.EstimativaPontoPojo;


@WebService(endpointInterface = "br.ufsc.ine.tcc.services.ILinhaOnibus")
public class LinhaOnibus implements ILinhaOnibus {

        static LinhaOnibus instance;
        public static LinhaOnibus createLinhaOnibus() {
                if (instance == null){
                        instance =  new LinhaOnibus();
                }
                return instance;
        }


        LinhaOnibusDao dao = new LinhaOnibusDao();

        public String getProximasLinhasByIdPonto(String idPonto){
                Collection c= dao.getProximasLinhasByIdPonto(idPonto);
                Iterator it = c.iterator();
                String retorno ="";
                while (it.hasNext()){
                        EstimativaPontoPojo e = (EstimativaPontoPojo) it.next();
                        String linha = e.getLinha().getNome() + " - ";
                        String horario = e.getHorarioPrevisto().getHours() + "h" +
e.getHorarioPrevisto().getMinutes();
                        String ad = linha  + horario;
                        if ((retorno.length() + ad.length()) > 158)
                                break;
                        else
                                retorno = retorno + " " + ad;
                }
                return retorno;

        }

        public Set getPontos() {
                return dao.getPontos();
        }
```

```
        public void populaBanco() {
                dao.populaBanco();

        }

        public void finalize(){
                dao.getDB().commit();
                dao.getDB().close();
                instance = null;
        }

}

package br.ufsc.ine.tcc.services;

import javax.jws.WebService;

@WebService
public interface ILinhaOnibus {


        public abstract String getProximasLinhasByIdPonto(String idPonto);

}

package br.ufsc.ine.tcc.services;

import javax.jws.WebService;


@WebService(endpointInterface = "br.ufsc.ine.tcc.services.ILinhaOnibus")
public class LinhaOnibus2 implements ILinhaOnibus {

        public String getProximasLinhasByIdPonto(String idPonto){
                return "2"+ idPonto;
        }

}
```

7. Referências Bibliográficas

NAN, SHI. *Mobile Commerce Applications*. IGI Publishing, 2004. 358 pag.

ELKARRA, Naseean. *A Web Services Strategy for Móbile Phones*.Disponível em:

http://webservices.xml.com/pub/a/ws/2003/08/19/mobile.html. Acesso em: 8 de fevereiro

de 2007.

SILVINO JR, JOÃO BOSCO. SMPP – *Protocolos e Aplicações*. Disponível em:
http://www.inforede.net/Technical/Layer_1/Wireless_Mobile/SMPP_specs_(POR).pdf. Acesso em 10 de junho de 2007.

SMS FORUM. *Short Message Peer to Peer Protocol Specification v5.0 19-February-2003*. Disponível em: www.smsforum.net. Acesso em 10 de junho de 2007.

Dao Pattern. Disponível em:
http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.htm. Acesso em 10 de junho de 2007.

Documentação Banco de dados db4o. Disponível em: http://developer.db4o.com/docs/ . Acesso em 10 de junho de 2007.

---

[1] http://www.anatel.gov.br. Acesso em: 18 de maio de 2007.

[2] INTERNATIONAL ENGINEERING CONSORTIUM. *Web ProForum Tutorial:*

Wireless SMS. Disponível em: http:/www.iec.org/tutorials/wap/. Acesso em 10 de junho

de 2007.

[3] GSM ASSOCIATION. ***GSM – SMS Overview***. Disponível por WWW em

http:/www.gsmworld.com/technology/sms.html . Acesso em 10 de junho de 2007.

[4] ERICSSON, *Messaging-over-IP SMS TMD 30*. U.S.A, 26 de Janeiro de 2000. EM/LZIUB 202 20 R1a. CD-ROM.

[5] SNELL, JAMES. *Programming Web Services with SOAP*. Oreilly & Assoc,1ª Edição,2001. 350 pág.

[6] W3C. *SOAP Version 1.2 Part 1: Messaging Framework*. Disponível em: http://www.w3.org/TR/soap12-part1/#intro. Acesso em: 10 de junho de 2007.

[7] Johnson, Ralph and Foote, Brian. Designing Reusable Classes. *Journal of Orbject-Oriented Programming,* Vol. 1, No.2, pp. 22-35, 1988.

[8] GUIMARÃES, José de Oliveira. Frameworks São Paulo, nov. 2000. Disponível em: http:/www.dc.ufscar.br/~jose/courses/oc/apostilas-paterns.zip . Acesso em : junho de 2007.

[9] Gamma, Erich; Helm, Richard, Johnson, Ralph; Vlissides, John. *Design Patterns.* s.l., Addison Wesley, 1995.

[10] *JSR 181: Web Services Metadata for the Java™ Platform***.** Disponível em: http://jcp.org/en/jsr/detail?id=181. Acesso em : 20 de junho de 2007.