PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Programa de Pós-Graduação em Informática

UM CLIENTE WMS PARA DISPOSITIVOS MÓVEIS

Yuri Jorge Kimo

Belo Horizonte
2009

Yuri Jorge Kimo

UM CLIENTE WMS PARA DISPOSITIVOS MÓVEIS

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Informática.

Orientador: Clodoveu A. Davis Jr.

Belo Horizonte
2009

FICHA CATALOGRÁFICA

Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

K49c

Kimo, Yuri Jorge
Um cliente WMS para dispositivos móveis / Yuri Jorge Kimo. – Belo Horizonte, 2009.
65f. : il.

Orientador: Clodoveu A. Davis Jr.
Dissertação (Mestrado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-graduação em Informática.
Bibliografia.

1. Sistemas de informação geográfica – Teses. 2. Cartografia – Processamento de dados. 3. Sistemas de transmissão de dados I. Davis Jr., Clodoveu A. II. Pontifícia Universidade Católica de Minas Gerais. III. Título

CDU: 681.3.01:621.39

Bibliotecário: Fernando A. Dias – CRB6/1084

PUC Minas
Programa de Pós-graduação em Informática

FOLHA DE APROVAÇÃO

Um Cliente WMS para Computadores Móveis

YURI JORGE KIMO

Dissertação defendida e aprovada pela seguinte banca examinadora:

Prof. Clodoveu Augusto Davis Junior - Orientador (UFMG)
Doutor em Ciência da Computação - UFMG

Profa. Fátima de L. P. Duarte Figueiredo - Co-orientadora (PUC Minas)
Doutora em Ciências da Computação - UFMG

Prof. Jugurta Lisboa Filho - (UFV)
Doutor em computação - UFRGS

Prof. Marco Túlio de Oliveira Valente - (PUC Minas)
Doutor em Ciência da Computação – UFMG

Belo Horizonte, 11 de setembro de 2009.

Anel Rodoviário – KM 23,5, Rua Walter Ianni, 255 – São Gabriel / Belo Horizonte – MG / CEP: 31980-110
Fone: (31) 3439-5204 – Fax: (31) 3439-5204 – e-mail: mestradoinfsec@pucminas.br

Dedicatória

Este trabalho é dedicado com amor e carinho a minha esposa Pamella, minha mãe Deuzanira, meu pai Antônio e minha sogra Denise, pois dos esforços e sacrifícios feitos por eles que terei coragem e força para superar minhas limitações e alcançar meus ideais. Dedico também a meu irmão Igor, alguém que com quem sempre posso contar, não importa o motivo ou lugar.

AGRADECIMENTOS

Agradeço a meus amigos Bruno e Liliane, que no início da minha jornada não se cansavam de me lembrar que os sacrifícios presentes trariam vitórias futuras. Agradeço ao Pedro por me proibir de tocar na banda até concluir o mestrado, e ao meu orientador Clodoveu, que mostro o que é ser um professor e sem quem eu nunca teria concluído este trabalho.

RESUMO

A evolução das redes sem fio e a redução do custo de computadores móveis como smartphones e *Personal Digital Assistants* (PDA), aliadas à crescente disponibilidade de dados geográficos online, criaram um cenário propicio para o surgimento e crescimento de serviços baseados em localização. Para esse tipo de serviço, dois fatores são fundamentalmente importantes: a escalabilidade e o atendimento a padrões. Um serviço baseado em localização deve ser escalável de modo a permitir a conexão simultânea de um grande número de clientes móveis, limitados em capacidade de processamento, armazenamento e energia. Por outro lado, os padrões voltados para a comunicação tecnologicamente neutra e interoperável de dados geográficos impõem uma sobrecarga considerável aos clientes. Esta dissertação apresenta soluções que auxiliam na consulta a dados geográficos em uma arquitetura orientada por serviços baseada nos padrões do Open Geospatial Consortium, mais especificamente usando o Web Map Service (WMS), com foco especifico em dispositivos móveis, apontando suas características, vantagens e desvantagens. Uma alternativa às abordagens existentes é proposta, visando à redução de custo monetário no uso destes serviços, além de redução de carga de processamento, trafego de rede e aumento de vida útil da carga da bateria. Além disso, a proposta apresentada introduz mecanismos para reduzir a carga de trabalho também nos servidores, para melhorar sua escalabilidade.

Palavras chave: rede sem fio, dispositivos móveis, WMS, Web Map Service, dados geográficos e mapas digitais.

ABSTRACT

The evolution of wireless networks and the reduction of the cost of mobile computers such as Personal Digital Assistants (PDA) and smartphones, along with the growing availability of online geographic data, have created a favorable setting for the creation and growth of location-based services. For such services, tow factors are fundamentally important: scalability and standards. A location-based service must be scalable in order to allow the simultaneous connection of a large number of mobile clients, which are limited as to processing, storage, and energy. On the other hand, standards that allow for the technologically neutral and interoperable communication of geographic data impose a considerable overload to clients. This dissertation presents solutions that provide access to geographic data in a service-based architecture as proposed by the Open Geospatial Consortium, more specifically using the Web Map Service (WMS), with a focus on mobile devices, showing its characteristics, strong and weak points. An alternative to the existing approaches is proposed, aiming at communication cost reductions, along with lower processing, network traffic, and battery consumption costs. The proposed solution contributes to reduce the work load over the server as well, therefore improving its scalability potential.

Keywords: wireless net, mobile device, geographic date, WMS, Web Map Service and digital map.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

A convergência entre tecnologias tais como sistemas de informação geográficos (*Geographic Information Systems*, GIS), Internet, *Global Positioning System* (GPS), comunicação *wireless* e computação móvel criou um cenário propício para o surgimento de serviços móveis de informação geográfica (Tang 2003).

Com a evolução da tecnologia móvel e conseqüente queda de preços, aumento da disponibilidade das redes sem fio (WiFi, GPRS e 3G), e maior integração entre GPS e PDAs ou aparelhos celulares, não só os órgãos governamentais, mas também pequenas empresas e usuários comuns passaram a ter acesso a dados geográficos praticamente em tempo real.

Várias aplicações que utilizam dados geográficos podem fornecer ao usuário serviços baseados em localização como, por exemplo, indicar a farmácia ou restaurante mais próximos. No entanto, em geral não existem opções que permitam selecionar a fonte ou a natureza dos dados. Um dos *map browsers* mais conhecidos é o Google Maps[1], que disponibiliza informações que podem ser acessadas por qualquer pessoa que possua um computador ligado a Internet, além de fornecer uma *application programming interface* (API) que permite que os mesmos dados, porém com alguma limitação, sejam acessados por um celular ou PDA. Os dados fornecidos pelo Google Maps são pré-compilados, ou seja, são organizados em fragmentos de imagens previamente processadas e servidas aos usuários sem que exista a possibilidade de interação direta com elementos do mapa. O usuário não pode, por exemplo, escolher quais camadas de dados serão visualizadas, ou solicitar mais informação sobre um elemento visível que seja de seu interessa dentro do mapa básico.

No cenário móvel, as soluções para utilização de informações geográficas são semelhantes às da Google e, em sua maioria, proprietárias. Empresas como Nokia, Yahoo! e Garmin criaram soluções simples que podem fornecer serviços de roteamento e identificação de pontos de interesse nas proximidades do usuário. É interessante lembrar que a eficiência de tais produtos está diretamente ligada à constante atualização dos dados. No entanto, como essas empresas não têm o mapeamento como atividade principal, a dinâmica do dia-a-dia freqüentemente torna obsoletos os dados armazenados em seus servidores.

[1] http://maps.google.com.br/maps?hl=pt-BR&tab=wl

Outro aspecto importante ligado aos provedores proprietários refere-se à falta de integração entre as soluções. Não há nenhum tipo de padrão entre as informações armazenadas nos servidores da Google, Yahoo!, Nokia ou nos mapas fornecidos pela Garmin. Isso impede que o usuário combine informações disponíveis nesses sistemas. É possível apenas apresentar informação proveniente de outras fontes em superposição ao mapa básico fornecido por cada serviço, porém não se pode complementá-lo ou corrigi-lo. Apesar da aparente possibilidade de integração entre fontes distintas de dados geográficos, na prática isso não se verifica.

Nesse contexto, o *Open Geospatial Consortium* (OGC) propôs e formalizou um conjunto de serviços *Web*, destinados a promover a interoperabilidade entre clientes e servidores de dados geográficos. Um dos serviços propostos pela OGC, denominado *Web Map Service* (WMS) (OGC 2006) é usado para criar e transferir mapas a clientes, executados em navegadores ou *desktops*.

Basicamente, o WMS renderiza (criar uma imagem a partir de um conjuto de dados armazenados) um mapa para cada pedido recebido, segundo as configurações recebidas do cliente, que pode especificar as camadas, a área e o nível de zoom desejado. O WMS permite que os usuários combinem várias fontes de dados segundo sua necessidade.

A arquitetura proposta pelo OGC, da qual o WMS faz parte, é um tipo de arquitetura orientada por serviço (*Service-Oriented Architecture*, SOA), projetada de modo a viabilizar a integração simples entre plataformas heterogêneas. Elementos dessa arquitetura viabilizam o intercâmbio de dados geográficos com base em padrões. Por exemplo, o padrão GML (*Geography Markup Language)*, uma extensão da linguagem XML é usado para transferir objetos geográficos entre clientes e servidores OGC. Outro padrão estabelece a forma de codificação da geometria de objetos geográficos (*Simple Features Specification*, SFS), tanto de forma abstrata como para armazenamento em servidores de bancos de dados objeto-relacionais (SFS for SQL) (OGC 1998). Para fornecer aos clientes informações sobre os dados disponíveis, o WMS utiliza um serviço de catálogo segundo a especificação SOA, que tem a finalidade de publicar metadados descrevendo o conjunto de dados armazenados no servidor, métodos e formas para acessá-los.

A partir de então, clientes para *web browsers* como, por exemplo, o OpenLayers[2], desenvolvido em JavaScript (JS), podem acessar as informações contidas nos servidores sem conhecer seu conteúdo pois, ao acessarem o catálogo, podem obter informações para conexão, seleção de serviços, listagem de dados disponíveis e parâmetros da imagem que será transmitida (Tu and Abdelguerfi 2006; Davis Jr. and Alves 2007). É interessante salientar que o serviço de consulta ao catalogo é padrão e não precisa necessariamente estar hospedado no servidor de renderização.

No cenário móvel, ainda são muito poucas as iniciativas de desenvolvimento de clientes que sigam o padrão WMS, visto que este não foi desenvolvido originalmente para equipamentos tais como PDAs ou celulares. Celulares e PDAs são plataformas para uma mobilidade que só pode ser alcançada se esses aparelhos estiverem conectados vinte e quatro horas por dia. Para que isso seja possível, eles fazem uso das redes WiFi e, na grande maioria dos casos, da rede GPRS (2,5G ou 3G), que pode ainda não estar funcionando em sua plenitude. Essas redes se caracterizam por possuírem custo relativamente elevado e normalmente atrelado à banda consumida. Por esse motivo, soluções desenvolvidas para essas plataformas devem gerar o menor *overhead* possível e armazenar a maior quantidade de dados úteis em *cache*, além de poupar recursos de processamento, que estão intimamente ligados aos limites da bateria do equipamento.

Serviços como o WMS não foram originalmente concebidos para funcionar em ambiente móvel, o que fica evidente quando analisamos o custo inicial para transmissão do catálogo devido ao *overhead* imposto pelo padrão XML, no qual o catálogo é escrito, ou ao se constatar sua falta de preocupação em conservar recursos de armazenamento, comunicação e processamento. Com isso, uma aplicação direta das especificações do WMS, apesar de garantir flexibilidade na utilização e combinação de várias fontes diferentes fontes de dados geográficos, não seria adequada. Por exemplo, haveria em princípio uma tendência a grande consumo de volume de transmissão, visto que os servidores WMS não implementam navegabilidade para os mapas renderizados: para cada deslocamento na tela do cliente móvel, em qualquer direção, uma nova imagem precisa ser solicitada pelo cliente, criada e retransmitida pelo servidor. Caso esta ação seja realizada com

[2] http://openlayers.org/

freqüência, seriam gerados custos elevados para o usuário, devido ao grande consumo de banda, e também para o servidor, que teria sua capacidade de processamento ocupada com a criação de partes redundantes dos mapas, prejudicando sua escalabilidade.

Esta dissertação busca propor, implementar e analisar alternativas tecnológicas para a concepção e desenvolvimento de clientes WMS, e apresenta um estudo comparativo entre soluções existentes no mercado e as alternativas aqui propostas.

1.1 Objetivo geral

O objetivo deste trabalho é propor e implementar uma solução de conexão a serviços WMS voltada para equipamentos móveis, procurando respeitar suas limitações. O desempenho de um protótipo será medido, e os resultados obtidos serão comparados com implementações convencionais existentes.

1.2 Objetivos específicos

Visando obter informações que contribuam para a validação do estudo e para a disseminação de aplicações móveis capazes de lidar com grandes volumes de dados, têm-se como objetivos específicos:

1. Verificar a redução da banda necessária para que a utilização de um cliente WMS em equipamento móvel se torne mais viável, reduzindo o custo monetário de utilização e aumentando o tempo de vida da bateria;
2. Propor uma alternativa à implementação direta do padrão WMS, buscando a redução do *overhead* de comunicação, principalmente no momento da consulta ao catálogo e transmissão de imagens para exibição;
3. Implementar a proposta;
4. Buscar reduzir a carga de processamento imposta ao servidor WMS, possibilitando maior escalabilidade;
5. Analisar o desempenho da solução proposta sob as rede WiFi e GPRS 2,5 e 3G e comparar os resultados obtidos com outro cliente WMS móvel.

1.3 Metodologia

Para definir um modelo alternativo ao existente, realizarmos testes consistentes e medirmos a eficiência da proposta, quatro passos foram seguidos:

1. Pesquisa

Buscamos na literatura ferramentas para computadores móveis que implementassem o padrão WMS ou algum outro que também tivesse por finalidade prover mapas para plataformas móveis;

2. Análise

Analisamos o comportamento de cada alternativa encontrada levando em consideração a eficiência no consumo de banda, tempo de resposta para o usuário, possibilidade de integração com outras bases geográficas, e consumo de recursos do servidor;

3. Criação do Protótipo

De posse das informações coletadas, foi possível identificar quais os principais pontos fracos das soluções encontradas, e definir as inovações que deveriam ser inseridas no protótipo;

4. Comparação de Resultados

Após a implementação do protótipo, seu desempenho, bem como sua flexibilidade, foram comparados com as soluções encontradas para que fosse possível confirmar suas vantagens.

1.4 Estrutura do trabalho

Esta dissertação está organizada da seguinte forma. O Capítulo 2 apresenta os trabalhos relacionados, apresenta as soluções atuais e suas limitações. O Capítulo 3 apresenta a solução proposta. O Capítulo 4 mostra os resultados obtidos e mostra a comparação do protótipo desenvolvido com um cliente WMS puro, e por fim, no Capítulo 5, apresentamos as conclusões e trabalhos futuros.

2. TRABALHOS RELACIONADOS

Esta dissertação se insere na área de infraestruturas de dados espaciais (IDE). O seu principal objetivo é prover meios adequados para acesso remoto e interoperável a dados geográficos disponíveis *online* usando equipamentos móveis. As próximas seções apresentam trabalhos relacionados a esse objetivo, inicialmente apresentando alguns conceitos a respeito de IDE, e em seguida, detalhando a fundamentação que apóia a implementação de IDE que usam arquiteturas orientadas por serviços, dentro da padronização proposta pelo *Open Geospatial Consortium*.

2.1 Infra-Estruturas de Dados Espaciais

A expressão original *Spatial Data Infrastructure* (SDI, posteriormente adaptada para o português como infraestrutura de dados espaciais, IDE) foi usada inicialmente para descrever recursos de acesso padronizado a dados geográficos (Maguire and Longley 2005). Segundo (Rajabifard, Williamson et al. 2000), a IDE pode ser vista como uma ferramenta onde usuários podem encontrar e utilizar facilmente informações sobre certo conjunto de dados espaciais, além de interagir com o provedor de informações para a manutenção dos dados. Uma IDE também implica na existência de algum conjunto de padrões para formulação de políticas e implementações, fornecendo metadados padronizados e completos, possibilitando acesso *online* a dados espaciais. A primeira geração de IDE foi desenvolvida visando garantir um amplo escopo temático, fundamental para fomentar o desenvolvimento econômico através do provimento de acesso público para múltiplos bens ou serviços.

IDEs são vistas como uma nova e interoperável alternativa para criação, distribuição e uso de informações geográficas (Davis Jr. 2008; Fonseca 2008). Além de distribuir mapas, dissemina dados espaciais relacionados a metadados que adicionam semântica às informações disponíveis. Dessa forma, o usuário tem maior facilidade para criar novas informações a partir da combinação dos dados obtidos. Sob este ponto de vista, IDEs podem ser importantes para o gerenciamento e crescimento sustentável da nossa sociedade (Davis Jr, Fonseca et al. 2009) , pois

fornece um conjunto de ferramentas capaz de auxiliar o estudo do comportamento da sociedade sob o ponto de vista geográfico.

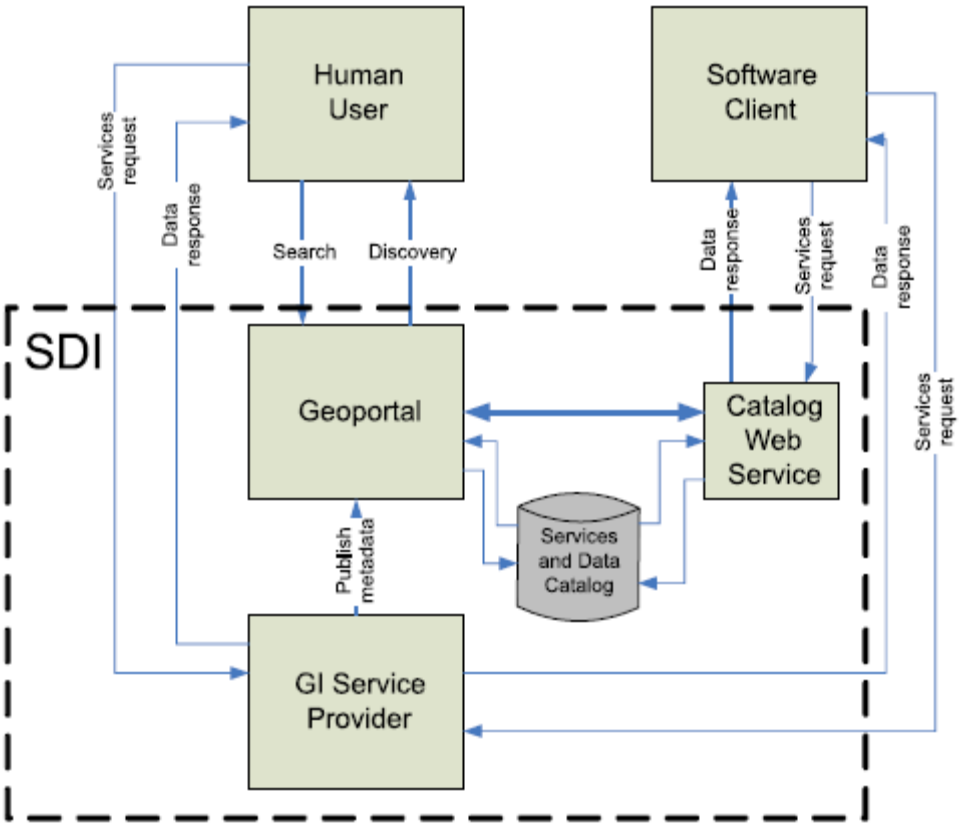

FIGURA 2.1: **Geoportais e SDI (Davis and Alves 2006)**

A FIGURA 2.1: Geoportais e SDI (Davis and Alves 2006)FIGURA 2.1 ilustra a arquitetura da IDE montada sobre a plataforma de SOA (*Service Oriented Architectures*), onde o *GI Service* representa o provedor de informações, o *Catalog Web Service* fornece a ferramenta para consulta dos serviços disponíveis, permitindo que outros *softwares* também utilizem o sistema, enquanto o *Geoportal* proporciona interação com usuários humanos. Essa estrutura permite a criação de serviços com fraco acoplamento (Oliveira, Davis et al. 2008). Na FIGURA 2.1, é possível perceber que usuários e aplicações compartilham os mesmos dados geográficos remotamente através de chamadas padronizadas.

A evolução da primeira geração de IDE ocorreu graças à recente expansão dos sistemas de informação baseados na *Web*. Nos Estados Unidos, o portal Geospatial One-Stop[3] (GOS) foi criado para fornecer acesso a informações

[3] http://gos2.geodata.gov/wps/portal/gos

geográficas, inaugurando o conceito de *geoportais* (Maguire and Longley 2005; Tait 2005), que, diferentemente das IDE, fornecem algum nível de acesso interativo e ferramentas de descoberta para os usuários. De fato, atualmente geoportais são vistos como componentes de IDEs, cuja finalidade é garantir acesso interativo para descoberta de fontes e exploração do conteúdo existente.

Enquanto as primeiras gerações de IDE eram pouco mais do que repositórios de arquivos para *download*, atualmente está em curso a implementação de recursos visando o acesso *online* aos dados e também a execução de tarefas. A reestruturação do modelo IDE permitirá geração de produtos mais complexos pelo encadeamento entre recursos de acesso aos dados e de processamento.

A implementação desse tipo de visão é viabilizada pelas arquiteturas orientadas a serviços, mais conhecidas como SOA. Arquiteturas SOA são viabilizadas tecnicamente pela criação de padrões para a descrição de dados e de operações, além da definição de serviços de descoberta, seleção e vinculação de informações. Existem dois grandes grupos de padrões de serviços para arquiteturas SOA com interesse para aplicações de geoinformática. O primeiro grupo, que será apresentado e discutido a seguir, teve sua padronização proposta e implementada pelo OGC. O segundo inclui a definição mais usual de serviço *web*, definida e padronizada pelo *World Wide Web Consortium* (W3C). A seguir, concentraremos nosso foco na apresentação de serviços e padrões OGC.

2.2 O *Open Geospatial Consortium* e Padrões para IDE

Em 1994 foi criado o OGC, composto por mais de 250 companhias, agências governamentais e universidades. O OGC foi criado para promover o desenvolvimento de tecnologias que facilitassem a interoperabilidade entre sistemas envolvendo informação espacial de localização (Davis Jr. 2005), sendo responsável por definir padrões abertos para a utilização de dados geográficos. Em 1998, um acordo entre OGC e a *International Organization for Standardization* (ISO) foi firmado, visando cooperação entre as duas organizações (OpenGEO 2007), resultando na transformação de padrões OGC em padrões internacionais ISO.

Os padrões definidos pela OGC e ratificados pela ISO incluem a definição de componentes tecnológicos como:

- SFS (*Simple Features Specification* ou Especificação de Feições Simples): Definições baseadas no padrão SQL para armazenamento, leitura e

atualização de feições geográficas em 2D com interpolação linear entre os vértices. (OpenGEO 2007);

- GML (*Geography Markup Language* ou Linguagem Geográfica de Marcação): Uma extensão da linguagem XML usada para transferir objetos geográficos entre clientes e servidores OGC (OpenGEO 2007);
- WFS (*Web Feature Service* ou Serviço de Feições Web): Definição de métodos de inserção, atualização e exclusão baseada no ambiente Web/HTTP baseados em GML (OpenGEO 2007);
- WMS (*Web Map Service* ou Serviço de Mapas Web): Define quatro métodos (*GetCapabilities*, *GetMap*, *GetFeatureInfo* e *DescribeLayer*) para acesso a dados geográficos baseados em vetores ou imagens. Essa especificação não permite alteração de conteúdo e todo o processo de renderização é realizado pelo servidor (OpenGEO 2007);
- WCS (*Web Coverage Service* ou Serviço de Cobertura Web): Define três métodos (*GetCapabilities*, *DescribeCoverage* e *GetCoverage*) para acesso a dados geográficos. Ao contrario do WMS, essa especificação define que a renderização dos mapas é feita pelo cliente (OpenGEO 2007).

2.3 *Web Map Service*

O *Web Map Service* (WMS) foi padronizado pela OGC no ano 2000. A versão WMS 1.1.1, liberada em 2002, é a versão mais suportada, sendo que a mais recente é a 1.3 (Open Geospatial Consortium (OGC), 2006). A FIGURA 2.2 apresenta o esquema de comunicação entre o cliente e o servidor WMS. O lado esquerdo da figura, que representa o cliente, lista cada requisição individualmente, sendo que essas requisições serão detalhadas a seguir.

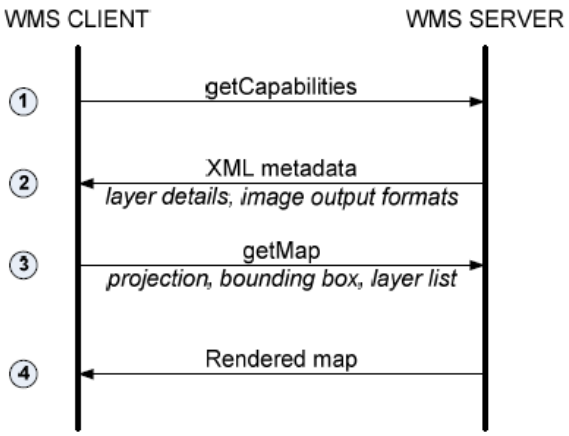

FIGURA 2.2: Comunicações Web Map Service

No WMS, assim como em outras especificações OGC baseadas em SOA, os serviços e os parâmetros podem ser descobertos usando-se uma requisição padrão. Assumindo-se que o cliente não conhece inicialmente os dados que ele pode acessar, sua interação inicial consiste em recuperar os serviços disponíveis usando o método *getCapabilities* (1). Como resposta, o servidor gera um documento XML, que contém detalhes de cada camada disponível.

Os metadados fornecidos pelo XML incluem, para cada camada disponível, título, resumo, palavras chave, informações de contato técnico ou com a instituição responsável pelo serviço e outros dados (2). Existem também informações sobre o formato em que cada dado pode ser fornecido pelo servidor, incluindo formatos tradicionais de imagem, como o JPEG ou PNG, e, dependendo da infra-estrutura do serviço, KML, SVG e XML. Uma vez que os serviços são conhecidos, o cliente pode requisitar a renderização do mapa usando o método *getMap* (3). Ele é invocado usando a requisição *Hypertext Transfer Protocol* (HTTP) *Get* tradicional, em que o cliente passa alguns parâmetros, tais como o sistema de projeção, coordenadas do retângulo envolvente, lista de camadas que serão visualizadas, e formato da imagem retornada. O servidor, então, cria e transmite de volta o mapa requisitado pelo cliente (4).

Para ilustrar, de forma prática, o funcionamento de um WMS, apresentaremos, a seguir, os passos e os resultados da consulta a um servidor com dados nacionais hospedado em `http://mapas.mma.gov.br`. Inicialmente, chamaremos o método *getCapabilities* para obtermos os metadados sobre o serviço pela url

http://mapas.mma.gov.br/cgi-bim/mapserv

```
?map=/opt/www/html/webservices/biorregioes.map
&request=GetCapabilities
&service=wms
&version=1.1.1.
```

Como resposta, recebemos o seguinte XML que poderá ser encontrado no Anexo I, embora não tenha sido adicionado na integra nesta dissertação. Seu tamanho total é de 11,3 KB, contendo informações sobre como executar o *getMap*, quais camadas (layers) estão disponíveis além de suas propriedades.

De posse das metainformações e agora conhecendo as camadas “Biomas do Brasil”, “Limites estaduais do Brasil”, “Revisão áreas prioritárias para conservação da biodiversidade (importância biológica)-2007” e “Revisão áreas prioritárias para conservação da biodiversidade (prioridade de ação)-2007”, Executaremos o *getMap* pela seguinte url, montada apenas com informações obtidas com *getCapabilities*:

```
http://mapas.mma.gov.br/cgi-bin/mapserv
?map=/opt/www/html/webservices/biorregioes.map
&request=GetMap
&styles=
&layers=biomas
&bbox=-76.5126,-36.9484,-25.5852,7.04601
&SRS=EPSG:4291
&TRANSPARENT=TRUE
&width=640&height=480
&format=image/png&version=1.0.
```

Como resultado, o servidor WMS no envia a imagem da FIGURA 2.3.

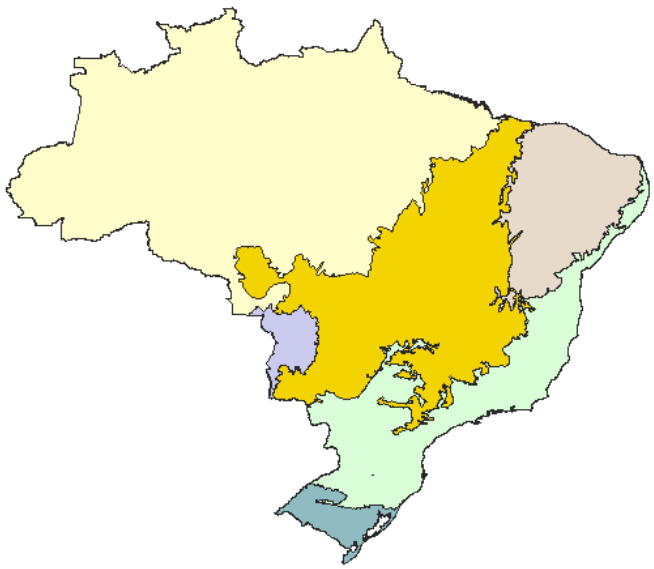

FIGURA 2.3: Imagem de 8,86 KB obtida pelo método *getMap*.

No passo seguinte (FIGURA 2.4), será executado um *pan* igual a metade da largura da imagem para a esquerda. A url usada será:

```
http://mapas.mma.gov.br/cgi-bin/mapserv
?map=/opt/www/html/webservices/biorregioes.map
&request=GetMap
&styles=
&layers=biomas
&bbox=-51.0489,-36.9484,-25.5852,7.04601
&SRS=EPSG:4291
&TRANSPARENT=FALSE
&width=320&height=480
&format=image/png
&version=1.0.
```

Para finalizar o exemplo (FIGURA 2.5), será executado um *zoom* para aumentar o tamanho da imagem em duas vezes. A url usada será: http://mapas.mma.gov.br/cgi-bin/mapserv?map=/opt/www/html/webservices/biorregioes.map&request=GetMap&styles=&layers=biomas&bbox=-51.0489,-36.9484,-25.5852,7.04601&SRS=EPSG:4291&TRANSPARENT=FALSE&width=452&height=678&format=image/png&version=1.0.

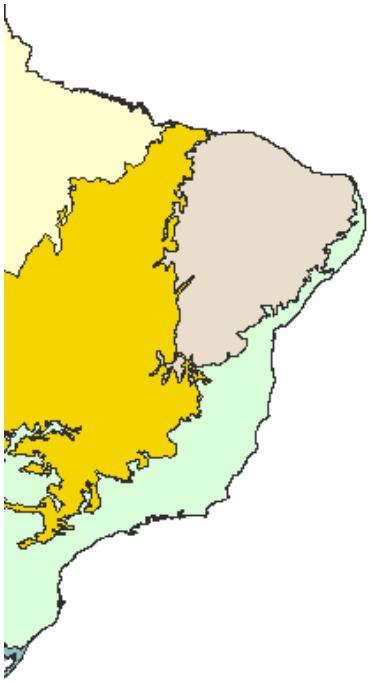

FIGURA 2.4: Imagem de 4,98 KB obtida pelo método *getMap* após *pan* para a esquerda.

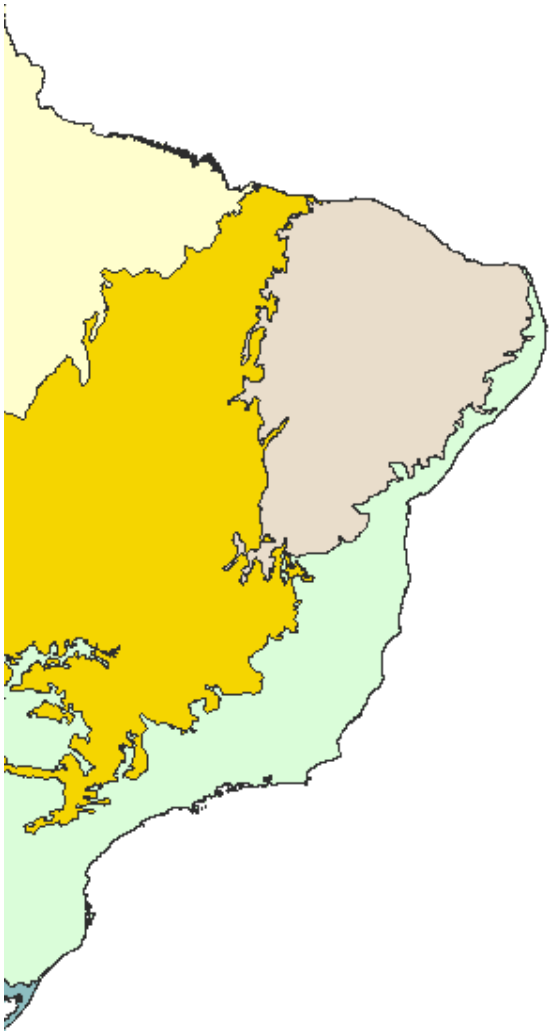

FIGURA 2.5: Imagem de 7,65 KB obtida pelo método *getMap* após *zoom* para aumento de duas vezes.

Clientes WMS têm a liberdade de selecionar camadas para visualização além fornecerem um meio de compará-las camadas como necessário, assim como selecionar escalas específicas de visualização e personalizar o tamanho dos mapas retornados de modo que estejam adequados à interface do usuário. Os clientes normalmente implementam interfaces que são similares a outras encontradas em GIS *desktop*, permitindo ao usuário ativar e desativar as camadas disponíveis. Além disso, os clientes permitem que os usuários se conectem simultaneamente a vários serviços diferentes, compondo mapas com elementos de diferentes provedores de dados geográficos.

2.4 Servidores WMS Existentes

A partir dos padrões definidos pelo OGC e ISO, serviços e aplicativos foram desenvolvidos para facilitar a utilização de dados geográficos na web. Na categoria Servidores WMS, podemos citar o MapServer[4] como uma ferramenta para criação e publicação de mapas na Internet.

O MapServer é uma ferramenta baseada em CGI (*Common Gateway Interface*), que depende de um servidor web, como o Apache[5], para que possa funcionar corretamente. Com o MapServer, é possível criar imagens a partir de dados armazenados em bancos de dados geográficos, imagens raster ou vetores. Como ferramenta de código aberto, ele pode ser personalizado através das linguagens PHP, C, Java e C# (MapServer 2008). Quanto à configuração de apresentação de renderização de mapas, esta é baseada em arquivos textos padronizados que podem ser editados manualmente para a definição das camadas disponíveis para consulta e suas propriedades.

O GeoServer[6] é um servidor baseado em J2EE, possui controle transacional para o WFS e um ambiente de configuração amigável baseado na Web (GeoServer 2008). Ao contrario do MapServer, ele pode funcionar sem o auxilio de outra aplicação, excluindo a integração com um Sistema Gerenciador de Banco de Dados Geográfico (SGDBG), necessário também ao MapServer. O GeoServer possui um cliente WMS integrado e, ao contrário do MapServer, possui uma aplicação responsável por configurar as camadas disponíveis para consulta, eliminando, assim, a necessidade do usuário conhecer a estrutura dos arquivos de configuração. Assim como o MapServer, o GeoServer também é uma ferramenta de código aberto, porém possui apenas o Java como ferramenta de personalização.

2.5 Clientes WMS Existentes

Para acessar as informações provenientes dos serviços WMS, é necessário o uso de clientes que consigam interagir com os servidores usando o padrão. Diversos clientes estão disponíveis atualmente, entre software livre e proprietário. Dentre eles, podemos citar OpenLayers[7], gvSIG[8] e Quantum GIS[9].

---

[4] http://mapserver.org/
[5] http://www.apache.org/
[6] http://geoserver.org/display/GEOS/Welcome
[7] http://openlayers.org/
[8] http://www.gvsig.gva.es/

O OpenLayers pode ser visto como um componente JavaScript, sendo o único, dentre os clientes WMS analisados, que foi desenvolvido para o ambiente Web. O OpenLayers é capaz de receber e manipular as imagens geradas por um servidor WMS, além de armazená-las em *cache* para consultas futuras. Os objetos do OpenLayers podem ser adicionados através de comandos *HyperText Markup Language* (HTML) na página desejada. Por implementar os principais padrões OGC, o OpenLayers foi incluído nas distribuições do GeoServer (Uchoa 2008).

O gvSIG é um cliente *desktop,* desenvolvido em Java, que é destinado à gestão de informações geográficas. Possui interface amigável, é capaz de trabalhar com imagens raster e vetoriais, além de implementar os padrões WMS, WFS e WCS. O gvSig é uma ferramenta de código aberto, o que permite aos desenvolvedores criar novas características a partir de suas bibliotecas.

Por fim, o Quantum GIS (qGIS) é uma ferramenta que permite criação, edição e exportação de mapas geográficos em formatos como GPX. O qGIS suporta dados em formato vetorial e raster, arquivos do tipo ESRI shapefiles, MapInfo, SDTS e GML, além de implementar o padrão OGC nas especificações WMS e WFS.

As Figuras 2.6, 2.7 e 2.8 ilustram, respectivamente o OpenLayes, o gvSig e o qGis. Nelas, podemos observar alguns controle e representações gráficas de dados geográficos obtidos de servidores WMS.

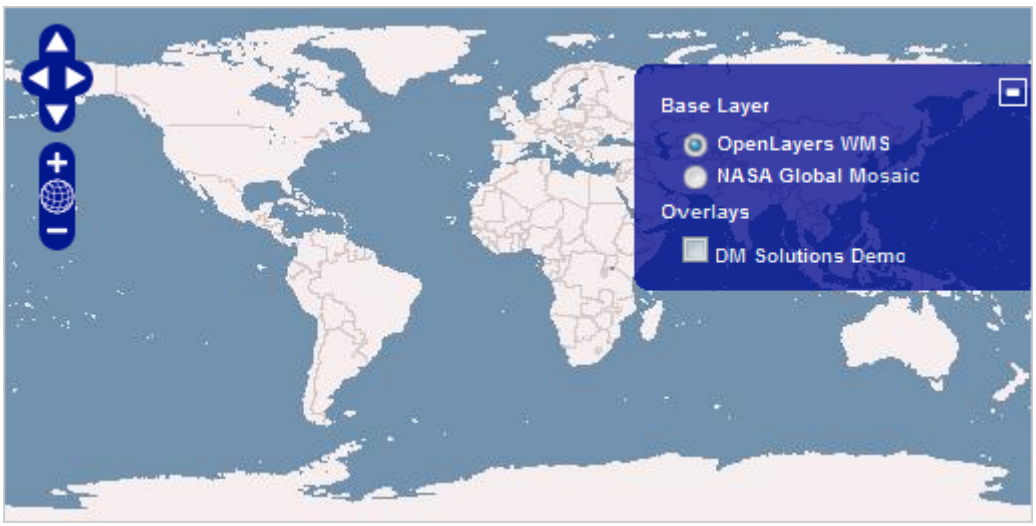

FIGURA 2.6: Imagem do OpenLayers apresentando controles de navegação e seleção de camadas (OpenLayer.org 2009).

[9] http://www.qgis.org/

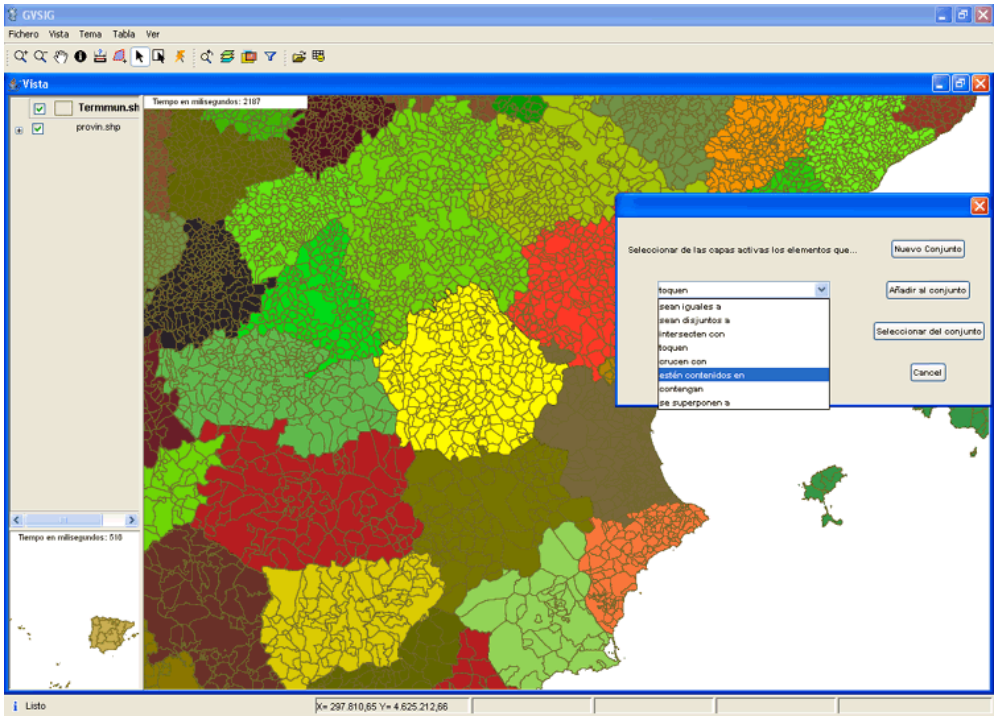

FIGURA 2.7: Imagem do gvSig apresentando controles de navegação e seleção de camadas (gvSIG 2009).

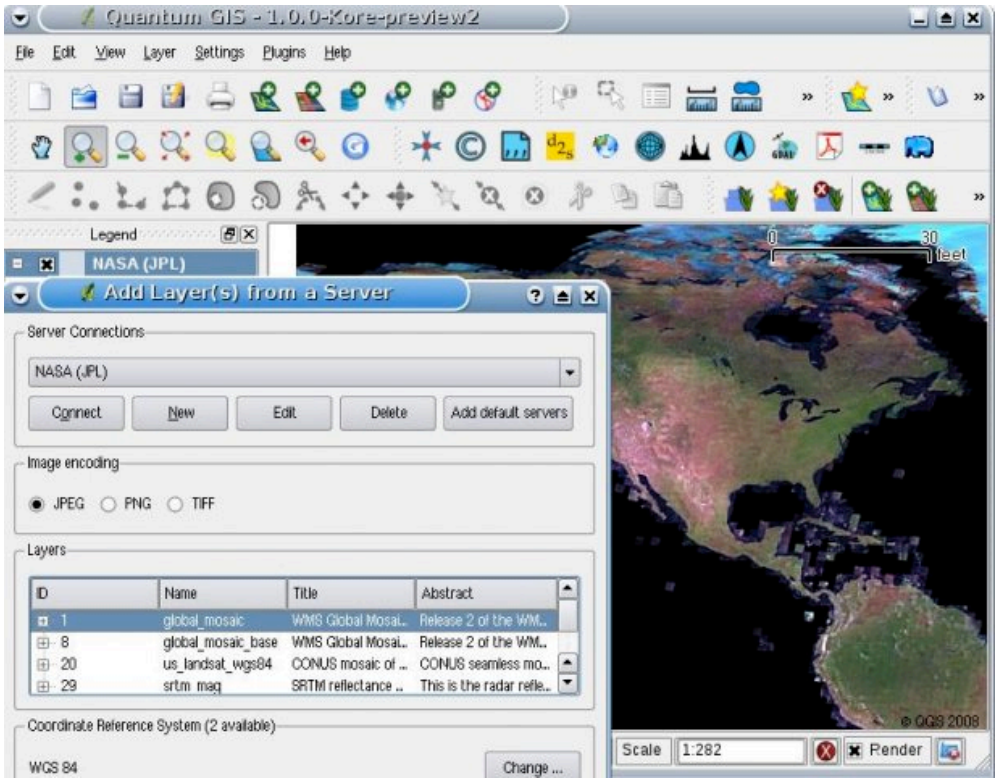

FIGURA 2.8: Imagem do qGIS apresentando controles de navegação, seleção de camadas e ferramentas de edição (Quantun GIS 2009).

Os clientes OpenLayers, gvSig e qGis são de código aberto e compatíveis com sistemas Windows e Linux.

No âmbito móvel, foram encontrados dois clientes WMS. Um deles, desenvolvido pela gvSIG para a plataforma Windows CE[10]. Ele é capaz de trabalhar com arquivos do tipo shapefile, EWC e WMS, além de possuir integração com GPS. A desvantagem do gvSIG Mobile está em sua reduzida compatibilidade com plataformas móveis.

O segundo cliente encontrado é o WMS Client desenvolvido pela Skylab[11]. Desenvolvido em Java, ele possui ampla compatibilidade como dispositivos móveis, sendo possível utilizá-lo em vários modelos de PDAs e celulares. Em ambas as soluções, o usuário pode selecionar a fonte de dados que lhe for conveniente, assim como nos clientes para *desktop.*

FIGURA 2.9: Client WMS Skylab

2.6 Banco de Dados

Para que seja possível ao servidor WMS renderizar os mapas solicitados pelos clientes, de forma eficiente, é necessário ele que este esteja integrado a um Sistema Gerenciador de Bancos de Dados (SGBD) capaz de lidar com pesquisas que envolvem índices espaciais.

O tratamento de dados espaciais por um SGBD é mais complexo e trabalhoso do que o exigido por dados comuns. O SGBD precisa ter suporte a tipos especiais

[10] http://msdn.microsoft.com/en-us/embedded/aa731407.aspx

[11] http://www.skylab-mobilesystems.com/en/products/j2me_wms_client.html

de dados e dispor de funções espaciais. Os índices, envolvem estruturas de dados voltadas para objetos geográficos (Ravada 1999).

Atualmente, dois SGBD's com as características necessárias para o gerenciamento de dados espaciais se destacam: o PostgreSQL[12], com uma extensão denominada PostGIS[13], e o Oracle.

O PostgreSQL é um SGBD objeto-relacional com ótima performance, mesmo para grandes volumes de dados (OpenGEO 2007). No entanto, para que possa fazer o armazenamento de objetos geográficos de forma eficiente, é necessária a instalação do módulo PostGIS. O PostGIS é responsável pelos acréscimos ao PostGreSQL que o tornam capaz de receber dados geográficos. Ele permite que objetos GIS sejam armazenados e recuperados no formato especificado pelo OGC (padrão *Simple Features Specifications*), e possui suporte aos índices espaciais GiST e R-Tree (Refractions Research 2008).

Ao contrario do PostgreSQL, que necessita do PostGIS para dar suporte completo a dados geográficos, o Oracle[14] possui em sua distribuição mais completa (denominada *Enterprise*) o Oracle Spatial[15], que é responsável por gerenciar informações geográficas. Nas versões mais básicas, como a XE (eXpress Edition[16]), está disponível o Oracle Locator.

Assim como o PostGIS, o Oracle permite a realização de consultas utilizando a tradicional *Structured Query Language* (SQL), que é complementada com funções voltadas especificamente para dados espaciais (Ravada 1999).

2.7 Limitações das plataformas

Embora serviços web forneçam maior padronização para interoperabilidade e tenham permitido maior dinamismo na utilização e combinação de informações, eles também são custosos quando observamos o consumo de processamento para renderizar os mapas e largura de banda necessária para transferência das imagens.

Os clientes apresentados anteriormente possuem algumas soluções para poupar recursos do servidor e, conseqüentemente, melhorar o tempo de resposta

[12] http://www.postgresql.org.br/
[13] http://postgis.refractions.net/
[14] http://www.oracle.com/global/br/index.html
[15] http://www.oracle.com/technology/products/spatial/index.html
[16] http://www.oracle.com/technology/products/database/xe/index.html

para o usuário. Pode-se citar funções de *cache*, que reduzem a necessidade de banda e o tempo de resposta, permitindo ao usuário trabalhar com maior eficiência.

Porém, a implementação direta do padrão não prevê navegabilidade para os mapas renderizados, o que força o servidor a criar uma nova imagem a cada movimentação realizada pelo usuário, tanto em dispositivos móveis quanto em *desktops*. As únicas restrições apresentadas na solução WMS são aquelas inerentes aos dados que serão recuperados como, por exemplo, *zoom* máximo ou dimensões dos retângulos envolventes.

A constante geração de mapas e a transmissão de mapas adicionam outro problema quando pensamos em escalabilidade para os servidores WMS: o consumo de banda, pois a transmissão de imagens é um processo custoso para qualquer tipo de rede. Por fim, podemos dizer que a constante renderização de mapas melhoraria a performance do servidor em relação ao consumo de memória e tempo de disco.

A cada requisição a um servidor WMS, novas consultas a bancos de dados geográficos são realizadas. A pesquisa por entidades geográficas envolvem a utilização de índices especificamente desenvolvidos para lidar com esse tipo de informação, comohttp://www.megaupload.com/pt/?d=W2SEX0Y9 B-Tree e R-Tree, sendo que a pesquisa por índices espaciais exige mais processamento e memória do que a pesquisa por índices convencionais. O WMS foi construído sobre o conceito de serviços Web, que por sua vez são construídos normalmente sobre padrões abertos e protocolos de transporte (baseados principalmente em XML) para troca de dados com os clientes, formando uma arquitetura de sistemas de informação com baixo acoplamento (Ferris and Farrell 2003; Alves and Davis Jr 2007).

Como poderá ser visto nas próximas seções, a utilização de XML para transporte de dados e protocolos, apesar de adicionar valor semântico às informações, cria um grande *overhead* na comunicação. Serviços *Web* também usam o HTTP na camada de aplicação (W3C 2002) para permitir que o cliente realize chamadas síncronas, o que também pode se tornar um problema, principalmente quando a aplicação estiver sendo suportada por redes sem fio. Isso porque o atraso inerente a esse tipo de rede devido à repetição no envio de pacotes pode criar atrasos consideráveis no tempo de resposta das transações e chamadas.

Existem outras limitações na computação móvel e redes sem fio que precisam ser listadas. Como as redes sem fio estão se tornando populares, a demanda por

elas também cresce. A qualidade do serviço, QoS, é um conceito relacionado a performance da rede e à satisfação do usuário. Parâmetros de QoS tais como *throughput*, atraso, *jitter* (variação estatística do retardo de dados na rede), disponibilidade e confiabilidade são alguns dos indicadores usados para avaliar a qualidade da rede para suporte a aplicações (Duarte-Figueiredo and Loureiro 2007). Redes sem fio têm algumas limitações em garantir QoS fim-a-fim para usuários móveis, incluindo falhas de comunicação quando o usuário está se movendo, o que obrigaria a retransmissão de pacotes aumentando o tráfego na rede, e variação no consumo de energia (Ribeiro, Duarte-Figueiredo et al. 2008). Juntamente com o QoS da rede, outras limitações da computação móvel são importantes para nossa análise, incluindo:

1. Insuficiência de banda, que se refere tanto à velocidade quando à qualidade do sinal para a transferência;
2. Limitação de memória local para armazenamento. Apesar de memória ser um recurso barato nos dias de hoje, computadores móveis ainda possuem limitação nesse aspecto, principalmente para armazenamento de dados multimídia;
3. Consumo de energia envolvido no gerenciamento da rede, interpretação dos dados recebidos do servidor e exibição dos mapas;
4. Interferência nas transmissões e interrupções gerando retransmissão de pacotes e aumentando o consumo de banda;
5. Segurança;
6. Regras de interface humana (Forman and Zahorjan 2004).

Atualmente, em se tratando de WMS, é importante enfatizar que, em muitas interações entre o cliente WMS e o servidor, são processadas imagens que não fornecem ao usuário nenhuma informação nova, dessa forma desperdiçando processamento, memória e largura de banda no servidor, prejudicando, assim, sua escalabilidade. Se o usuário decide realizar um *pan* (movimento da imagem na tela) parcial na imagem em alguma direção, um pedido é enviado ao servidor como uma requisição inteiramente nova, e uma nova imagem é criada e transmitida.

Este método contrasta fortemente com as soluções comerciais de navegação de mapas, como Google Maps, que mantêm partes do mapa pré-renderizadas em várias (e fixas) escalas, transmitindo apenas as partes que são pedidas pelo usuário. Intuitivamente, pode-se afirmar que a abordagem WMS demanda muito mais do

servidor. Assim, serviços WMS não são tão escaláveis quanto as soluções comerciais. Por outro lado, as soluções comerciais não permitem aos clientes selecionar e comparar camadas segundo sua própria necessidade.

Nossa proposta, apresentada no próximo capítulo, pretende encontrar um meio termo entre o uso de mapas online para navegadores (*web map browsers* como o GoogleMaps) e serviços WMS, usando uma aplicação servidora intermediária. Essa aplicação intermediária limita, em parte, a liberdade do cliente WMS, quanto a possibilidades de configuração de parâmetros de saída do serviço. Porém, apresenta maior escalabilidade e ajuda a economizar em fatores de custo relevante para a plataforma móvel.

3. AMBIENTE WMS PARA APLICAÇÕES MÓVEIS

3.1 Arquitetura da Solução

Como exposto anteriormente, existem diversos aplicativos para aparelhos móveis capazes de interagir com seus respectivos servidores de mapas. Dentre eles, podemos citar os desenvolvidos pela Google, Yahoo!, Nokia e Garmin. Esses aplicativos comunicam-se apenas com seus próprios servidores.

Para fornecer ao usuário flexibilidade na escolha e combinação de informações geográficas, a opção seria o padrão WMS. Porém, foram encontrados apenas dois clientes WMS para dispositivos móveis, descritos na seção 2.5. O primeiro, desenvolvido pelo gvSIG é suportado apenas pela plataforma Windows CE, o que restringe muito sua compatibilidade e, por isso, não foi tratado nesta dissertação. O segundo, desenvolvido pela Skylab[17], além de seguir a especificação OGC (Mobilesystems 2005), possui ampla compatibilidade, visto que foi desenvolvido em Java 2 *Micro Edition* (J2ME), linguagem que oferece um ambiente robusto e flexível para o desenvolvimento de aplicativos para celulares e PDAs (Sun Microsystems 2008), garantindo ampla gama de dispositivos compatíveis. A natureza das informações envolvidas numa transação WMS envolve volume de dados elevado, tanto quando se refere aos dados processados pelo servidor, quanto aos enviados para o cliente em forma de imagens ou metadados.

Quando se fala em ambientes de rede sem fio, freqüentemente o volume elevado de dados está ligado a falhas de comunicação e aumento no tempo de resposta. Nossa abordagem propõe dividir o cliente WMS em dois componentes: cliente móvel (CM), e a camada WMS de conectividade (CWC). O CM é responsável pela interação com o usuário, incluindo visualização de dados. O CWC recebe as requisições do CM, criando uma interface com o servidor WMS. Apenas o CWC implementa funções de acordo com o padrão WMS; dessa forma, o servidor WMS entende que o pedido vem de um cliente WMS padrão. Entre o cliente móvel e a camada WMS de conectividade foi criada uma conexão baseada em *socket*. A conexão por *socket* é muito mais leve para o computador móvel, por isso economiza banda, memória local, recursos de processamento e bateria. A subtração das rotinas de comunicação com *Web Services* ajuda a manter o CM pequeno, ocupando

[17] http://www.skylab-mobilesystems.com/en/products/j2me_wms_client.html

menos espaço na memória principal do computador móvel, permitindo a criação de *cache* para alguns dos dados transmitidos. O cliente móvel foi desenvolvido em J2ME, proporcionando ampla compatibilidade com telefones móveis e PDAs, inclusive para aparelhos com sistema Symbian. O CWC, também desenvolvido em Java, roda em um computador desktop comum, agindo como um cliente totalmente compatível com WMS, embora não possua camadas de visualização ou interação com o usuário.

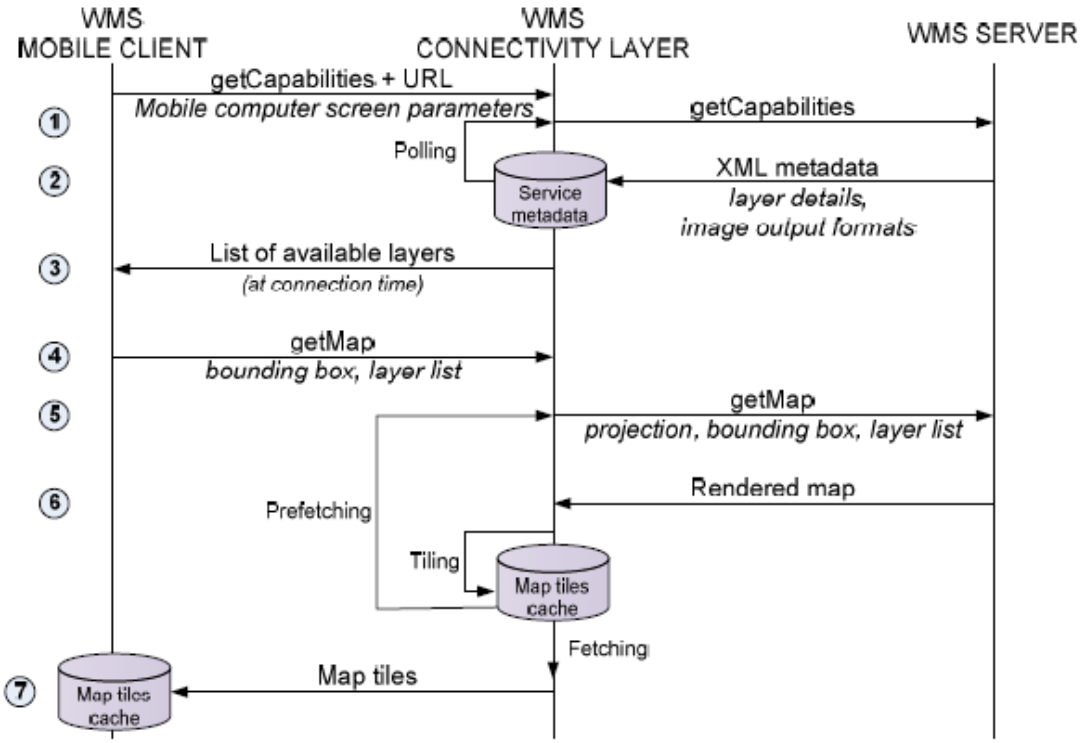

FIGURA 3.1: Ambiente WMS proposto

A FIGURA 3.1 apresenta um diagrama de interação, em que o protocolo de comunicação da nossa solução é detalhado. Os passos 1 e 2 mostram realização a mesma interação para o *getCapabilities* apresentado na FIGURA 2.2. Quando o computador móvel inicia o processo, ele envia uma URL para a requisição *getCapabilities*, juntamente com os parâmetros da tela (tamanho e resolução). Os resultados são, então, mantidos no CWC e parcialmente transmitidos para o cliente móvel de forma mais compacta (passo 3).

Nossos testes iniciais mostraram que esse simples *cache* de informações de camadas do servidor WMS pode poupar, ao cliente móvel, grande parte dos dados

que seriam transmitidos, especialmente quando o servidor possui dados de muitas camadas.

O passo 4 corresponde à requisição do mapa, que é enviada pelo cliente móvel de forma compacta e traduzida pelo CWC em uma requisição WMS completa (passo 5). A informação de projeção é mantida no CWC como padrão para evitar redundância na transmissão. O mapa é renderizado pelo servidor e retornado ao CWC (passo 6). No CWC, a imagem enviada pelo servidor WMS é dividida em pequenos pedaços, que são mais adequados para transmissão e exibição no CM.

Felizmente, os *tiles* (pedaços resultantes da divisão da imagem criada pelo servidor WMS) renderizados podem ser armazenados no CWC para evitar computação redundante no servidor WMS. Ainda para evitar computação redundante, a área de visualização enviada na requisição *getMap* pelo CM e expandida pelo CWC para que uma área maior esteja disponível para a divisão e posterior transmissão para o computador móvel. Nos testes realizados neste trabalho, esta expansão foi de duas vezes o tamanho original do retângulo envolvente. Isto é feito como uma estratégia de pré-processamento. Finalmente, as partes do mapa são transmitidas para o cliente (passo 7). Existe uma memória *cache* para partes da imagem tanto no CWC, quanto no CM, embora o último esteja sujeito a limitações de memória. Para evitar o estouro de memória nos computadores móveis, adotamos uma regra de invalidação de *cache* em que o *tile* mais distante do centro da tela é descartado para que a memória seja liberada para outras partes do mapa.

Ainda buscando reduzir o tráfego e o consumo de recursos do aparelho, o conjunto de ações disponíveis para o usuário foi reduzido ao mínimo necessário para realização de trabalhos relacionados a SDI. Assim, o CM pode realizar quatro ações: *zoom in* (aproximação da imagem), *zoom out* (afastamento da imagem), *pan* (deslocamento da imagem para qualquer direção) e mudança das camadas exibidas. Caso haja geração de novas imagens no servidor, o novo processo será equivalente a reiniciar os passos 3 e 4. Se o usuário quiser um *zoom in ou zoom out*, o processo retorna ao passo 4 e o servidor tem que gerar uma nova imagem.

No caso do pan, as partes do mapa em *cachê,* que forem necessárias para preencher a tela, serão enviadas para o cliente sem qualquer custo para o servidor WMS. Se a ação de *pan* for além da área pré-processada, um novo retângulo envolvente é enviado para o servidor e o processo é reiniciado.

Algumas ferramentas de mapeamento comercias para computadores móveis incluem uma função que permite ao usuário pré-carregar partes do mapa de servidores como Google Maps ou OpenStreetMap e armazenar na memória dos computadores móveis, para permitir a visualização mesmo em casos de falta de conexão de rede.

Nossa estratégia também possui esse recurso, além de permitir ao utilizador alterar a camada selecionada, quando uma conexão estiver disponível e, geralmente, determinar alguns parâmetros de visualização. Nossa solução também preserva a possibilidade de buscar dados de outros servidores OGC, combinando visualização de dados de diferentes fontes ao mesmo tempo.

Para aumentar a escalabilidade da nossa implementação e aumentar o uso da memória *cache*, nós estabelecemos alguns níveis fixos de *zoom*, e estabelecemos passos mínimos para o *pan*. Futuramente, incluiremos estratégias de agrupamento de camadas que permitam respostas menores para o *getCapabilities*, e uma seleção de camadas mais fácil. Essa estratégia de agrupamento também irá beneficiar o cache de mapas, visto que serão indexados também pela combinação de camadas da imagem renderizada. Atualmente, qualquer alteração na lista de camadas gera uma nova renderização de mapa.

3.2 Detalhamento da Solução

Como apresentado anteriormente, a solução proposta neste trabalho foi criada a partir da divisão do cliente WMS em dois componentes: O Cliente Móvel e a Camada de WMS de Conectividade.

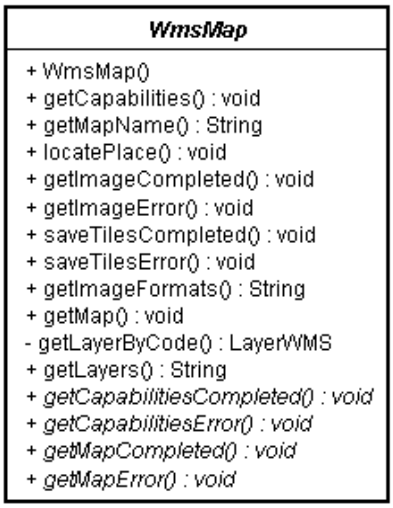

FIGURA 3.2: Classe WMSMap utilizado no Cliente Móvel

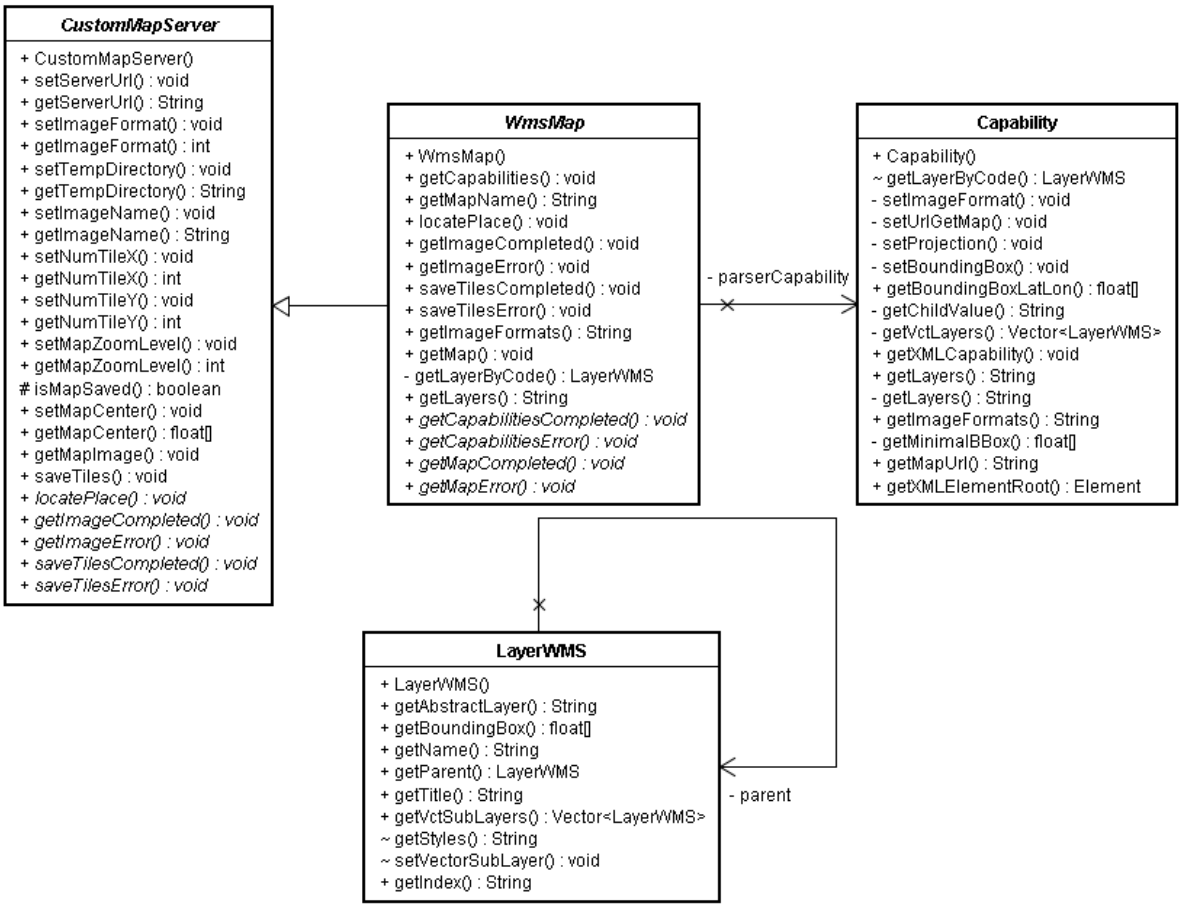

FIGURA 3.3: Classes da Camada WMS de Conectividade

A FIGURA 3.2 e a FIGURA 3.3 representam o esquema dos principais componentes da plataforma.

3.2.1 Camada WMS de Conectividade

A camada WMS de conectividade é o componente que permite à solução implementar o padrão WMS. Por ser projetada e executada sobre plataforma *desktop,* as limitações impostas sobre a rede móvel não influenciam seu desempenho.

A CWC foi divida em dois módulos principais: Módulo de controle de mapas e módulo de controle de clientes. Cada um dos módulos foi construído sobre classes abstratas, para facilitar a personalização, inclusão de novas características e implementação.

O módulo de controle de mapas é o responsável pela interação com os servidores de mapas. A classe principal, *CustomMapServer,* implementa funções comuns a interações com a maioria dos servidores de mapas como, por exemplo:

1. *setServerUrl*: Configura a url do servidor de mapas que será acessado pela aplicação;
2. *setImageFormat*: Configura o tipo de imagem que será solicitado ao servidor de mapas e posteriormente enviada ao cliente;
3. *setTempDirectory*: Configura o diretório onde os frames serão temporariamente armazenados enquanto são utilizados pelos clientes. Os frames armazenados nesse diretório também funcionam como *cache* para reduzir a carga sobre o servidor de mapas;
4. *setImageName*: Configura o nome que será dado aos arquivos em *cache*. Essa função é útil para separar os arquivos em categorias de mapas, camadas e *zoom*, para acesso concorrente de clientes;
5. *setNumTileX e setNumTileY*: Confiigura o número de *tiles* em que a imagem rederizada será divida, tanto sobre o eixo X, quando Y;
6. *setMapZoomLevel*: Configura o nível de *zoom* do mapa que será renderizado;
7. *saveTiles*: Divide o mapa recebido do servidor em *tiles* que serão enviados aos clientes;
8. *getMapImage*: Retorna as imagens processadas ao clientes.

Além desses, a classe *CustomMapServer* possui outros métodos, dentre eles os seguintes métodos abstratos:

- *getImageCompleted*;
- *getImageError;*
- *saveTilesCompleted;*
- *saveTilesError.*

Esses métodos foram criados para permitir maior integração e facilidade no momento de integrar ou incluir uma nova função à plataforma. O módulo de controle de clientes, por sua vez, foi projetado sobre as classes abstratas listadas a seguir:

1. *Server*: Responsável por controlar os acesso dos clientes e gerenciar suas conexões e pedidos;
2. *Client*: Representa a sessão do cliente conectado e armazena seus dados e preferências;
3. *ClientListener*: Responsável por dar a plataforma caracterista multi-usuário, pois permite ao servidor atender cada cliente separadamente.

A classe *Server* estende a classe *ClientListener*, tendo como métodos principais:

1. *startServer*: Inicia o servidor;
2. *onClientConnected*: Executado no momento da conexão do cliente;
3. *onReadSocketError*: Executado caso haja falha na comunicação com o cliente;
4. *onStartServerCompleted*: Executado quando o servidor é ativado;
5. *onStartServerError*: Executado caso haja falha na inicialização do servidor;
6. *onClientDisconnected*: Executado quando algum cliente se desconecta do servidor.

As funções *onClienteConnected e onReadSocketError* são implementações de funções abstratas da classe *ClientListener*. Já as funções *onStartServerCompleted, onStartError e onClientDisconnected*, são funções abstratas, ou seja, a implematação ficará a cargo do desenvolvedor, o proporcionará a flexibilidade para a criação de outras aplicações.

A classe *Client* representa se sessão do usuário dentro do servidor. Ela é responsável direta pela interação com os servidores de mapas, a partir da implementação da *CustomMapServer*. A classe *Client* não possui métodos públicos, sendo totalmente controlada pelos pedidos enviados pelos usuários do CM.

Como a carga mais pesada de trabalho está concentrada na renderização dos mapas, o CWC demanda pouco recurso da máquina onde está hospedado, Além disso, a memória *cache* armazenada na forma de arquivos temporários reduz a necessidade de comunicação com o servidor de mapas, proporcionando um consumo ainda menor de recursos e garantindo sua escalabilidade. O CWC não demanda recursos complexos de processamento ou memória. Ele trabalha com tarefas simples e pode ser visto como um interpretador ou tradutor, que permite que o CM interaja com o servidor de mapas escolhido. Porém, um estudo mais detalhado deve ser realizado para que se possa definir parâmetros de escalabilidade para o CWC. Pelo recurso de *cache* implementado pelo CWC, uma grande quantidade da carga de processamento e tempo que seria gasto com as requisições ao WMS são poupados, mas apenas se os CM utilizarem os mesmo recursos. Como esse é um cenário otimista e esta dissertação está focada na comunicação entre o CM e CWC, deixaremos essa discussão para trabalhos futuros.

3.2.2 Cliente Móvel

O Cliente Móvel, ou CM, foi construído em J2ME, versão de Java para aparelhos com baixo poder de processamento, limitações de memória e bateria usando arquitetura semelhante a do CWC. O desenvolvimento foi baseado em classes abstratas.

A principal classe do CM é a *WMSMap.* Ela é a responsável para interação com CWC. Apesar de possuir métodos como *getCapabilites* e *getMap*, nenhuma função relativa ao padrão WMS foi implementada do CM. Isso possibilitou a economia de recurso além de permitir que os métodos sejão personalizados para integração com servidores que por ventura não sigam o padrão WMS.

A seguir, é apresentado um exemplo do funcionamento real da solução proposta. A

FIGURA 3.4 mostra a tela de configuração, onde são introduzidos os endereços do servidor WMS e do CWC. Em seguida, é feita a conexão com o CWC. Caso a conexão seja bem sucedida, a aplicação automaticamente executa o método *getCapabilities* enviando a url configurada (@http://mapas.mma.gov.br/cgi-bin/mapserv?map=/opt/www/html/webservices/biorregioes.map&request=GetCapabilities&service=wms&version=1.1.1) e recendo do servidor a seguinte string: "@Biorregiões do Brasil#0&Biomas do Brasil#0#0&Limites estaduais do Brasil#0#1&Revisão Áreas prioritárias para conservação da biodiversidade (importância biológica)-2007#0#2&Revisão Áreas prioritárias para conservação da biodiversidade (prioridade de ação)-2007#0#3&". A *string* recebida contém os títulos das camadas disponíveis e um código de identificação atribuído pelo CWC.

A FIGURA 3.5 mostra a lista de camadas disponíveis, sendo que a camada que será recuperada encontra-se marcada. Ao clicarmos na opção "Atualizar Mapa", o metodo *getMap* é executado e, nesse momento, o CWC recebe as informações necessárias para que solicite ao WMS a renderização do mapa.

Ao final da transação, o CM automaticamente envia as requisições para os tiles do mapa que ocuparam a área visível da tela do computador móvel.

A FIGURA 3.6 mostra as imagens sendo carregadas no aparelho e a FIGURA 3.7 mostra o mapa totalmente formado.

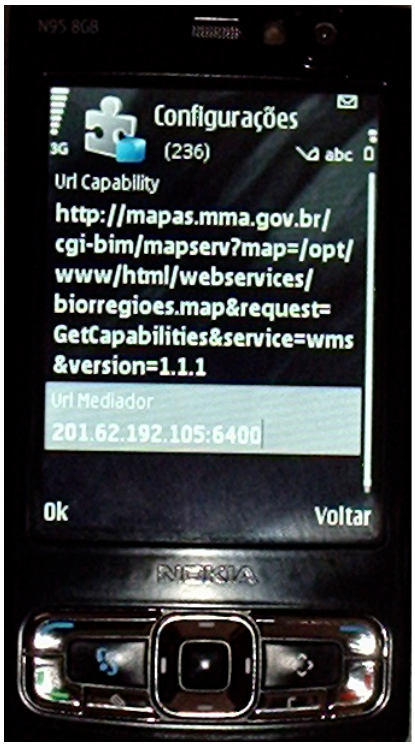

FIGURA 3.4: Configuração de endereço do WMS e CWC

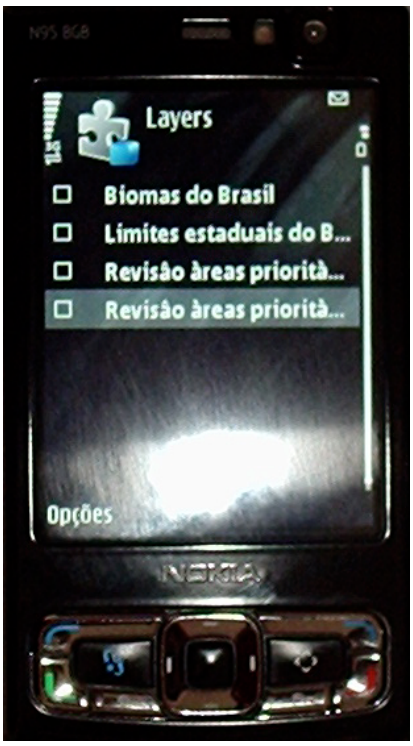

FIGURA 3.5: Lista de camadas disponíveis

FIGURA 3.6: Carregando no celular Nokia N95

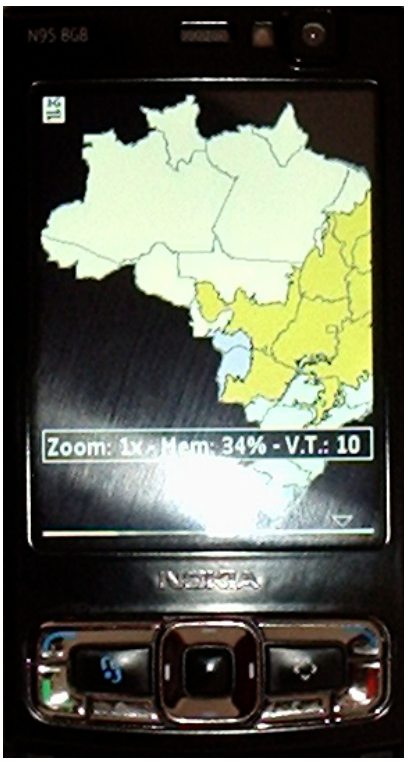

FIGURA 3.7: Tiles carregados

A TABELA 1 mostra a quantidade de bytes trafegados durante a execução do exemplo.

Ação	Tráfego de Saida (bytes)	Tráfego de Entrada (bytes)
Conectar	9	5
getCapabilities	137	268
getMap	10	11
getTiles	10	9.204

TABELA 1: Tabela de mensagens trocadas durante o teste de exemplo

Para que o consumo de banda fosse reduzido, foi necessário desenvolver um protocolo de comunicação baseado em *socket* que fosse capaz de transmitir as requisições e respostas para o cliente. Por isso, as mensagens transmitidas possuem tamanho reduzido e são identificadas por um código de apenas um byte.

A tabela a segui mostra o formato das mensagens que são trocadas entre os módulos.

Ação	Cliente	Servidor	Erro
Conectar	0x01@Lagura da tela@Altura da tela	0x01@Indice	0x01@0x0E
Desconectar	0x02		
definir wms	0x03@urlCapability	0x03@Lista de layers	0x03@0x0E
requisitar projeções	0x04	0x04@Qtde projeções@Lista de projeções	0x04@0x0E
requisitar layer abstract	0x05@Código do layer	0x05@Código do layer@abstract	0x05@0x0E
requisitar imagem de mapa	0x06@Zoom@Tipo da Imagem@Código do layer 1$...$Código do layer N	0x06@0x01	0x06@0x0E
requistar tile do mapa	0x07@Código do tile	0x07@Tile	0x07@0x0E

TABELA 2: Formato das mensagens trocadas entre CM e CWC

Quanto ao gerenciamento de memória, para evitar que a aplicação consuma mais memória do que o computador móvel possa disponibilizar, o CM monitora constantemente a memória disponível seguindo os critérios de invalidação de *cache* descritos anteriormente. Como as funções principais do CM foram escritas em Java padrão, a criação de um cliente para *desktop* poderia ser realizada com nível reduzido de retrabalho. O Anexo I apresenta o código fonte para o cliente móvel e para o cliente para *desktop*, sendo que este último é compatível com sistemas Windows e Linux.

4. Experimentos e análises

Foram realizados dois conjuntos de experimentos. No primeiro, o WMS utilizado foi o mantido pela *Natural Resources Canada*[18] e, no segundo, os dados foram obtidos do Ministério Brasileiro de Meio Ambiente e Recursos Naturais[19]. Em ambos os experimentos, o *log* de dados do aparelho celular foi utilizado para medir o tráfego gerado pelas aplicações. Consideramos a medida de tempo de resposta inconsistente, devido a flutuações de carga nos servidores envolvidos e na própria rede de transmissão móvel. Assim, como medida de desempenho, decidimos verificar a redução da transferência de dados do ponto de vista do computador móvel. Como consideração final, ressaltamos que os resultados foram obtidos a partir da média de três execuções. Os dados detalhados podem ser encontrados na TABELA 6.

Realizamos comparações entre o cliente WMS Skylab e nosso protótipo, considerando três tipos de interação, nesta ordem: (1) uma requisição getCapabilities, (2) a recuperação inicial de uma camada, e (3) uma operação de pan envolvendo um deslocamento para a direita igual à largura da tela. Esses experimentos foram realizados em três ambientes. No primeiro, usamos o emulador Sun Java Wireless Toolkit versão 2.5.2 (TABELA 3) sobre rede WiFi. No segundo usamos um telefone celular Nokia 6111 sobre rede GPRS 2.5G (TABELA 5). No terceiro, o aparelho utilizado foi o Nokia modelo N95, sobre rede GPRS G3 (TABELA 4 e TABELA 7). Todos os teste foram realizados usando a rede GPRS da TIM Brasil.

Ação	Skylab (KB)	Nossa Proposta (KB)	Redução
getCapabilities	81,17	18,34	77,00%
Recuperação de Layer	25,70	34,24	-35,00%
Pan	70,01	3,30	95,00%

TABELA 3: Resultado dos testes com emulador WTK v2.5.2 na rede WiFi

Ação	Skylab (KB)	Nossa Proposta (KB)	Redução
getCapabilities	89,71	21,89	76,00%
Recuperação de Layer	26,04	36,21	-39,00%
Pan	52,32	21,46	59,00%

TABELA 4: Resultado dos testes realizados com Nokia N95 na rede GPRS 3G

[18] http://132.156.88.15/wmsconnector/com.esri.wms.Esrimap/GDR_E?SERVICE=WMS&REQUEST=GetCapabilities&version=1.1.0

[19] http://mapas.mma.gov.br/cgi-bin/mapserv?map=/opt/www/html/webservices/biorregioes.map&request=GetCapabilities&service=wms&version=1.1.1

O aumento no volume da transmissão na operação de recuperação de camadas é devido ao *overhead* associado a cada pedaço do mapa individualmente. Na proposta apresentada, uma imagem grande é gerada pelo servidor, e então é dividida em pedaços pelo CWC. O aumento na quantidade de bytes transmitidos sobre a rede GPRS se deve à perda e à retransmissão de pacotes, pois a eficiência da transmissão está ligada à qualidade do sinal da operadora, no momento da utilização do recurso.

Com essa estratégia, podemos evitar a retransmissão quando o usuário realiza operações de *pan*, que são normalmente muito comuns, considerando as pequenas dimensões das telas dos celulares ou computadores móveis. O resultado dos testes indica que o *overhead* na recuperação de camadas pode ser totalmente compensado pela economia em uma única operação de *pan*: enquanto a recuperação de uma camada implica em uma desvantagem de 10 Kbytes para nossa proposta, uma operação de pan implica numa economia de mais de 30 Kbytes.

Este ganho na operação de *pan* é resultado direto do uso de cache para as partes da imagem no computador móvel e CWC. Uma vez que a imagem originalmente solicitada ultrapassa o tamanho da tela em 50% da altura e largura, quando o pan é executado vàrias das imagens necessárias já estão disponíveis. Note também a grande diferença nas medidas entre o emulador e o aparelho celular causada pela maior velocidade de transmissão e eficiência da rede WiFi em relação à rede GPRS. Isto indica que os testes em computadores móveis devem ser executados em redes GPRS, visto que esta limitação representa um desafio para a medição do *throughput* e tempo resposta, pois a velocidade da rede varia ao longo do dia, devido aos padrões de tráfego.

No segundo grupo de testes, os *tiles* foram criados com um terço e um quarto dos valores da largura e altura respectivamente. Nessa bateria, o cliente Skylab mostrou-se ineficiente, não apresentando nenhum resultado. Nesse experimento foram usadas quatro camadas disponíveis do servidor: (A) Biomas do Brasil, (B) Limites Estaduais do Brasil, (C) Revisão áreas prioritárias para conservação da biodiversidade (importância biológica)-2007 e (D) Revisão áreas prioritárias para conservação da biodiversidade (prioridade de ação)-2007.

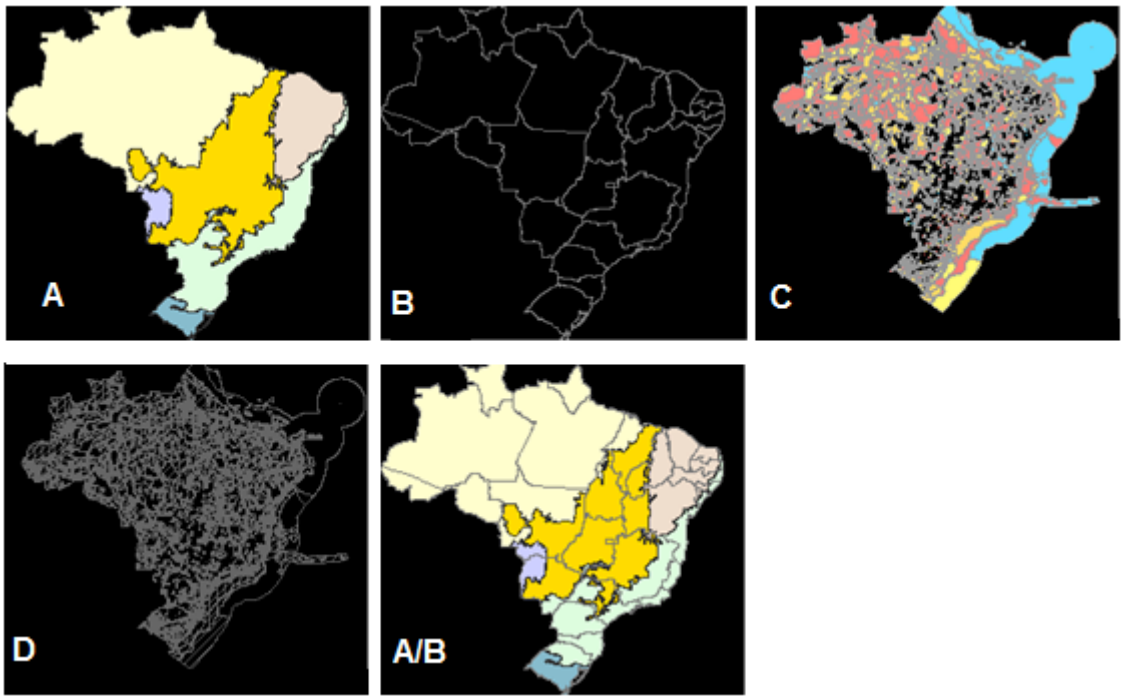

FIGURA 4.1 Camadas do WMS usado no segundo teste

A FIGURA 4.1 ilustra as camadas utilizadas nos testes apresentados em seguida.

Cinco testes foram realizados utilizando esses dados, tentado simular condições reais de utilização:

1. Recuperação de camada;
2. *Pan* de 50% do tamanho da tela para a esquerda;
3. Ao mento de *zoom*;
4. *Pan* de 50% para a direita;
5. Redução de 50% do *zoom*.

A TABELA 5 mostra os resultado obtidos durante os testes.

		Camadas					
		A	B	C	D	A/B	Média
CWC/CM	TX (KB)	4,87	4,91	5,29	5,58	5,20	5,17
	RX (KB)	21,12	21,60	23,73	24,72	23,86	23,00

TABELA 5: Resultado dos testes realizados com Nokia 6111 na rede GPRS 2.5G

Para o terceiro grupo de testes, os *tiles* criados no CWC possuíam dimensões iguais a um oitavo das medidas da tela, nesse caso 30 por 40 *pixels*. Para melhorar o desempenho da aplicação, foram carregados grupos de *frames* além da área de visualização do usuário, possibilitando que as operações de *pan* ocorressem de forma mais eficiente, reduzindo o tempo de espera do usuário. Nesse experimento

foram usadas as mesmas camadas do experimento anterior, assim como os mesmos procedimentos.

Quatro dos cinco testes realizados utilizaram camadas fornecidas pelo WMS individualmente (camadas de A a D), no quinto teste, foi usada a combinação entre as camadas A e B. A TABELA 7 mostra que nossa implementação é capaz de economizar até 46% (média de 37%) em bytes transmitidos e recebidos por interação.

		Camadas (Primeira execução)					
		A	B	C	D	A/B	Média
CWC/CM	TX (KB)	5,34	5,18	6,17	5,46	5,51	5,34
	RX (KB)	16,53	17,39	31,65	26,27	23,44	23,06
Skylab	TX (KB)	4,93	5,54	4,44	4,04	3,50	4,49
	RX (KB)	35,07	30,92	46,17	35,35	34,02	36,31
Ganho	TX (KB)	-8,32%	6,50%	-38,96%	-35,15%	-57,43%	-23,21%
	RX (KB)	52,87%	43,76%	31,45%	25,69%	31,10%	36,50%
	TX + RX	44,55%	50,26%	-7,51%	-9,46%	-26,33%	13,29%
		Camadas (Segunda execução)					
		A	B	C	D	A/B	Média
CWC/CM	TX (KB)	5,29	5,12	5,69	5,24	5,23	5,31
	RX (KB)	16,47	15,8	29,04	24,44	19,91	21,13
Skylab	TX (KB)	4,72	5,31	4,41	3,82	3,09	4,27
	RX (KB)	35,67	32,28	47,56	37,23	36,2	37,79
Ganho	TX (KB)	-27,13%	-5,76%	-51,38%	-69,18%	-117,70%	-45,72%
	RX (KB)	42,59%	48,83%	33,39%	27,51%	44,26%	38,68%
	TX + RX	15,47%	43,07%	-17,99%	-41,67%	-73,44%	-7,04%
		Camadas (Terceira execução)					
		A	B	C	D	A/B	Média
CWC/CM	TX (KB)	5,29	5,12	5,69	5,24	5,23	5,31
	RX (KB)	16,47	15,8	29,04	24,44	19,91	21,13
Skylab	TX (KB)	4,72	5,31	4,41	3,82	3,09	4,27
	RX (KB)	35,67	32,28	47,56	37,23	36,2	37,79
Ganho	TX (KB)	-2,46%	8,86%	-1,25%	-13,38%	-46,41%	-7,90%
	RX (KB)	66,09%	60,13%	51,47%	49,43%	58,38%	56,61%
	TX + RX	63,63%	68,99%	50,22%	36,05%	11,98%	48,71%

TABELA 6: Relação de trafego de dados nas três execuções da aplicação

Ao compararmos os resultados dos testes realizados nas redes GPRS 2.5G e 3G, é possível notar uma variação média na taxa de transferência de 2,7% para

envio e -8,13% para recebimento. Acreditamos que essa diferença seja resultado da maior repetição para envio de pacotes na rede 2,5G em relação à rede 3G, por isso a diferença de tráfego para envio foi tão pequena, já que os pacotes enviados do CM para o CWC são muito pequenos (na média, menor que 50 bytes).

		Camadas					
		A	B	C	D	A/B	Média
CWC/CM	TX (KB)	5,29	5,12	5,69	5,24	5,23	5,31
	RX (KB)	16,47	15,8	29,04	24,44	19,91	21,13
Skylab	TX (KB)	4,72	5,31	4,41	3,82	3,09	4,27
	RX (KB)	35,67	32,28	47,56	37,23	36,2	37,79
Ganho	TX (KB)	-12,08%	3,58%	-29,02%	-37,17%	-69,26%	-24,45%
	RX (KB)	53,83%	51,05%	38,94%	34,35%	45,00%	44,08%
	TX + RX	41,75%	54,63%	9,92%	-2,82%	-24,26%	19,63%

TABELA 7: Testes com WMS do Ministério Brasileiro de Meio Ambiente e Recursos Naturais

O protótipo apresentado neste trabalho transmite mais dados que o *Skylab* devido ao custo inicial para carregar cada *tile*. Contudo, o impacto provocado pela retransmissão de imagens de tamanho igual ao da tela dos aparelhos a cada interação gera um volume muito maior de trafego.

Os experimentos simularam as operações típicas de um usuário de serviços WMS. Uma melhor abordagem seria a de caracterizar um trabalho típico de usuário WMS, mas não existem padrões estabelecidos. Portanto, nossos experimentos devem ser expandidos no futuro, utilizando alguma interação homem-máquina ou técnicas de usabilidade, a fim de se avaliar o impacto da nossa abordagem em situações reais. Os resultados preliminares mostrados anteriormente apontam um ganho significativo por parte da nossa proposta.

Por fim, é importante ressaltar que o CWC foi instalado num PC com processador Athlon 400 MHz, placa mãe Soyo K7-VTA Pro, com 128 MB de memória RAM.

5. Conclusão

Esta dissertação apresentou como principal contribuição um cliente WMS funcional para dispositivos móveis. A solução oferece considerável redução no tráfego de dados, com conseqüências positivas sobre o custo da comunicação, tempo de processamento e duração da carga da bateria. Foram observadas reduções de até 44% na recepção de dados em uma simulação de uma interação curta. Interações mais longas poderão apresentar ganhos ainda maiores. A mesma estratégia também pode ser implementada em clientes *desktop*, para tentar aumentar o potencial de escalabilidade das soluções baseadas em infraestruturas de dados espaciais apoiados em SOA. Como exemplo, o Anexo I deste trabalho traz o código fonte da adaptação da solução implementada para uso em um cliente *desktop* ou em uma plataforma completa e de menor porte, como um netbook.

Testes mostraram que o volume de dados transmitidos entre o cliente móvel e o servidor foi significativamente reduzido, sendo que nos momentos iniciais da comunicação (*getCapabilities*), a redução chegou a 90%. Naturalmente, redução no volume de dados trafegados implica em menor consumo de banda, processamento e bateria. É importante ressaltar que, quanto menor o número de camadas e metadados disponíveis para transporte do servidor WNS para o cliente, menor será o ganho inicial. Porém, as estratégias de *cache* de imagens aliadas à divisão dos mapas em *tiles* nos garante economia suficiente para que a proposta apresentada produza resultados relevantes.

Trabalhos futuros incluem medições mais detalhadas relativas ao tráfego envolvido na comunicação entre o servidor WMS e o CWC e entre o CWC e CM. O objetivo será determinar com maior precisão o ganho obtido em termos de velocidade média das transações, bem como determinar o benefício gerado para o servidor WMS quanto ao aumento de escalabilidade devido à redução de processamento necessária para atender aos clientes móveis. Além disso, buscaremos formas eficientes de medir o potencial de escalabilidade da nossa própria solução. Uma medição mais apurada do tráfego gerado em diferentes tipos de redes móveis ainda precisa ser feita. Para tanto, será necessário caracterizar melhor uma carga de trabalho que se assemelhe a interações “típicas” e isso, por sua vez, só pode ser feito em função da implementação de aplicações realistas.

Outro trabalho futuro importante envolve estender a estratégia apresentada aqui para outros serviços OGC, em particular o WFS e o CWS. Juntamente com o WMS, esses são atualmente os serviços Web OGC mais utilizados e conhecidos. A partir da disponibilidade de serviços como o *Web Processing Service* (WPS), de padronização recente, será possível dispor, a partir de um computador móvel, não apenas da capacidade de consulta e visualização, mas também de acesso a objetos individuais e a funções e análises mais sofisticadas. Sendo assim, uma direção importante de pesquisa que sucede este trabalho envolve trazer todo o potencial de sistemas de informação geográficos de acoplamento fraco para o ambiente móvel.

6. Referencias

Alves, L. L. e C. A. Davis Jr (2007). Evaluation of OGC Web Services for Local Spatial Data Infrastructures and for the Development of Clients for Geographic Information Systems. 217-234.

C. A. Davis Jr e Alves, L. L. (2006). "Infra-Estruturas de Dados Espaciais: Potencial para Uso Local."

C. A. Davis Jr, F. Fonseca, et al. (2009). Integração entre Ciência e Comunidade. II Seminário sobre Grandes Desafios da Computação no Brasil, Manaus (AM), Sociedade Brasileira de Computação.

C. A. Davis Jr (2008). Spatial Data Infrastructures. Encyclopedia of Information Science and Technology. M. Khosrow-Pour. Hershey, Pennsylvania, USA, IGI Global. **VII:** 3548-3553.

C. A. Davis Jr e Alves L. L. (2007). Geospatial Web Services. Encyclopedia of GIS. S. Shekhar and H. Xiong. Berlin, Germany, Springer-Verlag**:** 1270-1273.

Duarte-Figueiredo, F. L. P. and A. A. Loureiro (2007). An end-to-end wireless QoS architecture evaluation. 12th IEEE Symposium on Computers and Communications (ISCC 2007).

Ferris, C. and J. Farrell (2003). "What Are Web Services?" Communications of the ACM **46**(6): 31.

Fonseca, F. T. (2008). Spatial Data Infrastructure. Encyclopedia of GIS. S. X. Shekhar, Hui London ; New York, Springer.

Forman, G. H. and J. Zahorjan (2004). "The challenges of mobile computing." IEEE Computer **27**(4): 38-47.

GeoServer. (2008). "Users Guide." Retrieved 10-08-2008, 2008, from http://geoserver.org/display/GEOSDOC/Users+Guide.

gvSIG (2009).

Maguire, D. J. and P. A. Longley (2005). "The emergence of geoportals and their role in spatial data infrastructures." Computers, Environment and Urban Systems **29**(1): 3-14.

MapServer. (2008). "Documentation." Retrieved 10-08-2008, 2008, from http://mapserver.gis.umn.edu/bem-vindo-ao-mapserver?set_language=pt.

Mobilesystems, S. (2005). J2ME OGC WMS Client, User Manual, Skylab Mobilesystems.

OGC (1998). OpenGIS Simple Features Specification for SQL. Boston, Open GIS Consortium.

OGC (2006). OpenGIS® Web Map Server Implementation Specification, Open Geospatial Consortium Inc.

Oliveira, P. A., C. A. Davis, et al. (2008). "Proposição de Infra-Estrutura de Dados Espaciais (SDI) Local, Baseada em Arquitetura Orientada por Serviços (SOA)."

Open Geospatial Consortium (OGC) (2006) "OpenGIS Web Map Server Implementation Specification, version 1.3.0." DOI: Document number OGC 06-042.

OpenGEO (2007). "Plano Diretor de Geoprocessamento da Prefeitura Municipal de Fortaleza."

OpenLayer.org (2009).

Quantun GIS, P. (2009). QGIS 1.0.

Rajabifard, A., I. P. Williamson, et al. (2000). "From Local to Global SDI initiatives: a pyramid of building blocks."

Ravada, S. (1999). "Oracle8i Spatial: Experiences with Extensible Databases."

Refractions Research, I. (2008). "PostGIS 1.4.0SVN Manual."

Ribeiro, A. I., F. L. P. Duarte-Figueiredo, et al. (2008). An artificial neural network approach for mechanisms of call admission control in UTMS 3G networks. 8th International Conference on Hybrid Intelligent Systems (HIS'08).

Sun Microsystems, I. (2008). "Java ME at a Glance." Retrieved 2008-08-15, 2008, from http://java.sun.com/javame/index.jsp.

Tait, M. G. (2005). "Implementing geoportals: applications of distributed GIS." Computers, Environment and Urban Systems **29**(1): 33-47.

Tang, W. S. M. (2003). Mobile Geographic Information Services (M-GIS): Technology Tha Changes the Way We Navigate Our World, The Hong Kong Intitution of Engineeing Surveyors. **Vol 5:** 43-50.

Tu, S. and M. Abdelguerfi (2006). "Web Services for Geographic Information Systems." IEEE Internet Computing **10**(5): 13-15.

Uchoa, H. N. (2008). "SIG em Software Livre Aplicado ao Cadastro Social."

W3C (2002). Web Services Architecture Working Group, Web Services Architecture Requirements, W3C Working Draft, World-Wide Web Consortium.

Anexo I – XML *GetCapabilities*

```
<?xml version='1.0' encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE WMT_MS_Capabilities SYSTEM
"http://schemas.opengis.net/wms/1.1.1/WMS_MS_Capabilities.dtd"
 [ <!ELEMENT VendorSpecificCapabilities EMPTY> ]>
<!-- end of DOCTYPE declaration -->

<WMT_MS_Capabilities version="1.1.1">

<!-- MapServer version 5.2.1 OUTPUT=GIF OUTPUT=PNG OUTPUT=JPEG OUTPUT=WBMP
OUTPUT=SVG SUPPORTS=PROJ SUPPORTS=AGG SUPPORTS=FREETYPE SUPPORTS=ICONV
SUPPORTS=WMS_SERVER SUPPORTS=WMS_CLIENT SUPPORTS=WFS_SERVER
SUPPORTS=WFS_CLIENT SUPPORTS=THREADS SUPPORTS=GEOS INPUT=EPPL7
INPUT=POSTGIS INPUT=OGR INPUT=GDAL INPUT=SHAPEFILE -->

<Service>
  <Name>OGC:WMS</Name>
  <Title>Biorregiões do Brasil</Title>
  <Abstract>Biorregiões do Brasil. Biorregião é um espaço geográfico que
abriga integralmente um ou vários ecossistemas, sendo demarcado por seu
sistema natural, cultural e histórico</Abstract>
        <KeywordList>
          <Keyword>Biomas</Keyword>
          <Keyword>Bioma</Keyword>
          <Keyword>Biodiversidade</Keyword>
          <Keyword>Biorregiões</Keyword>
        </KeywordList>
  <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:href="http://mapas.mma.gov.br/cgi-
bin/mapserv?map=/opt/www/html/webservices/biorregioes.map&amp;amp;"/>
  <ContactInformation>
    <ContactPersonPrimary>
      <ContactPerson>Edmar Moretti</ContactPerson>
      <ContactOrganization>Min do Meio-ambiente
Brasil</ContactOrganization>
    </ContactPersonPrimary>
      <ContactPosition>Gerente de projeto</ContactPosition>
    <ContactAddress>
        <AddressType>postal</AddressType>
        <Address></Address>
        <City>Brasilia</City>
        <StateOrProvince>Distrito Federal</StateOrProvince>
        <PostCode></PostCode>
        <Country>Brasil</Country>
    </ContactAddress>

<ContactElectronicMailAddress>edmar.moretti@mma.gov.br</ContactElectronicMa
ilAddress>
  </ContactInformation>
  <Fees>none</Fees>
  <AccessConstraints>vedado o uso comercial</AccessConstraints>
</Service>

...

  <Layer>
    <Name>biorregioes</Name>
    <Title>Biorregiões do Brasil</Title>
    <SRS>epsg:4291</SRS>
    <SRS>epsg:4326</SRS>
```

```
    <LatLonBoundingBox minx="-76.5126" miny="-36.9484" maxx="-25.5852"
maxy="7.04601" />
    <BoundingBox SRS="EPSG:4291"
                 minx="-76.5126" miny="-36.9484" maxx="-25.5852"
maxy="7.04601" />
    <Layer queryable="1" opaque="0" cascaded="0">
        <Name>biomas</Name>
        <Title>Biomas do Brasil</Title>
        <Abstract>Biomas do Brasil. O mapeamento dos biomas brasileiros é
resultado da parceria entre o IBGE e o Ministério do Meio Ambiente
(MMA)</Abstract>
        <KeywordList>
          <Keyword>Biomas</Keyword>
          <Keyword>Bioma</Keyword>
          <Keyword>Biodiversidade</Keyword>
          <Keyword>Biorregiões</Keyword>
        </KeywordList>
        <SRS>epsg:4291</SRS>
        <SRS>epsg:4326</SRS>
        <LatLonBoundingBox minx="-73.9909" miny="-33.6039" maxx="-32.349"
maxy="5.27222" />
        <BoundingBox SRS="EPSG:4291"
                     minx="-73.9909" miny="-33.6039" maxx="-32.349"
maxy="5.27222" />
        <Style>
          <Name>default</Name>
          <Title>default</Title>
          <LegendURL width="107" height="101">
             <Format>image/png</Format>
             <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple" xlink:href="http://mapas.mma.gov.br/cgi-
bin/mapserv?map=/opt/www/html/webservices/biorregioes.map&amp;version=1.1.1
&amp;service=WMS&amp;request=GetLegendGraphic&amp;layer=biomas&amp;format=i
mage/png&amp;STYLE=default"/>
          </LegendURL>
        </Style>
    </Layer>
    <Layer queryable="1" opaque="0" cascaded="0">
        <Name>estadosl</Name>
        <Title>Limites estaduais do Brasil</Title>
        <Abstract>Limites estaduais do Brasil. A delimitação tendo como
base a Malha Municipal Digital do Brasil - IBGE (2001)</Abstract>
        <KeywordList>
          <Keyword>limites estaduais</Keyword>
          <Keyword>estados</Keyword>
          <Keyword>estados do Brasil</Keyword>
        </KeywordList>

        <Name>areas_priori_priori</Name>
        <Title>Revisão áreas prioritárias para conservação da
biodiversidade (prioridade de ação)-2007</Title>
        <Abstract>Revisão áreas prioritárias para conservação da
biodiversidade (2007)-classificadas de acordo com prioridade de
ação</Abstract>
        <KeywordList>
          <Keyword>áreas prioritárias</Keyword>
          <Keyword>conservação</Keyword>
          <Keyword>biodiversidade</Keyword>
          <Keyword>revisão</Keyword>
          <Keyword>prioridade de ação</Keyword>
        </KeywordList>
```

```
        <SRS>epsg:4291</SRS>
        <SRS>epsg:4326</SRS>
        <!-- WARNING: Optional LatLonBoundingBox could not be established
for this layer.  Consider setting LAYER.EXTENT or wms_extent metadata. Also
check that your data exists in the DATA statement -->
        <Style>
          <Name>default</Name>
          <Title>default</Title>
          <LegendURL width="134" height="50">
             <Format>image/png</Format>
             <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
xlink:type="simple" xlink:href="http://mapas.mma.gov.br/cgi-
bin/mapserv?map=/opt/www/html/webservices/biorregioes.map&amp;amp;version=1.1.1
&amp;amp;service=WMS&amp;amp;request=GetLegendGraphic&amp;amp;layer=areas_priori_priori
&amp;amp;format=image/png&amp;amp;STYLE=default"/>
          </LegendURL>
        </Style>
    </Layer>
  </Layer>
</Capability>
</WMT_MS_Capabilities>
```

Anexo II - Cliente Móvel

```
package mobile.wms;

import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Canvas;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Graphics;
import javax.microedition.lcdui.Image;
import mediador.wms.WMSMap;

/**
 *
 * @author Yuri
 */
public class Screen extends Canvas implements CommandListener, Runnable
{

  private static final int SHOW_SPLASH = 0x01;
  private static final int CREATE_MAP = 0x02;
  private static final int SHOW_MAP = 0x03;
  private int status = SHOW_SPLASH;
  private Command cmdShowConfig;
  private Command cmdConnectServer;
  private Command cmdDesconnectServer;
  private Command cmdShowLayers;
  private Command cmdShowHelp;
  private WMSMap wmsMap;
  private Command cmdExit;
  private Image imgBackGroung;
  private Image imgArrow;

  public Screen()
  {
    setTitle("Mobile WMS Client");
    setFullScreenMode(true);
    init();
  }

  private Image resizeImage(Image srcImage)
  {
    int srcWidth = srcImage.getWidth();
    int srcHeight = srcImage.getHeight();
    Image tmp = Image.createImage(getWidth(), getHeight());
    Graphics g = tmp.getGraphics();

    int ratio = (srcWidth << 16) / getWidth();
    int pos = ratio / 2;

    //Horizontal Resize

    for (int x = 0; x < getWidth(); x++)
    {
      g.setClip(x, 0, 1, srcHeight);
```

```
      g.drawImage(srcImage, x - (pos >> 16), 0, Graphics.LEFT |
Graphics.TOP);
      pos += ratio;
    }

    Image resizedImage = Image.createImage(getWidth(), getHeight());
    g = resizedImage.getGraphics();
    ratio = (srcHeight << 16) / getHeight();
    pos = ratio / 2;

    //Vertical resize

    for (int y = 0; y < getHeight(); y++)
    {
      g.setClip(0, y, getWidth(), 1);
      g.drawImage(tmp, 0, y - (pos >> 16), Graphics.LEFT |
Graphics.TOP);
      pos += ratio;
    }

    return resizedImage;
  }//resize image

  private void init()
  {
    imgBackGroung = loadImage("/mobile/logo.png");
    imgArrow = loadImage("/mobile/arrow.png");

    cmdConnectServer = new Command("Conectar ao servidor",
Command.ITEM, 0);
    cmdShowConfig = new Command("Configurações", Command.ITEM, 1);
    cmdDesconnectServer = new Command("Desonectar do servidor",
Command.ITEM, 10);
    cmdShowLayers = new Command("Layers", Command.ITEM, 1);
    cmdShowHelp = new Command("Ajuda", Command.ITEM, 2);
    cmdExit = new Command("Sair", Command.ITEM, 9);

    addCommand(cmdShowConfig);
    addCommand(cmdConnectServer);
    addCommand(cmdShowHelp);
    addCommand(cmdExit);

    setCommandListener(this);

    (new Thread(this)).start();
  }

  private void showSplash()
  {
    status = CREATE_MAP;
  }

  public void createWMSMap()
  {
    wmsMap = new mediador.wms.WMSMap(MobileWMSClient.urlConnection,
MobileWMSClient.urlCapability, getWidth(), getHeight(), true)
    {
```

```
      public void wmsConnectionError(Exception exception)
      {
        MobileWMSClient.setCurrent(new Alert("Erro",
exception.getMessage() + (char)10 + "Erro conectando ao servidor.",
null, AlertType.ERROR), MobileWMSClient.screem);
        System.out.println(exception.getMessage());
      }

      public void connectionCompleted()
      {
        MobileWMSClient.setCurrent(new FrmGauge("Lendo catálogo..."));
        removeCommand(cmdConnectServer);
      }

      public void getCapabilityError(Exception exception)
      {
        MobileWMSClient.setCurrent(new Alert("Erro", "Erro ao ler
catálogo.", null, AlertType.ERROR), MobileWMSClient.screem);
        System.out.println(exception.getMessage());
      }

      public void getCapabilityCompleted()
      {
        addCommand(cmdShowLayers);

        MobileWMSClient.setCurrent(getListLayers());

        System.out.println("GetCapability completado");
      }

      public void getImageFormatsError(Exception exception)
      {
        MobileWMSClient.setCurrent(new Alert("Erro", "Erro ao ler lista
de formato de imagens.", null, AlertType.ERROR),
MobileWMSClient.screem);
        System.out.println(exception.getMessage());
      }

      public void getImageFormatsCompleted()
      {
        MobileWMSClient.setCurrent(MobileWMSClient.screem);
        System.out.println("GetImageFormats completado");
      }

      public void getLayerAbstractError(Exception exception)
      {
        MobileWMSClient.setCurrent(new Alert("Erro", "Erro ao ler
abstract layer.", null, AlertType.ERROR), MobileWMSClient.screem);
        System.out.println(exception.getMessage());
      }

      public void getLayerAbstractCompleted(String layerCode, String
layerAbstract)
      {
        MobileWMSClient.setCurrent(new FrmInfo("Abstract Layer",
layerAbstract));
        System.out.println("Get Layer Abstract completado");
      }
```

```
      public void getTileError(Exception exception)
      {
        MobileWMSClient.setCurrent(new Alert("Erro", "Erro ao carregar
tile.", null, AlertType.ERROR), MobileWMSClient.screem);
        System.out.println(exception.getMessage());
      }

      public void getTileCompleted(int x, int y, Image tile)
      {
        MobileWMSClient.setCurrent(MobileWMSClient.screem);
        //System.out.println("Get Tile completado");
      }

      public void getMapError(Exception exception)
      {
        MobileWMSClient.setCurrent(new Alert("Erro", "Erro ao pedir
mapa.", null, AlertType.ERROR), getListLayers());

        System.out.println(exception.getMessage());
      }

      public void getMapCompleted()
      {
        MobileWMSClient.setCurrent(MobileWMSClient.screem);

        System.out.println("Get Map completado");
      }

      public void getMapStarted()
      {
        MobileWMSClient.setCurrent(new FrmGauge("Buscando mapa..."));
      }
    };

    status = SHOW_MAP;
  }

  protected void paint(Graphics g)
  {
    switch (status)
    {
      case SHOW_SPLASH:
        showSplash();
        break;
      case CREATE_MAP:
        createWMSMap();
        break;
      case SHOW_MAP:
        g.drawImage(imgBackGroung, (getWidth() -
imgBackGroung.getWidth()) / 2, (getHeight() -
imgBackGroung.getHeight()) / 2, 0);

        wmsMap.paint(g);

        g.drawImage(imgArrow, (getWidth() - (imgArrow.getWidth() * 2)),
getHeight() - 12, 0);
```

```
      break;
  }
}

protected void keyPressed(int keyCode)
{
  int action = keyCode;

  try
  {
    action = getGameAction(keyCode);
  }
  catch (Exception exception)
  {
    exception.getMessage();
  }

  wmsMap.keyPressed(keyCode, action);
}

protected void keyReleased(int keyCode)
{
  int action = keyCode;

  try
  {
    action = getGameAction(keyCode);
  }
  catch (Exception exception)
  {
    exception.getMessage();
  }

  wmsMap.keyReleased(keyCode, action);
}

public void commandAction(Command command, Displayable d)
{
  if (command == cmdShowConfig)
    MobileWMSClient.setCurrent(MobileWMSClient.frmConfig);
  else if (command == cmdConnectServer)
  {
    MobileWMSClient.setCurrent(new FrmGauge("Conectando..."));

    wmsMap.connect();
  }
  else if (command == cmdDesconnectServer)
  {
    MobileWMSClient.setCurrent(new FrmGauge("Desconectando..."));
    wmsMap.desconnect();
  }
  else if (command == cmdShowLayers)
    MobileWMSClient.setCurrent(wmsMap.getListLayers());
  else if (command == cmdShowHelp)
  {
  }
  else if (command == cmdExit)
    MobileWMSClient.quit();
```

```
  }

  void lapso(long time)
  {
    try
    {
      Thread.sleep(time);
    }
    catch (Exception exception)
    {
      exception.getMessage();
    }
  }

  public void run()
  {
    while (true)
    {
      lapso(50);

      repaint();
    }
  }

  private static Image loadImage(String imagePath)
  {
    try
    {
      return Image.createImage(imagePath);
    }
    catch (Exception exception)
    {
      exception.getMessage();

      return null;
    }
  }
}
```

Anexo III - Cliente Netbook

```
/*
 * FrmPrincipal.java
 *
 * Created on 30/06/2009, 02:06:18
 */
package notewmsclient;

import java.awt.Graphics;
import java.awt.Image;
import java.util.Vector;
import mediador.wms.WMSMap;

/**
 *
 * @author Yuri
 */
public class FrmPrincipal extends javax.swing.JFrame
{

  WMSMap wmsMap;
  private Graphics graphics;
  private Repaint repaint;

  /** Creates new form FrmPrincipal */
  public FrmPrincipal()
  {
    initComponents();

    init();
  }

  void paintMap()
  {
    wmsMap.paint(graphics);
  }

  private void init()
  {
    setBounds(100, 100, 650, 510);
    txtGetCapabilities.setText("http://mapas.mma.gov.br/cgi-
bin/mapserv?map=/opt/www/html/webservices/biorregioes.map&request=GetCa
pabilities&service=wms&version=1.1.1");
    txtCWC.setText("127.0.0.1:6400");

    graphics = pnlMap.getGraphics();

    repaint = new Repaint(this);
  }

  private void createWMSMap()
  {
    if (wmsMap != null)
    {
      wmsMap.desconnect();
      wmsMap = null;
    }
```

```
    String cwc[] = txtCWC.getText().split(":");

    wmsMap = new WMSMap(cwc[0], Integer.parseInt(cwc[1]),
txtGetCapabilities.getText(), pnlMap.getWidth(), pnlMap.getHeight(),
true)
    {

      @Override
      public void wmsConnectionError(Exception exception)
      {
        lblDebug.setText("Erro de conexão: " + exception.getMessage());
      }

      @Override
      public void connectionCompleted()
      {
        lblDebug.setText("Conexão completada");
      }

      @Override
      public void getCapabilityError(Exception exception)
      {
        lblDebug.setText("Erro na execução do getCapabilities: " +
exception.getMessage());
      }

      @Override
      public void getCapabilityCompleted(String capabilities)
      {
        lblDebug.setText("GetCapabilities completado");

        String lstTemp[] = capabilities.split("&");

        Vector<String> vctTemp = new Vector<String>();

        for (int i = 0; i < lstTemp.length; i++)
        {
          if (lstTemp[i].split("#").length > 2)
          {
            vctTemp.add(lstTemp[i]);
          }
        }

        lstLayers.setListData(vctTemp);
      }

      @Override
      public void getImageFormatsError(Exception exception)
      {
        throw new UnsupportedOperationException("Not supported yet.");
      }

      @Override
      public void getImageFormatsCompleted()
      {
        throw new UnsupportedOperationException("Not supported yet.");
      }
```

```
      @Override
      public void getLayerAbstractError(Exception exception)
      {
        throw new UnsupportedOperationException("Not supported yet.");
      }

      @Override
      public void getLayerAbstractCompleted(String layerCode, String
layerAbstract)
      {
        throw new UnsupportedOperationException("Not supported yet.");
      }

      @Override
      public void getMapStarted()
      {
        lblDebug.setText("GetMap iniciado");
      }

      @Override
      public void getMapError(Exception exception)
      {
        lblDebug.setText("Erro no getMap: " + exception.getMessage());
      }

      @Override
      public void getMapCompleted()
      {
        lblDebug.setText("GetMap completado");

        repaint.start();
      }

      @Override
      public void getTileError(Exception exception)
      {
        lblDebug.setText("Erro no getTile: " + exception.getMessage());
      }

      @Override
      public void getTileCompleted(int x, int y, Image tile)
      {
        lblDebug.setText("Tile " + x + ", " + y + " recebido");
      }
    };

    wmsMap.connect();
  }

  /** This method is called from within the constructor to
   * initialize the form.
   * WARNING: Do NOT modify this code. The content of this method is
   * always regenerated by the Form Editor.
   */
  @SuppressWarnings("unchecked")
    private void initComponents() {
```

```
        txtCWC = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        txtGetCapabilities = new javax.swing.JTextField();
        jScrollPane1 = new javax.swing.JScrollPane();
        lstLayers = new javax.swing.JList();
        btnAtualizar = new javax.swing.JButton();
        btnEsquerda = new javax.swing.JButton();
        btnDireita = new javax.swing.JButton();
        btnSubir = new javax.swing.JButton();
        btnDescer = new javax.swing.JButton();
        lblDebug = new javax.swing.JLabel();
        btnCreateMap = new javax.swing.JButton();
        jSeparator1 = new javax.swing.JSeparator();
        pnlMap = new java.awt.Panel();
        btnZoomOut = new javax.swing.JButton();
        btnZoomIn = new javax.swing.JButton();


    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);

        txtCWC.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(java.awt.event.ActionEvent evt) {
            txtCWCActionPerformed(evt);
          }
        });
        getContentPane().add(txtCWC);
        txtCWC.setBounds(380, 30, 210, 20);

        jLabel1.setText("Endereço do CWC");
        getContentPane().add(jLabel1);
        jLabel1.setBounds(380, 10, 150, 14);

        jLabel2.setText("Endereço para getCapabilities");
        getContentPane().add(jLabel2);
        jLabel2.setBounds(10, 10, 210, 14);
        getContentPane().add(txtGetCapabilities);
        txtGetCapabilities.setBounds(10, 30, 360, 20);

        jScrollPane1.setViewportView(lstLayers);

        getContentPane().add(jScrollPane1);
        jScrollPane1.setBounds(10, 60, 140, 260);

        btnAtualizar.setText("Atualizar");
        btnAtualizar.addActionListener(new java.awt.event.ActionListener()
    {
          public void actionPerformed(java.awt.event.ActionEvent evt) {
            btnAtualizarActionPerformed(evt);
          }
        });
        getContentPane().add(btnAtualizar);
        btnAtualizar.setBounds(10, 340, 90, 23);

        btnEsquerda.setText("Esquerda");
        btnEsquerda.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
        btnEsquerdaActionPerformed(evt);
      }
    });
    getContentPane().add(btnEsquerda);
    btnEsquerda.setBounds(110, 340, 100, 23);

    btnDireita.setText("Direita");
    btnDireita.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnDireitaActionPerformed(evt);
      }
    });
    getContentPane().add(btnDireita);
    btnDireita.setBounds(220, 340, 120, 23);

    btnSubir.setText("Subir");
    btnSubir.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnSubirActionPerformed(evt);
      }
    });
    getContentPane().add(btnSubir);
    btnSubir.setBounds(350, 340, 130, 23);

    btnDescer.setText("Descer");
    btnDescer.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnDescerActionPerformed(evt);
      }
    });
    getContentPane().add(btnDescer);
    btnDescer.setBounds(490, 340, 110, 23);
    getContentPane().add(lblDebug);
    lblDebug.setBounds(10, 424, 480, 20);

    btnCreateMap.setText("Criar Mapa");
    btnCreateMap.addActionListener(new java.awt.event.ActionListener()
{
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnCreateMapActionPerformed(evt);
      }
    });
    getContentPane().add(btnCreateMap);
    btnCreateMap.setBounds(490, 420, 110, 23);
    getContentPane().add(jSeparator1);
    jSeparator1.setBounds(0, 400, 610, 10);

    javax.swing.GroupLayout pnlMapLayout = new
javax.swing.GroupLayout(pnlMap);
    pnlMap.setLayout(pnlMapLayout);
    pnlMapLayout.setHorizontalGroup(

pnlMapLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
      .addGap(0, 430, Short.MAX_VALUE)
    );
    pnlMapLayout.setVerticalGroup(
```

```
pnlMapLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
      .addGap(0, 260, Short.MAX_VALUE)
    );

    getContentPane().add(pnlMap);
    pnlMap.setBounds(160, 60, 430, 260);

    btnZoomOut.setText("Zoom -");
    btnZoomOut.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnZoomOutActionPerformed(evt);
      }
    });
    getContentPane().add(btnZoomOut);
    btnZoomOut.setBounds(170, 370, 100, 23);

    btnZoomIn.setText("Zoom +");
    btnZoomIn.addActionListener(new java.awt.event.ActionListener() {
      public void actionPerformed(java.awt.event.ActionEvent evt) {
        btnZoomInActionPerformed(evt);
      }
    });
    getContentPane().add(btnZoomIn);
    btnZoomIn.setBounds(290, 370, 110, 23);

    pack();
  }

    private void btnCreateMapActionPerformed(java.awt.event.ActionEvent
evt)
    {
      createWMSMap();
    }

    private void btnAtualizarActionPerformed(java.awt.event.ActionEvent
evt)
    {
      int index[] = lstLayers.getSelectedIndices();

      if (index.length > 0)
      {
        String layersCode[] = new String[index.length];

        for (int i = 0; i < index.length; i++)
        {
          String layer[] = ((String)
lstLayers.getModel().getElementAt(index[i])).split("#");

          layersCode[i] = "";

          for (int f = 1; f < layer.length; f++)
          {
            if (layersCode[i].length() > 0)
            {
              layersCode[i] += "#";
            }
```

```
            layersCode[i] += layer[f];
          }
        }

        wmsMap.getMap(layersCode);
      }
    }

    private void btnEsquerdaActionPerformed(java.awt.event.ActionEvent
evt)
    {
      wmsMap.moveX(-5);
    }

    private void btnDireitaActionPerformed(java.awt.event.ActionEvent
evt)
    {
      wmsMap.moveX(5);
    }

    private void btnSubirActionPerformed(java.awt.event.ActionEvent
evt)
    {
      wmsMap.moveY(-5);
    }

    private void btnDescerActionPerformed(java.awt.event.ActionEvent
evt)
    {
      wmsMap.moveY(5);
    }

    private void btnZoomOutActionPerformed(java.awt.event.ActionEvent
evt)
    {
      wmsMap.zoomOut();
    }

    private void btnZoomInActionPerformed(java.awt.event.ActionEvent
evt)
    {
      wmsMap.zoomIn();
    }

  /**
   * @param args the command line arguments
   */
  public static void main(String args[])
  {
    java.awt.EventQueue.invokeLater(new Runnable()
    {

      public void run()
      {
        new FrmPrincipal().setVisible(true);
      }
    });
```

```
    }


    private javax.swing.JButton btnAtualizar;
    private javax.swing.JButton btnCreateMap;
    private javax.swing.JButton btnDescer;
    private javax.swing.JButton btnDireita;
    private javax.swing.JButton btnEsquerda;
    private javax.swing.JButton btnSubir;
    private javax.swing.JButton btnZoomIn;
    private javax.swing.JButton btnZoomOut;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JSeparator jSeparator1;
    private javax.swing.JLabel lblDebug;
    private javax.swing.JList lstLayers;
    private java.awt.Panel pnlMap;
    private javax.swing.JTextField txtCWC;
    private javax.swing.JTextField txtGetCapabilities;

}
```

Anexo IV – Servidor (CWC)

```
import mediador.socket.Client;
import mediador.socket.Server;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
/**
 *
 * @author root
 */
public class Main
{

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args)
  {
    Server server = new Server(6400)
    {

      @Override
      public void onStartServerCompleted()
      {
        System.out.println("Servidor iniciado.");
      }

      @Override
      public void onStartServerError(Exception exception)
      {
        System.out.println("Erro iniciando Servidor. " +
exception.getMessage());
      }

      @Override
      public void onClientDisconnected(long index)
      {
        System.out.println("Cliente " + index + " desconectou");
      }

      @Override
      public void onWriteSocketError(Exception exception)
      {
        System.out.println("Erro escrevendo no socket. " +
exception.getMessage());
      }

      @Override
      public void onProcessRequestError(Client client, Exception exception)
      {
        System.out.println(exception.getMessage());
      }
    };

    server.startServer();
  }
}
```