UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JEAN PAULO DA SILVA

ESTUDO DE TECNOLOGIAS PARA DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO GEOGRÁFICA EM AMBIENTE WEB

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

JEAN PAULO DA SILVA

ESTUDO DE TECNOLOGIAS PARA DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO GEOGRÁFICA EM AMBIENTE WEB

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof Claudio Leones Bazzi.

Co-orientador: Prof Pedro L. de Paula Filho

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

ESTUDO DE TECNOLOGIAS PARA DESENVOLVIMENTO DE SISTEMAS DE INFORMAÇÃO GEOGRÁFICA EM AMBIENTE WEB

Por

Jean Paulo da Silva

Este Trabalho de Diplomação (TD) foi apresentado às 15:50 h do dia 23 de novembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram argüidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Claudio Leones Bazzi
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Marcio Angelo Matté
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Nelson Miguel Betzek
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

AGRADECIMENTOS

Agradeço a Deus, amigo fiel e àquele que sempre está ao meu lado me dando sustento e amparo nos momentos em que mais preciso. Sem Ele nada seria possível e se tenho algo de bom foi por que Ele assim me criou.

Gratidão aos pais, José Moreira e Leonice, pelas lições de vida em gestos e palavras, pelas cobranças, pelo incansável apoio e, principalmente, pelos momentos em que eles esqueceram a sua própria vida, os seus próprios sonhos e desejos, para que eu tivesse uma vida o mais feliz possível e para que os meus sonhos e desejos fossem aos poucos se tornando realidade.

Agradeço, de forma especial, a companheira, amiga e namorada Thais pelo amor dedicado em forma de carinho e apoio nos momentos difíceis, por entender minhas ausências e por celebrar comigo nos momentos de vitória.

Por fim, sem excluir ninguém importante, agradeço a todos os amigos, parceiros e professores, “poderosa proteção”, pelos momentos de alegria e descontração, pelos dias de música e partilha, pelas palavras importantes e pelas não tão importantes.

RESUMO

SILVA, Jean P. Estudo de tecnologias para desenvolvimento de sistemas de informação geográfica em ambiente web. 2011. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira. 2011.

Este trabalho apresenta o resultado de pesquisas relacionadas a conceitos importantes necessários para o entendimento da estrutura e desenvolvimento de Sistemas de Informação Geográfica (SIGs). Expõe conceitos úteis de Geoprocessamento, formas de aquisição e principais tipos de representação das informações geográficas aninhados a padrões especificados para o melhor tratamento dos dados espaciais. Aborda os principais sistemas de banco de dados com extensões espaciais utilizados para o armazenamento dos dados descrevendo características e listando as principais funcionalidades para manipulação desses dados. Apresenta os dois principais servidores de mapas, GeoServer e Mapserver, que representam repositórios de dados pré-organizados com opções de estilização das camadas para visualização, servindo de base para aplicações e tecnologias de visualização de mapas. Descreve as funcionalidades, características e componentes da biblioteca OpenLayers e sua integração com outras bibliotecas chamadas *WebGIS*. Por fim, apresenta o *framework OpenLayers for JSF* como opção de integração entre as tecnologias OpenLayers e Java Server Faces destacando vantagens e demonstrando sua utilização através de exemplos. Traz como resultado a implementação de uma aplicação demonstrativa que oferece recursos para criação de geometrias através da integração das tecnologias abordadas e o GoogleMaps bem como a obtenção de dados descritivos referentes à geometria criada.

Palavras-chaves: Sistemas de Informação Geográfica, Geoprocessamento, OpenLayers, OL4JSF, WebGIS.

RESUMO EM LINGUA ESTRANGEIRA

SILVA, Jean P. Estudo de tecnologias para desenvolvimento de sistemas de informação geográfica em ambiente web. 2011. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira. 2011.

This work show the result of researches related to important concepts needed to the understating of the Geographic Information Systems (GIS) structure. Exposes Geoprocessing useful concepts, acquisition forms and the main representation of geographic information types nested into specified patterns for the best spatial data treatment. Approaches the main spatial extended database systems used for data storing, describing its characteristics and listing its main functionalities for data manipulation. Presents the two main map servers, the Geoserver and the Mapserver, which represent pre-organized data repositories with layer stylization for visualization, being base for applications and map visualization technologies. Describes the functionalities, characteristics and components of OpenLayers libraries and its integration with others libraries called WebGIS. Lastly, presents the OpenLayers framework for JSF as integration option between the OpenLayers and JavaServer Faces technologies, highlighting the advantages and presenting its utilization trough examples. Brings as result a demonstrative application implementation which offers resources for geometries creation trough the previously approached technologies and Google Maps integration well as descriptive data acquisition related to the created geometry.

Keywords: Geographical Information Systems, GIS, OpenLayers, OL4JSF, WebGIS.

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

AJAX	Asynchronous JavaScript XML
API	*Application Programming Interface*
CGI	*Common Gateway Interface*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
ESRI	*Environmental Systems Research Institute*
GeoJSON	*JavaScript Object Notation Geometry*
GeoRSS	*Really Simple Syndication Geometry*
GIF	*Graphic Interchange Format*
GIS	*Geography Information Systems*
GML	*Graffiti Markup Language*
GPS	*Sistema Global de Posicionamento*
IDE	*Integrated Development Environment*
INPE	*Instituto Nacional de Pesquisas Espaciais*
ISO	*International Organization for Standardization*
JEE	*Java Enterprise Edition*
JPEG	*Joint Picture Expert Group*
JSF	*JavaServer Faces*
KML	*Keyhole Markup Language*
MDSYS	*Multi Dimensional System*
MVC	*Model-View-Controller*
MVC	*Model-View-Controller*
ODBC	*Open DataBase Connectivity*
OGC	*Open Geospatial Consortium*
OL4JSF	*OpenLayers for JSF*
PHP	*Hypertext Preprocessor*
PNG	*Portable Network Graphics*
SDO	*Spatial Data Option*
SFS	*Shared File System*
SGBD	*Sistema de Gerenciamento de Bando de Dados*
SGBDOR	*Sistema de Gerenciamento de Banco de Dados Objeto-Relacional*

SIG	*Sistema de Informação Geográfica*
SLD	*Styled Layer Descriptor*
SQL	*Structured Query Language*
SRID	*Spatial Reference Identifier*
SVG	*Scalable Vector Graphics*
URL	*Universal Resource Locator*
WCS	*Web Coverage Service*
WebGIS	*Web Geography Information System*
WFS	*Web Feature Service*
WFS-T	*Web Feature Service Transactions*
WKB	*Well-Know Binary*
WKT	*Well-Know Text*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, a ascensão da Internet e de tecnologias voltadas à mobilidade de aplicações aliado às necessidades dos usuários e corporações instiga os desenvolvedores a oferecerem soluções baseadas nessas tecnologias. Não poderia ser diferente com os Sistemas de Informação Geográfica. Esses sistemas, que gerenciam a manipulação da informação geográfica, auxiliam na tomada de decisão e na confecção de materiais geográficos, quando unidos às vantagens características dessas tecnologias, se transformam em ferramentas poderosas e ao mesmo tempo flexíveis.

Os SIGs-WEB possuem todas as principais características e funcionalidades de um SIG normal, porém apresentam como diferencial a flexibilidade e a possibilidade de integração com soluções disponíveis na web, como o GoogleMaps por exemplo.

Portanto, a intenção desse estudo é expor de forma detalhada as principais tecnologias utilizadas para o desenvolvimento de SIGs com base na visualização por meio do ambiente web.

1.1 OBJETIVO GERAL

Realizar uma pesquisa relacionada às tecnologias que auxiliam no desenvolvimento de Sistemas de Informação Geográfica disponibilizados em ambiente web e apresentar exemplos de aplicação.

1.2 OBJETIVOS ESPECÍFICOS

- Pesquisar os principais conceitos e características de geoprocessamento aplicados nos SIGs;
- Detalhar os tipos de dados geográficos e as formas de aquisição, tradução e representação;

- Realizar uma pesquisa relacionada a bancos de dados geográficos, padrões OGC e os principais sistemas que possuem extensão geográfica;
- Realizar um estudo direcionado às principais tecnologias para disponibilização de dados espaciais na web;
- Implementar uma ferramenta de teste dos recursos das tecnologias apresentadas.

1.3 JUSTIFICATIVA

De uma forma geral, a utilização da informação geográfica implica em tarefas trabalhosas, em análises complexas e exigem dedicação e prazos muito extensos. Engloba diversas áreas de conhecimento que necessitam de material cartográfico e dados estatísticos para tomadas de decisão como a própria Cartografia e as áreas de Transportes, Engenharia, Planejamento Urbano, Recursos Naturais e Comunicação.

É nesse contexto que pode ser visualizado a importância dos SIGs para essas áreas corporativas e institucionais. Eles automatizam processos, diminuem prazos, descomplicam tarefas e análises e de forma especial atuam na confecção de mapas significativos e muito próximos da realidade, seguindo o conceito de abstração.

Os SIGs, na sua forma tradicional, já são amplamente utilizados nessas áreas citadas a um bom tempo de forma robusta e confiável. Porém, o desafio atual é que essas aplicações se tornem cada vez mais flexíveis e integradas com aplicações ricas através da Internet e dos dispositivos móveis.

Portanto, um estudo direcionado às tecnologias que já possuem essas características contribui na sua propagação e desperta o interesse de profissionais da área de TI de pesquisar e desenvolver soluções.

1.4 ESTRUTURA DO TRABALHO

O trabalho foi dividido em cinco capítulos conforme o conteúdo de estudo e as conclusões obtidas:

O capítulo dois aborda o conceito de geoprocessamento e define a estrutura geral de um SIG. Apresenta os dados geográficos como peças principais no desenvolvimento de SIGs, mostrando desde sua forma de aquisição e tradução para o ambiente computacional até as principais e mais utilizadas formas de representação.

O capítulo três traz as principais tecnologias que compõem um ambiente SIG-WEB, desde a base de dados, servidores de mapas até as bibliotecas clientes de visualização. Aborda o framework *OpenLayers for JSF* (OL4JSF) como integração entre as bibliotecas de visualização e a linguagem Java através do padrão *Model-View-Controller* (MVC).

No capítulo quatro é detalhada a aplicação de testes que foi implementada utilizando OL4JSF, servidor de mapas (Geoserver) e banco de dados geográfico (PostgreSQL/PostGIS).

O quinto capítulo reúne as conclusões obtidas através do conteúdo pesquisado.

2 SISTEMAS DE INFORMAÇÃO GEOGRÁFICA

2.1 INTRODUÇÃO

Os Sistemas de Informação Geográfica (SIGs), do inglês *Geographic Information System* (GIS), são ferramentas computacionais utilizadas na manipulação das informações geográficas, permitindo realizar análises complexas, integração de dados de diversas fontes, criação de bancos de dados georreferenciados e ainda a automatização na produção de documentos cartográficos (CÂMARA;DAVIS;MONTEIRO, 2001, p. 1).

De uma forma mais abrangente, Ozemoy, Smith e Sicherman (1981, p. 92) definem um SIG como "um conjunto de funções automatizadas, que fornecem aos profissionais, capacidades avançadas de armazenamento, acesso, manipulação e visualização de informação georreferenciada".

> O termo Sistemas de Informação Geográfica (SIG) é aplicado para sistemas que realizam o tratamento computacional de dados geográficos e recuperam informações não apenas com base em suas características alfanuméricas, mas também através de sua localização espacial; oferecem ao administrador (urbanista, planejador, engenheiro) uma visão inédita de seu ambiente de trabalho, em que todas as informações disponíveis sobre um determinado assunto estão ao seu alcance, interrelacionadas com base no que lhes é fundamentalmente comum – a localização geográfica (CAMARA;QUEIROZ. 2001, p1).

Relacionado a seu objetivo principal, pode-se dizer que SIG apóia a tomada de decisão, "envolvendo integração de informação georreferenciada num ambiente de resolução de problemas" (COWEN, 1988, p. 1554).

Os SIGs são parte integrante e fundamental na disciplina de Geoprocessamento caracterizada por ser uma "disciplina do conhecimento que utiliza técnicas matemáticas e computacionais para o tratamento da informação geográfica", englobando e influenciando diversas áreas como a cartografia, análise de recursos naturais, transportes, comunicação, energia e planejamento urbano e regional (CÂMARA;DAVIS;MONTEIRO, 2001, p. 1).

O termo geoprocessamento implica na fusão entre o tratamento de informações geográficas com a utilização de soluções tecnológicas. O seu crescimento está totalmente ligado ao surgimento de inovações na área tecnológica, principalmente na área da informática.

Uma frase muito conhecida que define de forma sucinta o geoprocessamento é: “Se onde é importante para seu negócio, então geoprocessamento é sua ferramenta de trabalho” (CÂMARA;DAVIS;MONTEIRO, 2001, p. 1).

Dentre as funcionalidades de um SIG, destacam-se:

- Entrada e validação de dados espaciais;
- Armazenamento e gerenciamento desses dados;
- Saída e apresentação visual desses dados;
- Transformação de dados espaciais;
- Interação com o usuário;
- Combinação de dados espaciais para criar novas representações do espaço geográfico;
- Ferramentas para análise espacial.

2.2 ESTRUTURA DE UM SIG

A estrutura geral de um SIG (Figura 1) é composta inicialmente por uma estrutura de dados geográficos, gerenciada por ferramentas que oferecem funções e condições para as devidas análises e consultas que poderão ser efetuadas.

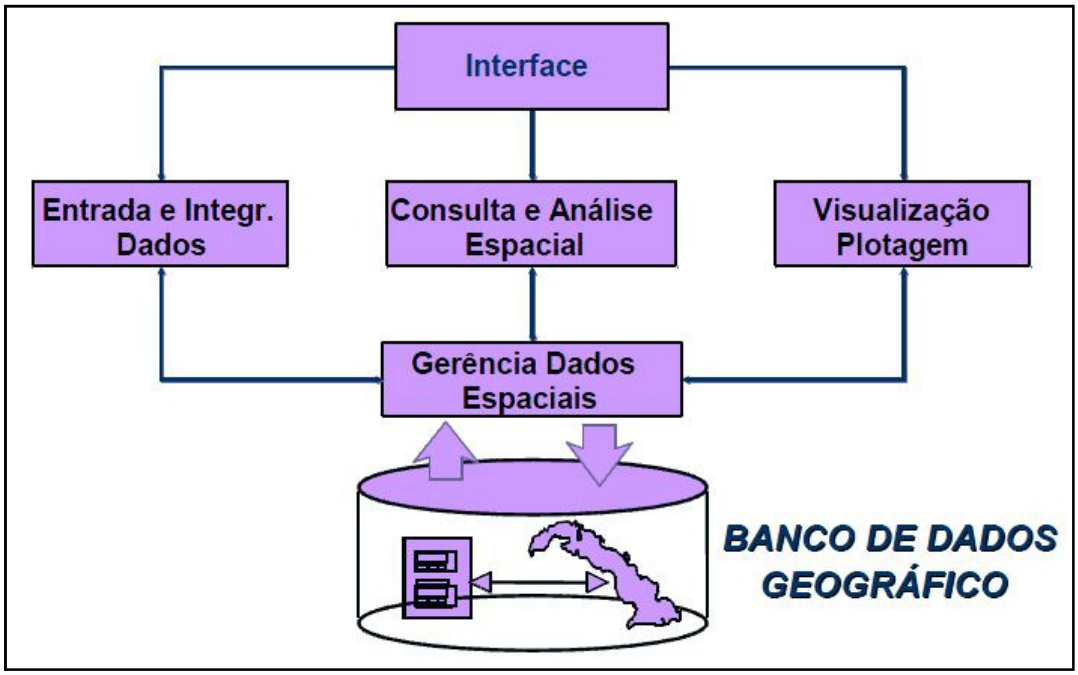

Figura 1 - Estrutura de um SIG
Fonte: INPE

A camada de Entrada e Integração de Dados corresponde à inclusão e alteração de dados geográficos, enquanto que a camada de Visualização e Plotagem corresponde à disponibilização dos dados em forma de camadas, fornecendo o resultado em forma de imagem e coerente com o esperado pelo usuário. A Interface corresponde ao conjunto de componentes gráficos utilizados para a entrada, manipulação, consulta, análise e visualização dos dados.

Partindo de sua estrutura básica, é possível indicar suas principais características, segundo Casanova et al. (2005, p. 2-3):

- Inserir e integrar, numa única base de dados, informações espaciais provenientes de meio físico-biótico, de dados censitários, de cadastros urbano e rural, e outras fontes de dados como imagens de satélite, e GPS.
- Oferecer mecanismos para combinar as várias informações, através de algoritmos de manipulação e análise, bem como para consultar, recuperar e visualizar o conteúdo da base de dados geográficos.

2.3 DADOS GEOGRÁFICOS

Os dados geográficos são os principais componentes de um SIG. Um dado geográfico representa a localização geográfica de uma entidade contida em um sistema de coordenadas. Portanto, utilizando-se de dados geográficos é possível representar áreas, rodovias, rios, enfim, entidades geográficas das mais diversas formas.

> Do ponto de vista da aplicação, o uso de sistemas de informação geográfica (SIG) implica em escolher as representações computacionais mais adequadas para capturar a semântica de seu domínio de aplicação. Do ponto de vista da tecnologia, desenvolver um SIG significa oferecer o conjunto mais amplo possível de estruturas de dados e algoritmos capazes de representar a grande diversidade de concepções do espaço (CASANOVA et al., 2005, p. 4).

Segundo Ferreira (2006, p. 15), “para que seja possível produzir informações geográficas, é necessário ‘alimentar’ os computadores e os programas computacionais de SIG com dados sobre o mundo real”. Ou seja, os dados geográficos constituem-se em uma abstração de atributos geográficos presentes no mundo real que são obtidos, transformados e fornecidos para uma determinada utilidade dentro de um determinado SIG.

2.3.1 Tradução da Informação Geográfica para o computador

Essa abstração é chamada por Câmara, Davis e Monteiro (2001, p-2) de um "processo de tradução do mundo real para o ambiente computacional". Inspirado pelo "paradigma dos quatro universos" de Gomes e Velho (1995), ele adapta esses quatro universos para a área da geoinformação. Os quatro universos correspondem a quatro passos ou níveis de abstração para representar computacionalmente um conceito do mundo real: universo do *mundo real*, universo *matemático*, universo de *representação* e universo de *implementação*.

Apesar desse paradigma não se limitar a sistemas de Geoprocessamento, sua aplicação a esses sistemas é apropriada por permitir equacionar os problemas da área (CÂMARA;DAVIS;MONTEIRO. 2001, p-3).

Dentro do universo do mundo real ou ontológico como é conhecido, é realizado um levantamento dos conceitos do mundo real a serem representados no computador (OLIVEIRA. 2009, p-22). "O projeto de um sistema de informação requer, como passo inicial, a escolha das entidades a ser representadas e, se possível, a descrição organizada destas entidades por meio de conceitos" (CÂMARA. 2005, p-8). Esses conceitos podem ser divididos em dois tipos básicos:

- Conceitos físicos – correspondem a fenômenos físicos do mundo real;
- Conceitos sociais – conceitos criados para representar entidades sociais e institucionais.

O universo matemático ou formal corresponde à etapa de formalização dos conceitos obtidos no universo ontológico, gerando um modelo lógico do sistema a ser desenvolvido (OLIVEIRA. 2009, p-22). É considerado um componente intermediário entre os conceitos do universo ontológico e as estruturas de dados e algoritmos computacionais (CÂMARA. 2005, p-10).

> No universo formal, buscamos estabelecer um conjunto de entidades lógicas que agrupem os diferentes conceitos da ontologia de aplicação da forma mais abrangente possível. Adicionalmente, neste universo definimos ainda como serão associados valores aos diferentes conceitos; ou seja, como podemos medir o mundo real (CÂMARA. 2005, p-10-11).

No universo de representação ou estrutural ocorre o mapeamento das entidades geradas no universo formal para estruturas de dados geométricas e alfanuméricas. Nessa fase, pode se considerar duas formas de representação: Representação Vetorial e Representação Matricial.

Na representação vetorial, opta-se por representar o elemento ou objeto da forma mais exata possível, utilizando-se de três formas básicas de representação: pontos, linhas, áreas ou polígonos.

Já na representação matricial ocorre a representação em forma de matriz, construída célula a célula, utilizada geralmente na representação de imagens de sensoriamento remoto e alguns tipos de modelo numérico de terreno.

O quarto e último universo compreende a implementação efetiva do projeto, chamado de universo implementado. Nesse nível serão indicadas quais as estruturas de dados que serão utilizadas para construir o sistema, as decisões concretas relacionadas à programação (CÂMARA;DAVIS;MONTEIRO. 2001, p-31).

2.4 MODELOS DE REPRESENTAÇÃO DE DADOS GEOGRÁFICOS

Como mencionado anteriormente, existem duas estruturas predominantes de representação computacional dos dados geográficos: a representação matricial e a representação vetorial.

2.4.1 Representação Matricial

Também conhecido como *"raster"*, representa o espaço em forma de matriz de linhas e colunas, "onde cada célula possui um número de linha, um número de coluna e um valor correspondente ao atributo estudado e cada célula é individualmente acessada pelas suas coordenadas" (CAMARA;DAVIS;MONTEIRO. 2001, p-17).

O espaço nesse caso é tratado como uma superfície plana e cada célula representa uma porção do terreno. A resolução é obtida através da relação entre as dimensões da célula e área total do terreno (Figura 2).

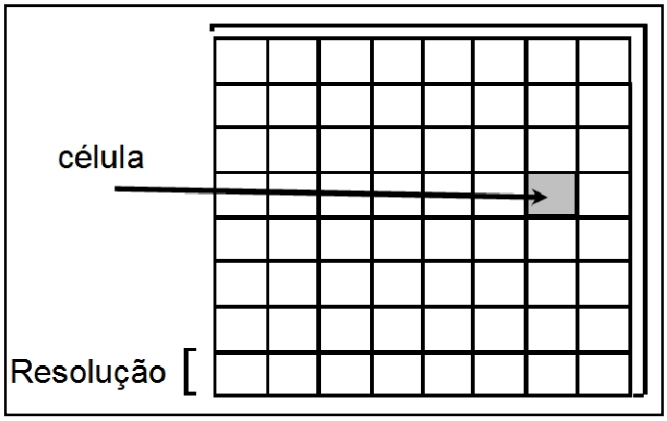

Figura 2 - Estrutura matricial
Fonte: Câmara, 2005

A Figura 3 traz um comparativo entre representações matriciais de resolução diferente. Quanto maior a resolução, maior o nível de exatidão e em contrapartida, proporcionalmente maior será o espaço de armazenamento.

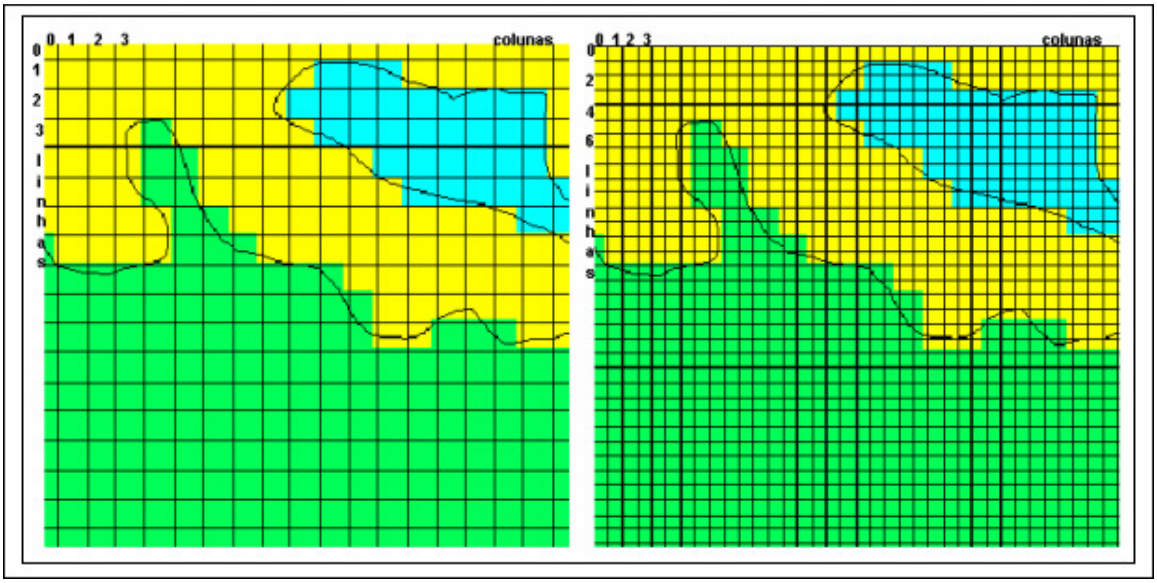

Figura 3 - Diferentes representações matriciais para um mapa
Fonte: Câmara e Monteiro, 2001.

2.4.2 Representação Vetorial

O modelo vetorial tem por principal característica a representação dos objetos, sua localização e aparência, através de um ou mais pares de coordenadas. Nesse modelo, destacam-se três elementos gráficos: Ponto, Linha e Polígono (área).

Um ponto representa um par ordenado de coordenadas espaciais (x, y) que pode possuir aninhado a si atributos não espaciais. Um exemplo de ponto no modelo

vetorial seria uma antena de sinal celular, localizada numa coordenada *x* e *y* que possui como atributo o nome da operadora.

As linhas são constituídas de uma coleção de pontos interligados. Exemplos da representação com linhas são as rodovias, limites políticos e rios.

A forma de representação através de polígonos é recomendada quando se deseja representar regiões (municípios, estados, terrenos, etc.) e é formada por linhas conectadas aonde a coordenada do último ponto seja igual ao do primeiro (Figura 4).

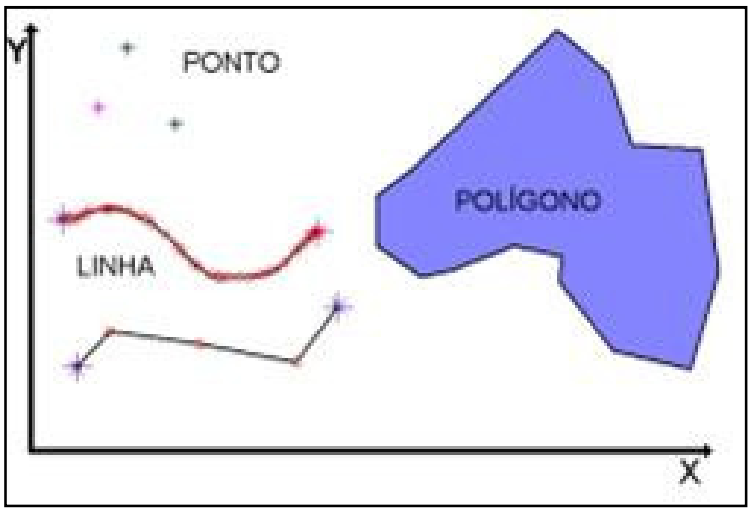

Figura 4 - Elementos de representação vetorial
Fonte: Câmara e Monteiro, 2001.

Os polígonos podem ser classificados em três formas diferentes de utilização: isolados, aninhados e adjacentes. Quando classificados como isolados, significa que os objetos não se tocam, por exemplo, quando uma imobiliária quer representar geograficamente os terrenos que estão disponíveis para venda.

O caso dos polígonos aninhados pode ser compreendido quando temos um polígono dentro de outro, onde os seus limites não se tocam, dando uma impressão de estarem empilhados, caso típico das curvas de níveis e isolinhas.

Enfim, os objetos adjacentes são os que podem compartilhar fronteiras com outros polígonos, como por exemplo, as divisões territoriais de municípios, mapas geológicos e temáticos.

As linhas também podem ser classificadas em três tipos: isoladas, em árvore e em rede. As linhas isoladas, como o nome diz, são representadas sem ligação com outras linhas. As árvores representam linhas que possuem ramificações, muito utilizada na apresentação de rios e seus afluentes. As redes representam as linhas interconectadas, por exemplo, redes elétricas, telefônica, rodovias e ferrovias.

2.4.3 Comparação entre representações matriciais e vetoriais

As principais vantagens na utilização de um modelo vetorial estão na maior precisão e no menor espaço necessário para armazenamento quando comparado ao modelo matricial (Figura 5). Já quando o interesse consiste na facilidade de implementação das operações de álgebra, o modelo matricial tem vantagem sobre o vetorial.

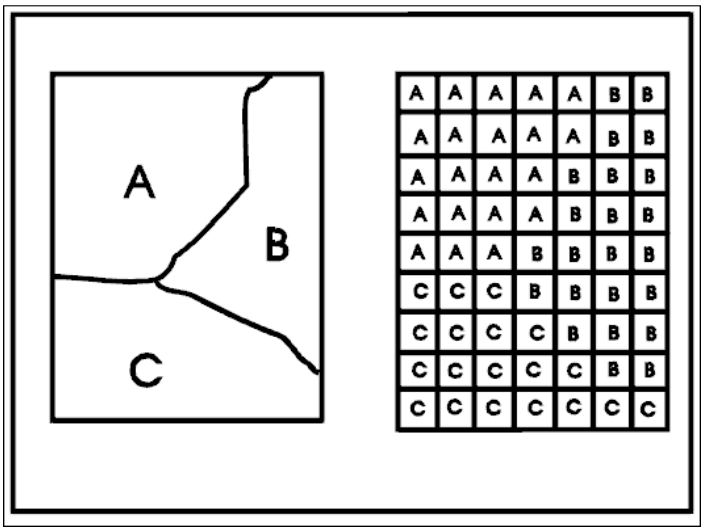

Figura 5 - Representação Vetorial e Matricial de um mapa temático
Fonte: Câmara e Monteiro, 2001.

A Tabela 1 apresenta uma comparação entre os dois tipos de representação, apontando vantagens e desvantagens para mapas temáticos.

Tabela 1 - Comparação entre representações para mapas temáticos - 2001

Aspecto	Representação Vetorial	Representação Matricial
Relações espaciais entre objetos	Relacionamentos topológicos entre objetos disponíveis	Relacionamentos espaciais devem ser inferidos
Ligação com banco de dados	Facilita associar atributos a elementos gráficos	Associa atributos apenas a classes do mapa
Análise, Simulação e Modelagem	Representação indireta de fenômenos contínuos Álgebra de mapas é limitada	Representa melhor fenômenos com variação contínua no espaço Simulação e modelagem mais fáceis
Escalas de trabalho	Adequado tanto a grandes quanto a pequenas escalas	Mais adequando para pequenas escalas (1:25.000 e menores)
Algoritmos	Problemas com erros geométricos	Processamento mais rápido e eficiente
Armazenamento	Por coordenada (mais eficiente)	Por matrizes

Fonte: Câmara e Monteiro, 2001.

2.4.4 Shapefile

Shapefile é um formato de arquivos de vetor que reúne informações de um conjunto de coordenadas geométricas e informações associadas atribuídas para uma característica espacial.

Desenvolvido e mantido pelo Environmental Systems Research Institute (ESRI) o *shapefile* é um "formato de arquivo proprietário que mantém uma interoperabilidade com diversos outros softwares, fazendo com que ganhe grande popularidade" (ESRI, 1998).

Ele é composto por três tipos de arquivos:

- .shp – um arquivo que estoca as características geométricas;
- .shx – um arquivo que estoca o índice das características geométricas;
- .dbf – um arquivo *dBASE* que estoca informações dos atributos e características.
- .prj – armazena informações sobre o sistema de coordenadas (opcional)

2.5 AQUISIÇÃO DE DADOS GEOGRÁFICOS

Como visto até então, o objeto mais importante quando tratamos de geoprocessamento e SIGs é a informação geográfica. Para que seja possível a manipulação de dados geográficos, antes é preciso que estes dados estejam disponíveis.

> "Para que o SIG cumpra suas finalidades, há a necessidade de dados. A aquisição de dados em Geoprocessamento deve partir de uma definição clara dos parâmetros, indicadores e variáveis, que serão necessários ao projeto a ser implementado" (FUNDAMENTOS...)

Atualmente, a tecnologia presta um grande serviço ao auxiliar os profissionais da área na coleta de dados geográficos. Mesmo assim, essa aquisição ainda é considerada uma tarefa difícil e trabalhosa.

Segundo Ferreira (2006, p-25), a abstração do mundo real a fim de se obter dados geográficos em quantidade e qualidade exige a utilização de ciências, tecnologias, técnicas e instrumentos adequados.

As principais fontes de informações espaciais são oriundas dos seguintes meios de coletas de dados:

- Levantamento de campo – com o auxílio de GPS é possível obter as propriedades espaciais referente a uma determinada localização;
- Aerofotogrametria – aeronave equipada com câmeras fotográficas métricas com o objetivo de fotografar verticalmente uma determinada;
- Mapas – podem ser digitais ou não.

3 TECNOLOGIAS GEOESPACIAIS

Para que seja o possível desenvolvimento de um SIG, se faz necessário um estudo aprofundado das tecnologias que podem ser envolvidas e quais recursos elas oferecem para otimizar processos de manipulação de dados geográficos.

As tecnologias que serão apresentadas compreendem desde a parte de armazenamento dos dados geográficos em bases de dados robustas até a manipulação, consulta e análise. Seguindo a estrutura básica de um SIG proposta anteriormente, os dados podem ser visualizados através de uma interface com o usuário.

No caso específico deste trabalho, as tecnologias de visualização serão voltadas para sistemas *web*, sendo, portanto, constituídas de servidores de mapas e apresentadas em navegadores.

3.1 BANCOS DE DADOS GEOGRÁFICOS

Caracteriza-se por banco de dados geográfico o SGBD que possui dentre suas características a capacidade de armazenar e manter a integridade de dados geoespaciais e descritivos, bem como fornecer uma linguagem para manipulação e consulta dos dados através da linguagem SQL (*Structured Query Language*) (OLIVEIRA, 2009, p.42).

Com objetivo de convencionar estruturas de armazenamento e manipulação de dados espaciais, a Open Geospatial Consortium (OGC) desenvolveu o padrão OpenGIS SFS (*Simple Features Specification*).

Os dois principais SGBDs que possuem extensões que implementam esse padrão são o PostgreSQL, através da extensão PostGIS, e o Oracle com a extensão OracleSpatial.

3.1.1 PostgreSQL e PostGIS

O PostgreSQL é considerado um Sistema de Gerenciamento de Banco de Dados Objeto-Relacional (SGBDOR), ou seja, além de possuir todos os atributos de Sistema Gerenciador de Banco de Dados (SGBD) relacional, oferece aos

desenvolvedores a capacidade de integrar ao banco de dados seus próprios tipos de dado e métodos personalizados (POSTGRESQL, 2005).

Caracterizado por ser um sistema robusto, confiável, flexível e rico em recurso, o PostgreSQL possui código aberto e atualmente está na versão 9.1. Fornece suporte completo a chaves estrangeiras, junções (*JOINs*), visões, gatilhos e procedimentos armazenados (em múltiplas linguagens). Inclui a maior parte dos tipos de dados do ISO SQL:1999, incluindo *INTEGER*, *12 NUMERIC*, *BOOLEAN*, *CHAR*, *VARCHAR*, *DATE*, *INTERVAL*, e *TIMESTAMP*. Suporta também o armazenamento de objetos binários, incluindo figuras, sons ou vídeos. Possui interfaces nativas de programação para C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre outros, e uma excepcional documentação (POSTGRESQL, 2005).

Os desenvolvedores podem realizar tarefas de administração através da ferramenta pgAdmin III (Figuras 7 e 8), que possui uma interface gráfica para atender a todas as funcionalidades do PostgreSQL.

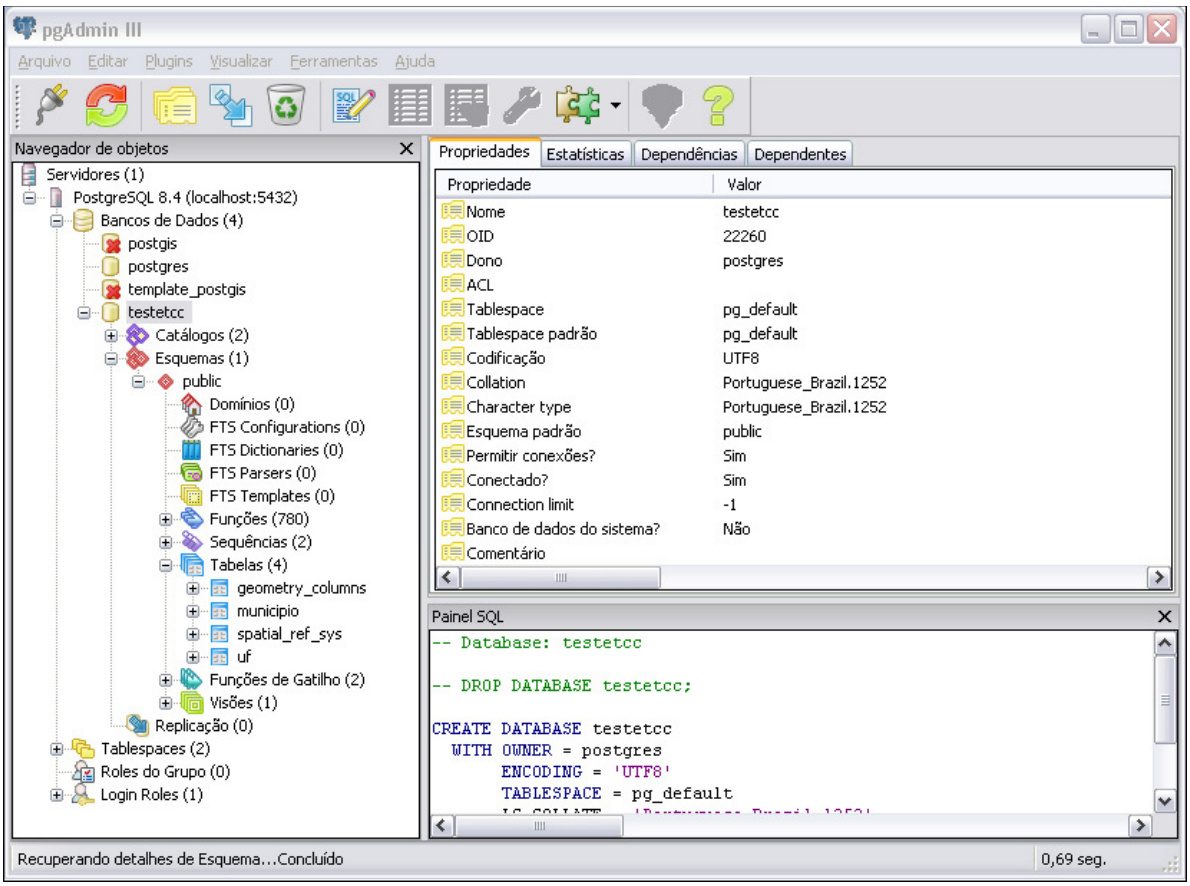

Figura 6 - Tela do pgAdmin III

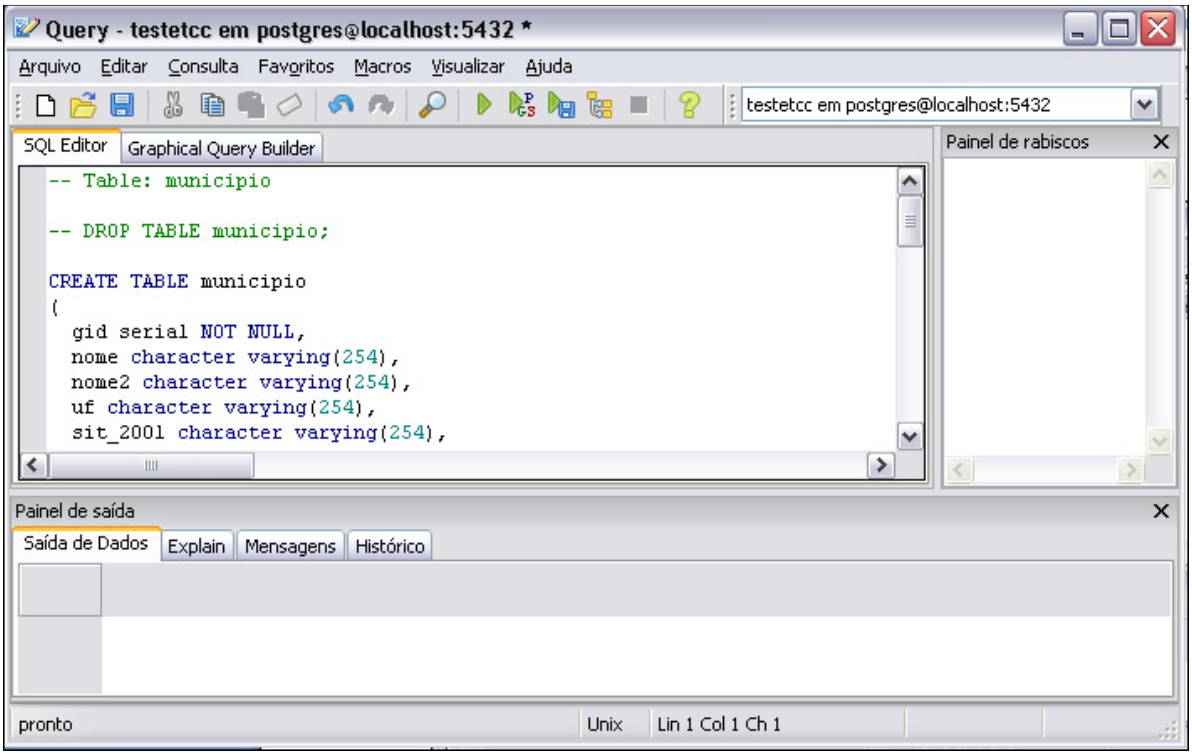

Figura 7 - Editor SQL do pgAdmin III

O PostgreSQL suporta aplicações voltadas para geoprocessamento por incluir em suas funcionalidades uma extensão para armazenamento e manipulação de dados geográficos: o PostGIS.

Desenvolvido pela *Refractions Research Inc.* como uma tecnologia de banco de dados espacial no projeto de pesquisa, o PostGIS segue o padrão de interoperabilidade da OGC.

Por padrão, o PostGIS possui um vasto conjunto de “funções para análise e processamento de objetos GIS” (POSTGIS, 2010).

3.1.1.1 Objetos de GIS

Segundo a especificação OGC, as bases de dados devem oferecer suporte objetos de GIS. O PostGIS suporta os seguintes tipos de geometrias especificados pela OGC: *Point*, *Linestring*, *Polygon*, *Multipoint*, *Multilinestring*, *Multipolygon* e *GeometryCollection* (Figura 9).

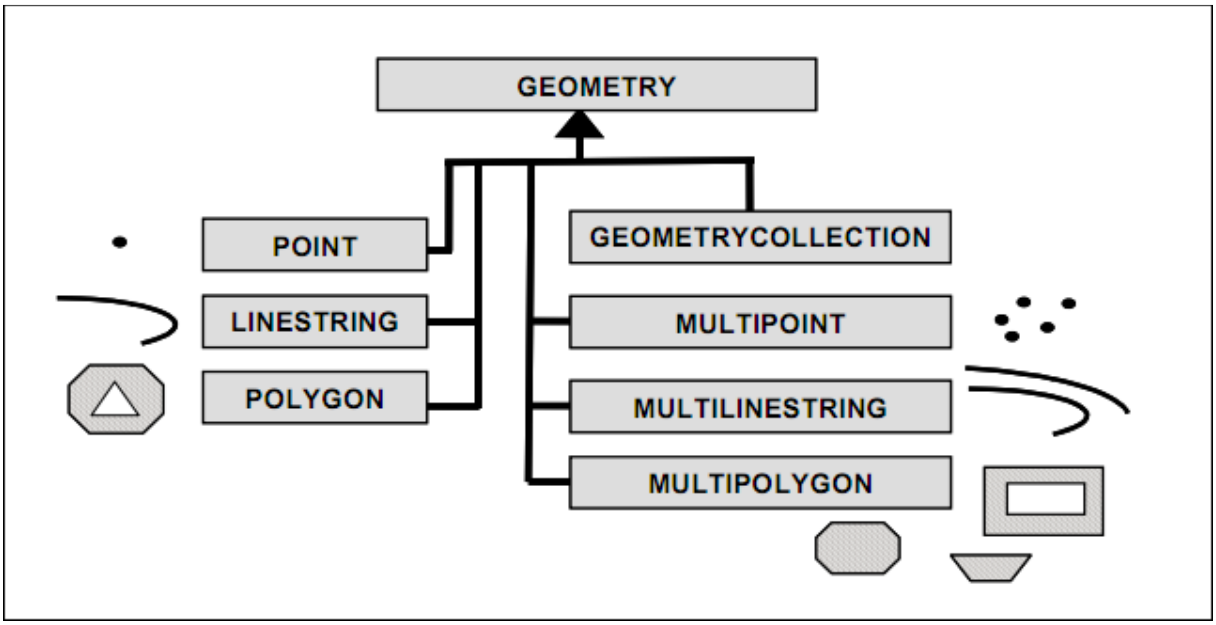

Figura 8 – Estrutura dos tipos de geometria suportados pelo PostGIS
Fonte: Quadros (2010)

Há duas formas de para representação de objetos GIS especificados pela OGC e implementados pelo PostGIS: *Well-Know Text* (WKT) e *Well-Know Binary* (WKB).

Na primeira forma de representação, a sintaxe para manipulação dos objetos é feita em formato de texto, formada da descrição do tipo da geometria seguida das suas coordenadas. Alguns exemplos a seguir:

- POINT (3, 5)
- LINESTRING (0 0, 1 1, 1 2)
- POLYGON ((0 0, 5 0, 5 5, 0 5, 0 0),(1 1, 2 1, 2 2, 1 2, 1 1))
- MULTIPOINT (0 0, 1 2)
- MULTILINESTRING ((0 0, 2 3, 2 5), (2 3, 3 2, 5 4))
- MULTIPOLYGON (((0 0, 5 0, 5 5, 0 5, 0 0),(1 1, 2 1, 2 2, 1 2, 1 1)), ((-1 -1, -1 -2, -2 -2, -2 -1, -1 -1)))
- GEOMETRYCOLLECTION(POINT(4 3), LINESTRING (0 0, 1 1, 1 2))

A especificação OpenGIS requer que o formato de armazenamento interno dos objetos geográficos inclua um identificador referente à localização do objeto no espaço. Esse identificador, conhecido como *Spatial Reference System Identifier* (SRID), referencia uma tabela presente no PostGIS e criada automaticamente chamada *spatial_ref_sys*, que guarda informações sobre coordenadas e projeções utilizadas em mapas.

O Quadro 1 demonstra um comando SQL de inserção simples de um dado geográfico. A função *GeomFromText* aceita como parâmetro a definição da geometria em formato WKT e o SRID correspondente, e devolve a geometria a ser inserida no banco de dados.

```
insert into teste (the_geom, nome)
values (GeomFromText('POINT(-135.4 28.89)', 4326), 'Um lugar')
```

Quadro 1 - Exemplo de inserção de dados espaciais

A forma de representação em WKB manipula dados em forma binária (*bytes*). O Quadro 2 apresenta algumas funções para manipulação de dados representados em WKB.

```
bytea wkb = asBinary(geometry);
geometry the_geom = GeomFromWKB(bytea wkb, SRID);
```

Quadro 2 - Funções para manipulação de dados geográficos em formato WKB

3.1.1.2 Criação de tabelas na base de dados com PostGIS

A partir do momento em que se cria uma base de dados no PostgreSQL com extensão PostGIS duas tabelas são automaticamente criadas para auxiliar nas operações com dados espaciais: *spatial_ref_sys* e *geometry_columns*.

- *spatial_ref_sys*: contém os identificadores (SRIDs) referentes à projeção a ser escolhida. Eles são usados no momento da criação das tabelas, quando da criação do campo geográfico, bem como no momento da inserção de dados. Caso em nenhum momento o SRID seja informado, é assumido o valor -1 (Figura 10).

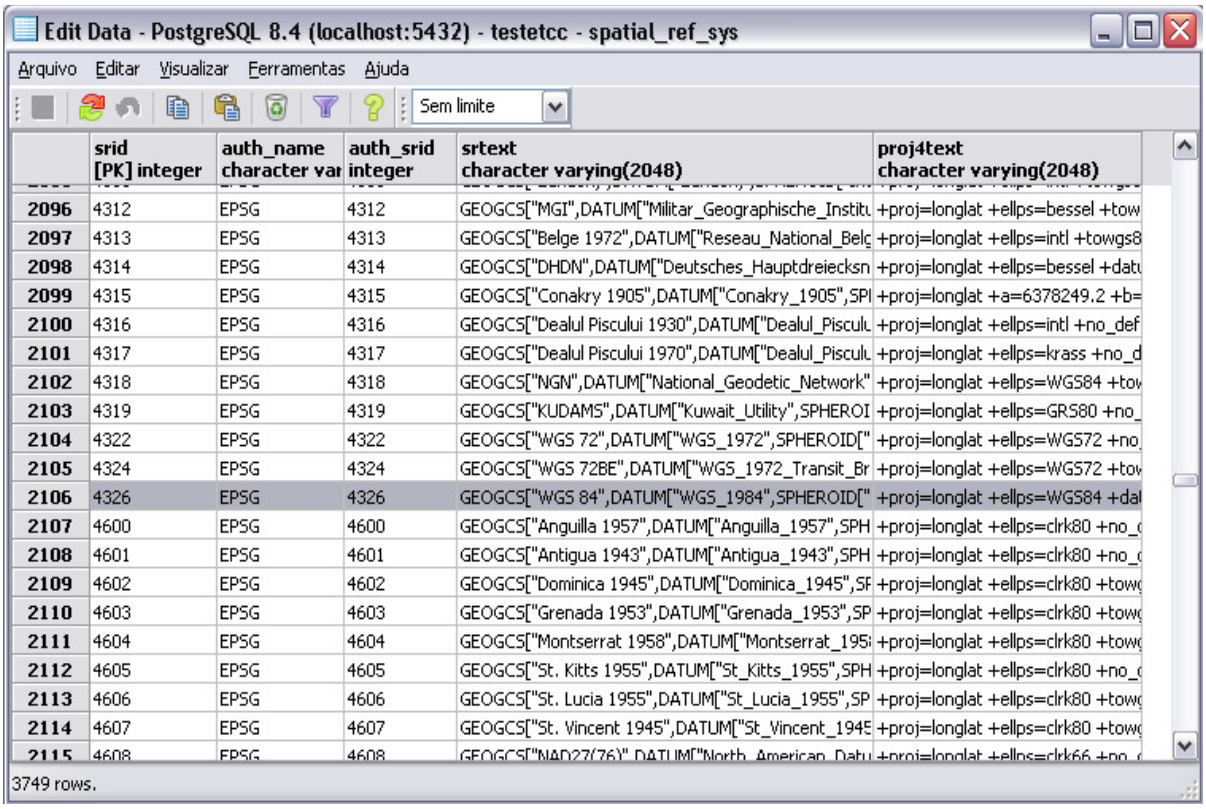

Figura 9 - Tabela spatial_ref_sys

- *geometry_columns*: esta tabela identifica todos os campos geométricos existentes no banco de dados, servindo como padrão de leitura para a maioria dos programas (PALMARES GEOPROCESSAMENTO, 2006) (Figura 11).

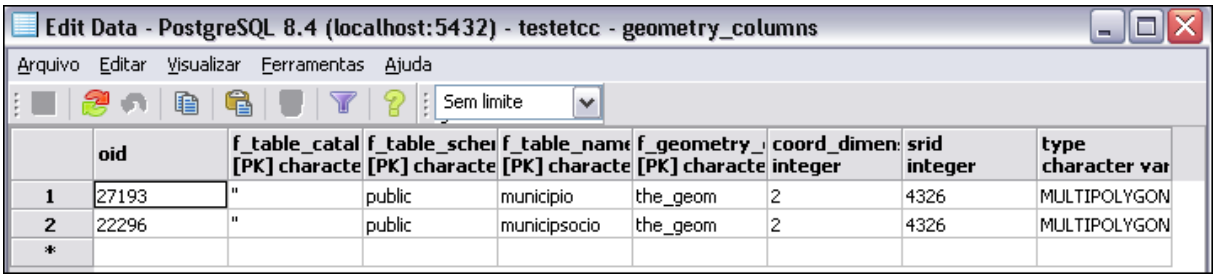

Figura 10 - Tabela geometry_columns

Para a criação de uma tabela com campos geográficos a sintaxe é a mesma da utilizada em SGBDs comuns, exceto pelo uso da função OpenGIS *AddGeometryColumn* (Quadro 3).

```
create or replace table geometrias(id int4, nome varchar(60);
select AddGeometryColumn("testestcc", "geometrias", "the_geom",
                         4326, 'MULTIPOLIGON', 2);
```

Quadro 3 - Exemplo de criação de tabela com campo geográfico

Sintaxe:

- AddGeometryColumn(<schema_name>, <table_name>, <column_name>, <srid>, <type>, <dimension>)

Caso o parâmetro referente ao *schema* seja omitido, assume-se o corrente. O parâmetro *type* pode ser informado como um tipo *geometry*, considerado genérico para todos os tipos de geometrias.

3.1.1.3 Funções OpenGIS

O PostGIS implementa funções especificadas pela OGC para a manipulação de dados geográficos. Por padrão, após a instalação da extensão, são incorporadas um total de 780 funções, sendo possível a adição de funções.

As funções refletem desde operações DML (Linguagem de Manipulação de Dados) e DDL (Linguagem de Definição de Dados), até consultas e análises referentes aos diversos relacionamentos entre os objetos espaciais.

Através delas é possível recuperar, por exemplo, fatores como distância entre duas geometrias; área, comprimento e centróide de uma geometria; relacionamentos de intersecção, cruzamento e sobreposição, dentre outros vários fatores.

Dentre as principais funções, destacam-se as seguintes:

- **Funções de Gerenciamento:**
  - *AddGeometryColumn(varchar,varchar,varchar,integer,varchar,integer)*: acrescenta uma coluna geométrica a uma tabela existente;
  - *DropGeometryColumn(varchar,varchar,varchar)*: remove uma coluna geométrica de uma tabela existente;
  - *SetSRID(geometry)*: atribui um SRID a uma geometria. É utilizado na construção de consultas.
- **Funções de Relacionamento da Geometria:**
  - *Distance(geometry,geometry)*: retorna a distância cartesiana entre duas geometrias em unidades projetadas;
  - *Intersects(geometry,geometry)*: retorna 1 (verdadeiro) se a primeira geometria informada cruza a segunda;
  - *Touches(geometry,geometry)*: retorna 1 (verdadeiro) se a primeira geometria toca na segunda;

- *Within(geometry,geometry)*: retorna 1 (verdadeiro) se a primeira geometria está dentro do espaço da segunda;
- *Overlaps(geometry,geometry)*: retorna 1 (verdadeiro) se a primeira geometria sobrepõe a segunda.

- **Funções de Processamento da Geometria:**
  - *Centroid(geometry)*: retorna o identificador de centro da geometria como um ponto;
  - *Area(geometry)*: retorna a área da geometria, em caso de polígono ou multipolígono;
  - *Length(geometry)*: retorna o comprimento da geometria;
  - *Intersection(geometry,geometry)*: retorna uma geometria que representa a interseção do ponto fixo da primeira geometria em relação a segunda.
- **Assessores Geométricos:**
  - *AsText(geometry)*: retorna a representação WKT da geometria;
  - *AsBinary(geometry)*: retorna a representação WKB da geometria;
  - *SRID(geometry)*: retorna o SRID da geometria;
  - *NumGeometries(geometry)*: caso a geometria seja do tipo *GEOMETRYCOLLECTION* ou *MULTIPOLYGON*, retorna o número de geometrias, caso contrário retorna nulo;
  - *NumPoints(geometry)*: retorna o número de pontos da geometria;
  - *GeometryType(geometry)*: retorna o tipo da geometria em formato de texto (*String*);
  - *X(geometry)*: retorna a coordenada de X do primeiro ponto da geometria;
  - *Y(geometry)*: retorna a coordenada de Y do primeiro ponto da geometria;
  - Construtores Geométricos:
  - *GeomFromText(text,SRID)*: retorna uma geometria convertendo do formato WKT para o SRID informado;
  - *GeomFromWKB(bytea,SRID)*: retorna uma geometria convertendo do formato WKB para o SRID informado;

3.1.2 Oracle e Oracle Spatial

Atualmente na versão 11g, o Oracle é um SGBD Objeto-Relacional proprietário e considerado líder de mercado. Assim como o PostgreSQL com o PostGIS, o Oracle possui uma extensão para manipulação de dados espaciais chamada Oracle Spatial.

Sendo Objeto-Relacional significa que o Oracle possui a opção de armazenamento de objetos e nesse caso específico, objetos geográficos.

O Oracle Spatial possui as seguintes características (ORACLE SPATIAL, 2006):

- Modelos de dados (MDSYS) que definem tipos de dados espaciais;
- Mecanismo de indexação espacial;
- Um conjunto de operadores e funções para realizar consultas, junção espacial e outras operações de análise espacial;
- Topologia de modelo de dados para trabalhar com dados sobre nós, arestas e faces;
- GeoRaster, um recurso que permite armazenar, consultar, criar índices, analisar e distribuir dados do tipo *raster.*

A Figura 12 mostra os principais tipos de geometrias suportados pelo Spatial.

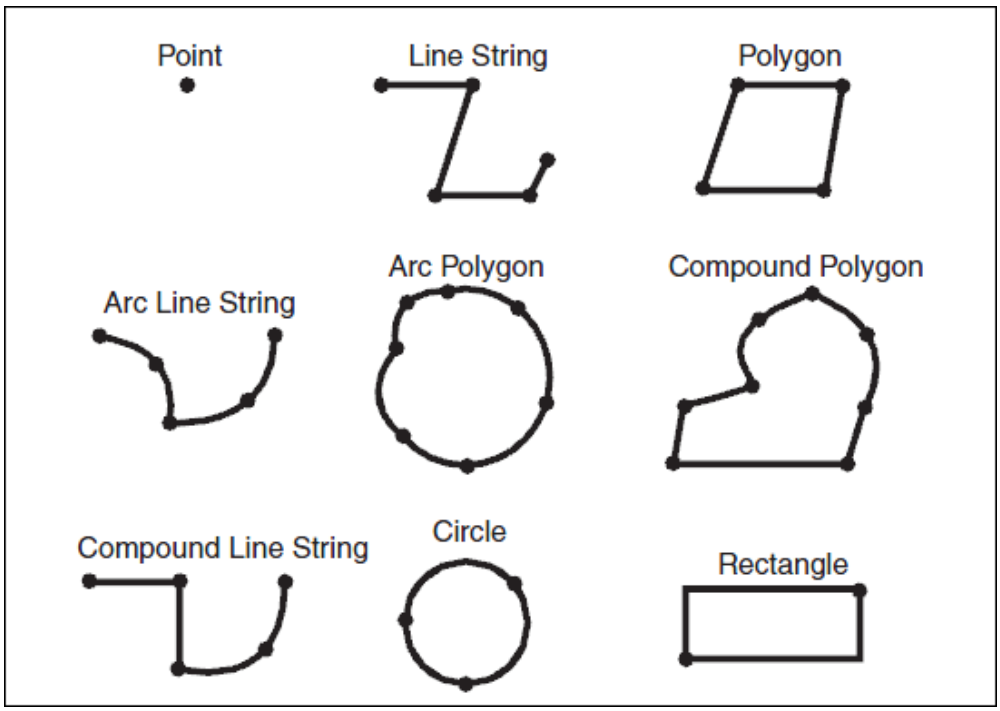

Figura 11 - Tipos geométricos suportados pelo Oracle Spatial
Fonte: Oracle Spatial, 2006.

O tipo *SDO_GEOMETRY* corresponde ao tipo padrão dos objetos correspondentes as geometrias utilizadas no banco de dados, como visto anteriormente o tipo *geometry* do PostGIS.

O Quadro 4 demonstra a criação de uma tabela com uma coluna correspondente a um objeto geográfico. Uma tabela pode ter uma ou mais colunas geográficas.

```
CREATE TABLE geometrias(
        id NUMBER PRIMARY KEY,
        nome VARCHAR2(60),
        shape SDO_GEOMETRY);
```

Quadro 4 - Criação de tabela com objeto geográfico no Oracle Spatial

O Quadro 5 apresenta um exemplo de inserção na tabela criada seguido da descrição dos parâmetros (BAPTISTA, 2006).

```
INSERT INTO geometrias VALUES(
        1, 'Geometria 1',
        SDO_GEOMETRY(
                2003, -- SDO_GTYPE
                NULL, -- SDO_SRID
                NULL, -- SDO_POINT
                SDO_ELEM_INFO_ARRAY(1,1003,1), -- SDO_ELEM_INFO
                SDO_ORDINATE_ARRAY(5,1, 8,1, 8,6, 5,7, 5,1) -- SDO_ORDINATES
        )
);
```

Quadro 5 - Exemplo de inserção com Oracle Spatial

- *SDO_GTYPE* – indica o tipo da geometria possuindo quatro dígitos no formato *dltt*, onde:
  - *d*: indica o número de dimensões (2, 3 ou 4);
  - *l*: usado em geometrias de 3 dimensões (padrão = 0);
  - *tt*: indica o tipo de geometria: *00 – Unknown, 01 – Point, 02 – Line* ou *Curve, 03 – Polygon, 04 – Collection, 05 – MultiPoint, 06 – Multiline, 07 – MultiPolygon.*
- *SDO_SRID* – identifica um sistema de coordenadas;
- *SDO_POINT* – usado para armazenar pontos;
- *SDO_ELEM_INFO* – vetor que armazena as características dos elementos que compõem a geometria;
- *SDO_ORDINATES* – vetor com as coordenadas da geometria.

O Oracle Spatial utiliza um modelo de consulta baseado em duas etapas chamadas primeiro e segundo filtro. O primeiro filtro (*SDO_FILTER*) considera as

aproximações das geometrias, seguindo o critério de mínimo retângulo envolvente (MBR), reduzindo assim a complexidade computacional.

O segundo filtro trabalha com funções exatas, aplicando as consultas em cima do resultado do primeiro filtro (BAPTISTA, 2006). A função mais utilizada nesse caso é *SDO_RELATE* que recebe duas geometrias e um terceiro atributo referente à sub-consulta que será realizada, podendo ser: *EQUAL*, *DISJOINT*, *TOUCH*, *INSIDE*, *COVERS*, *COVEREDBY*, *OVERLAPDYINTERSECT*, *ON*, *CONTAINS*, *OVERLAPBDYDISJOINT* e *ANYINTERACT* (ORACLE SPATIAL, 2006).

3.2 SERVIDORES DE MAPAS

Os servidores de mapas possuem como objetivo prover um ambiente de desenvolvimento para construção de aplicativos espaciais na internet (RECKZIEGEL, 2008). Eles tem a capacidade de integrar diversos repositórios de dados geográficos, abstraindo recursos oferecendo simplicidade na utilização em paralelo com uma alta performance e interoperabilidade dos dados.

Esses contêineres ainda tem a característica de oferecer interfaces amigáveis aos usuários para manipulação direta.

Um dos conceitos chaves desses servidores e de praticamente todos os aplicativos utilizados para visualização de dados geográficos é a apresentação em camadas (*layers*). Os dados são enfileirados e sobrepostos, permitindo aos usuários a manipulação e combinação dos dados através, por exemplo, das funções espaciais citadas anteriormente.

Os dois principais servidores de mapas conhecidos atualmente são o *Geoserver* e o *Mapserver* não só pela sua utilização e aprovação de uma grande comunidade de usuários, mas também por implementar em quase sua totalidade os padrões definidos pela OpenGIS (OLIVEIRA, 2009).

A aplicação *web* usada para teste no estudo de caso desse trabalho utiliza o servidor *Geoserver.*

3.2.1 Geoserver

Geoserver é um servidor de código-fonte aberto escrito em Java que possibilita aos usuários compartilhar e editar dados geográficos (GEOSERVER, 2011). É uma implementação de referência dos padrões OGC, *Web Feature Service* (WFS) e *Web Coverage Service* (WCS), sendo compatível com os certificados *Web Map Service* (WMS).

Dentre as principais características do Geoserver, destacam-se (RECKZIEGEL, 2008):

- Fácil utilização através da ferramenta de administração via web (Figura 13);
- Suporte maduro a PostGIS, Shapefile, ArcSDE e Oracle;
- Saída do WMS como JPEG, GIF, PNG, SVG e GML;
- Imagens com *anti-aliasing*;
- Suporte nativo a estilos especificados pela OGC, chamados SLD (*Styled Layer Descriptor*)
- Suporte completo a filtros em todos os formatos de dados no WFS;
- Suporte a transações de bancos de dados atômicas através do protocolo WFS-T (*Web Feature Service Transaction*), disponível para todos os formatos de dados;
- Baseado em *servlets Java Enterprise Edition* (JEE);
- Projetado para extensões.

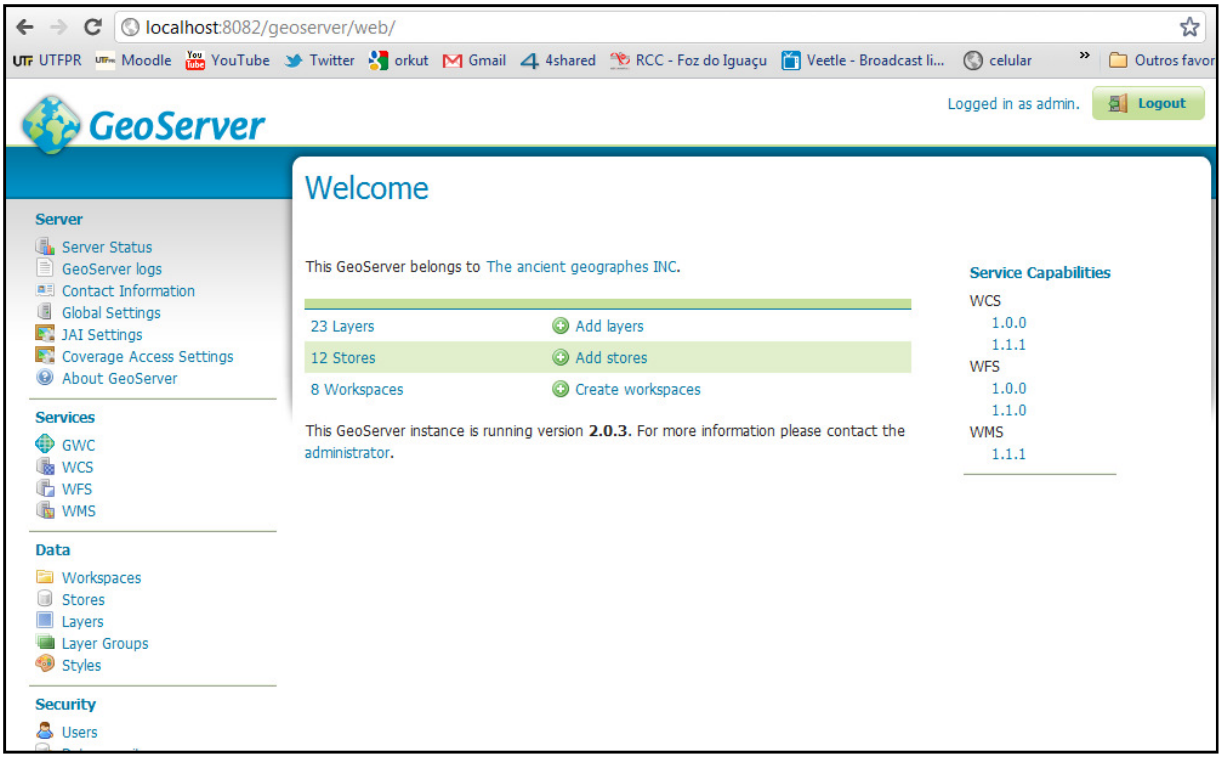

Figura 12 - Interface de administração para usuários do Geoserver

Através do ambiente gráfico é possível interagir com representações vetoriais obtidas de bancos de dados geográficos, arquivos espaciais, *shapefiles* e WFS. É possível também manipular representações matriciais (*raster*) (Figura 14).

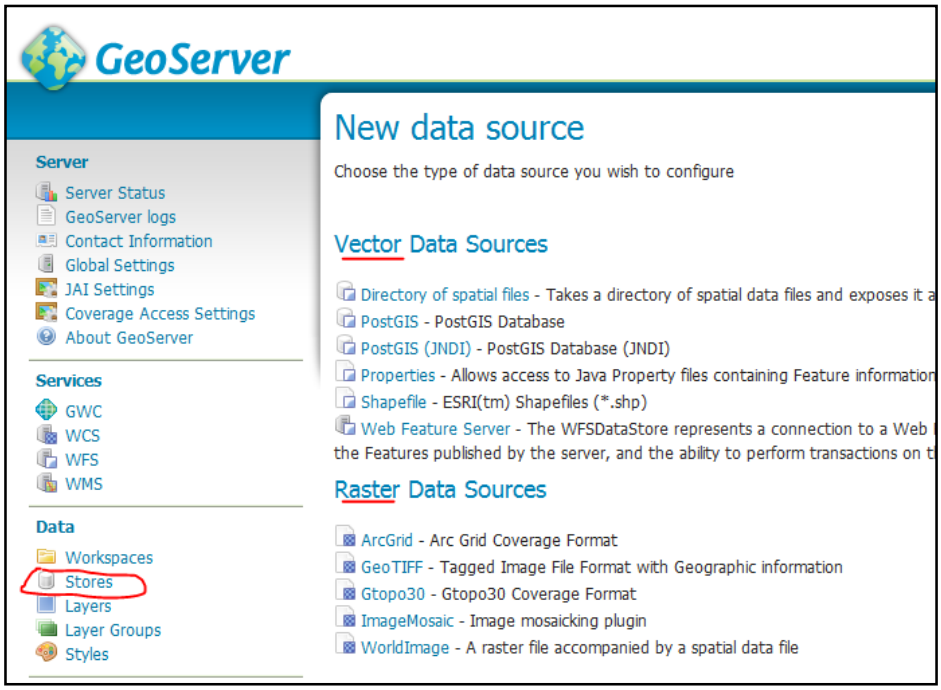

Figura 13 - Opções de interação com dados geográficos (Stores)

Após a criação das interações, é possível criar camadas (*layers*) para "enxergar" esses dados (Figuras 15 e 16). As *layers* criadas ainda podem ser agrupadas (Layer Groups) (Figura 17), estilizadas (*Styles*) e pré-visualizadas.

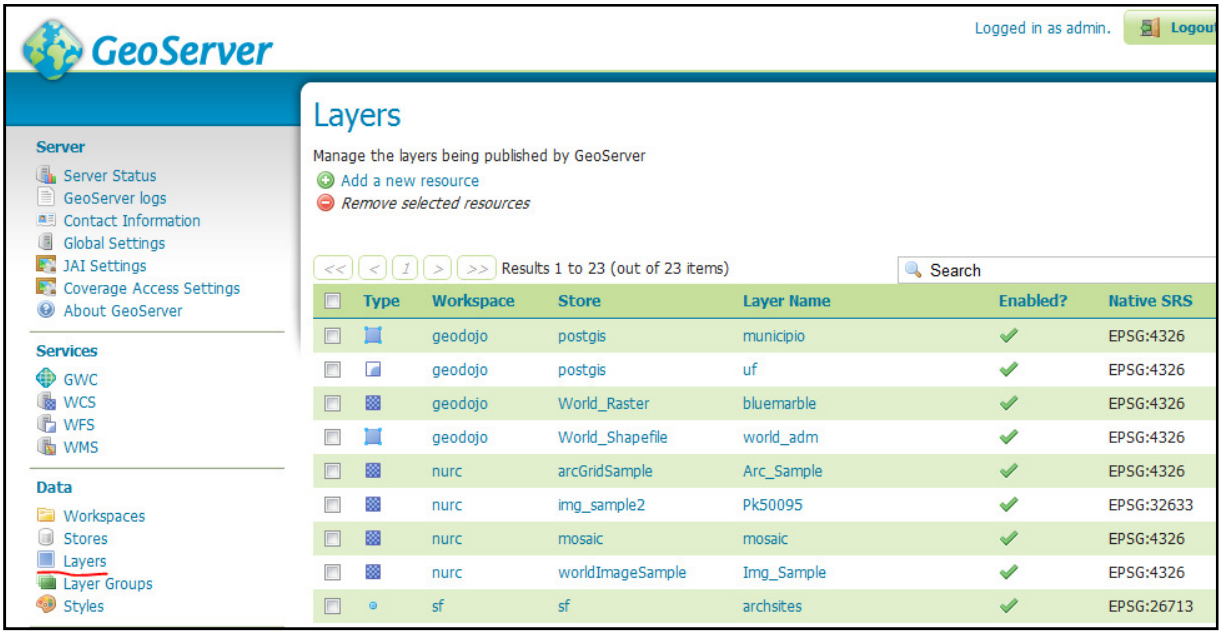

Figura 14 - Lista de camadas criadas

Figura 15 - Criação de uma nova camada a partir das interações feitas

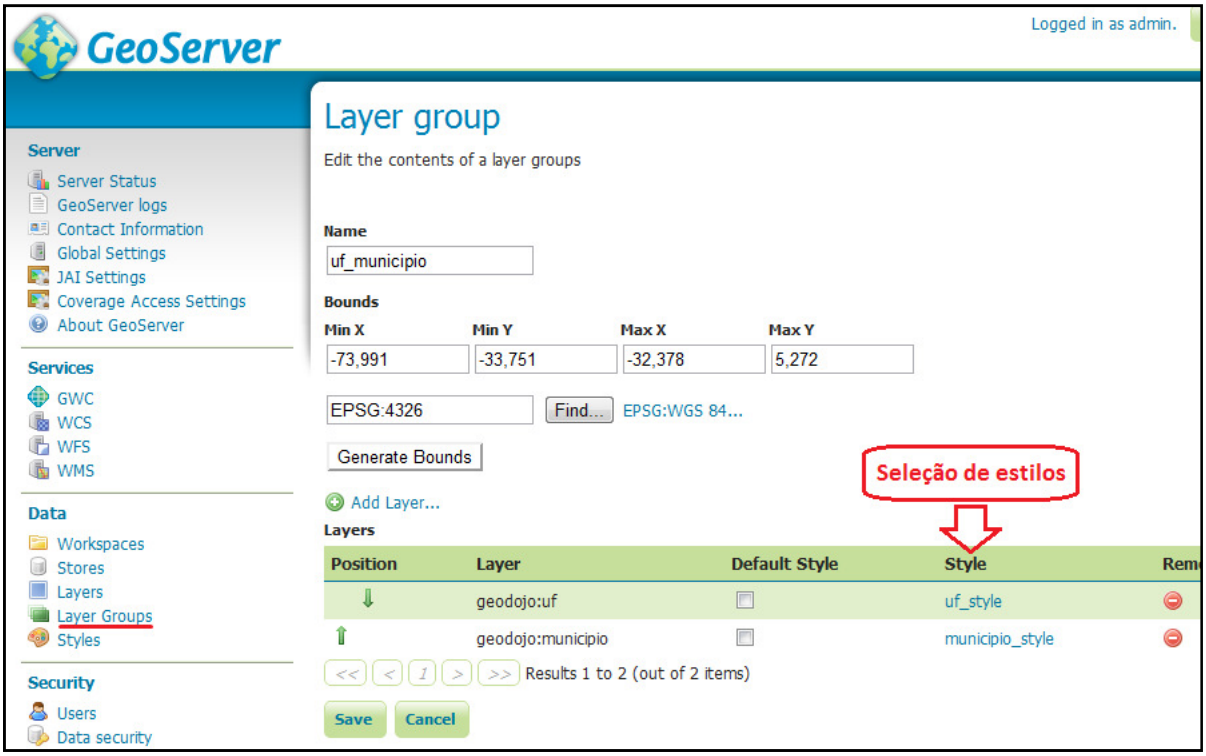

Figura 16 - Agrupamento de camadas e escolha de estilos

Uma das características apontadas anteriormente descrevia a capacidade da criação de estilos chamados de SLD. Esses estilos estão ligados à forma como as geometrias serão apresentadas, contendo atributos referentes a cores, formas, tamanho, dentre outros atributos, descritos num arquivo XML.

O Geoserver possui uma gama de estilos pré-definidos e oferece a opção da importação de estilos criados bem como a modificação dos existentes (Figura 18).

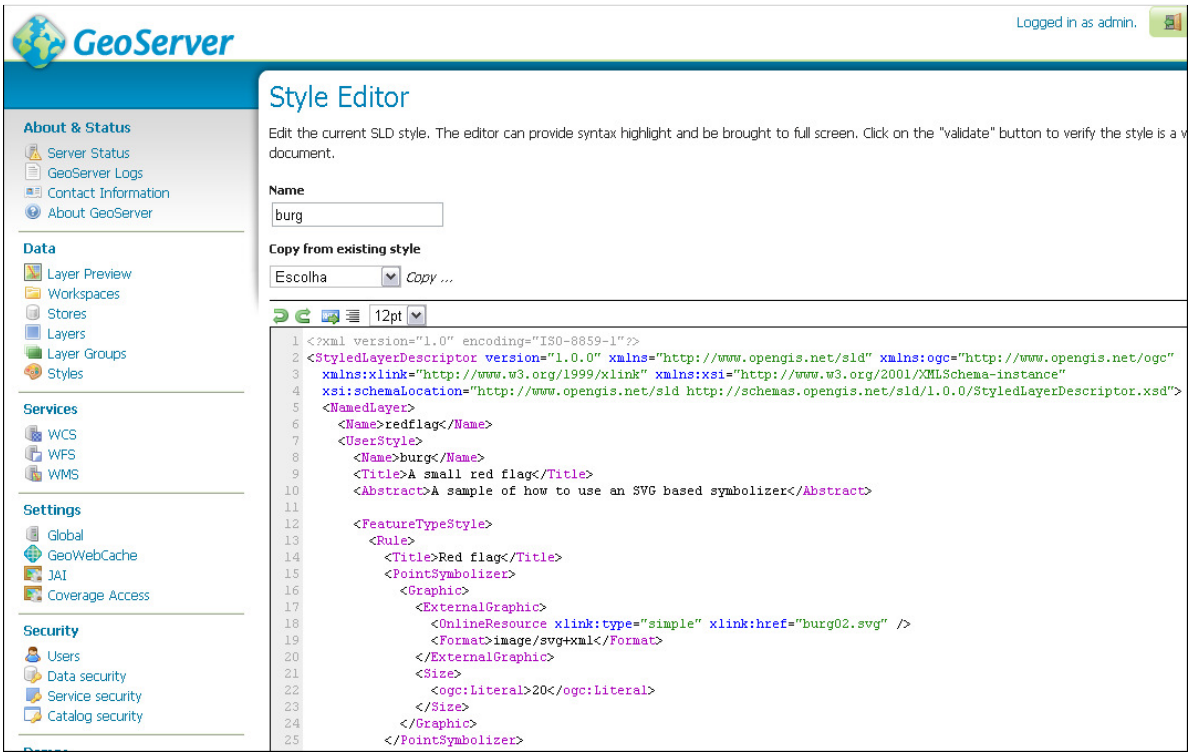

Figura 17 - Criação e edição de arquivos SLD

As camadas criadas podem ser visualizadas através do ambiente do Geoserver (Figura 19). Além da visualização utilizando OpenLayers (Figura 20), o Geoserver oferece opções para download de arquivos do tipo *Keyhole Markup Language* (KML) utilizado para visualização no Google Earth, Google Maps e outros aplicativos bem como a visualização de arquivos do tipo *Geography Markup Language* (GML) que são descritores XML padronizados pela OGC.

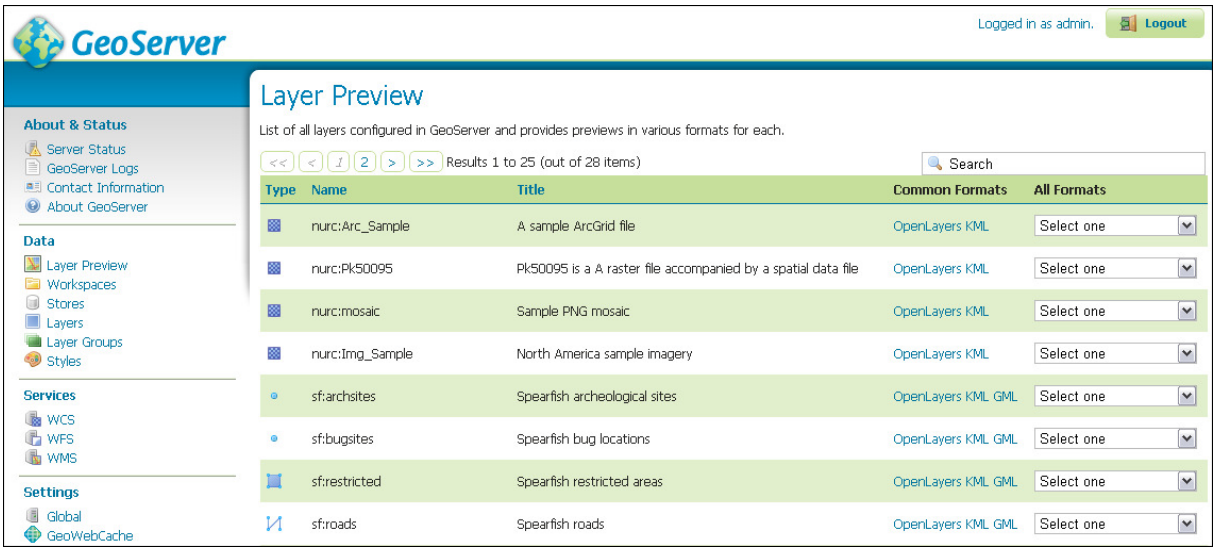

Figura 18 – Lista de camadas criadas com opção de pré-visualização

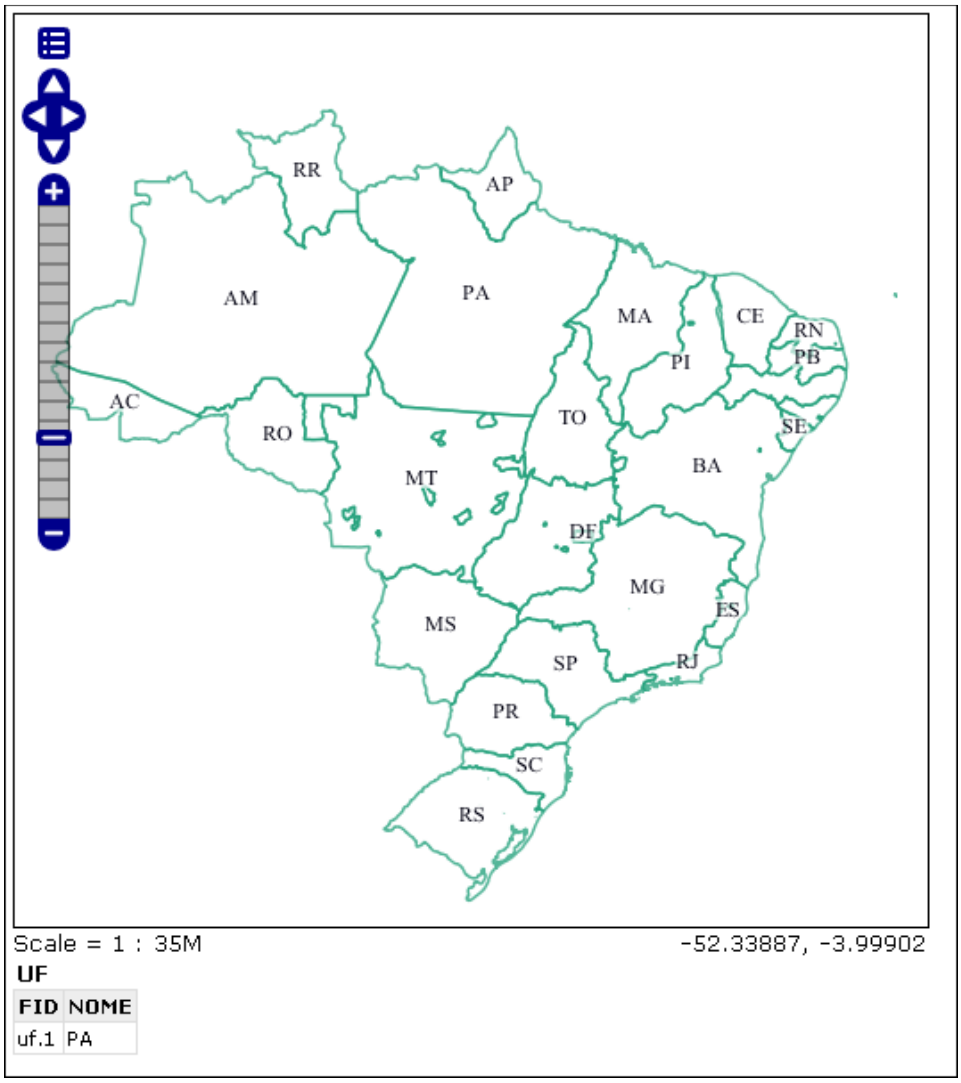

Figura 19 - Visualização da camada uf_municipios com OpenLayers

3.2.2 Mapserver

O Mapserver, assim como o Geoserver, é um servidor de código aberto para manipulação e visualização de dados geográficos via web, sendo um dos primeiros a surgir e um dos mais utilizados atualmente. Como principal característica destaca-se a flexibilidade no desenvolvimento de soluções web, tornando-o melhor que muitos servidores.

Sua principal funcionalidade é coordenar a leitura e manipulação dos dados espaciais e formatá-los em resultados processados, entregando-os as ferramentas de produção de mapas digitais (RECKZIEGEL, 2008) (Figura 21).

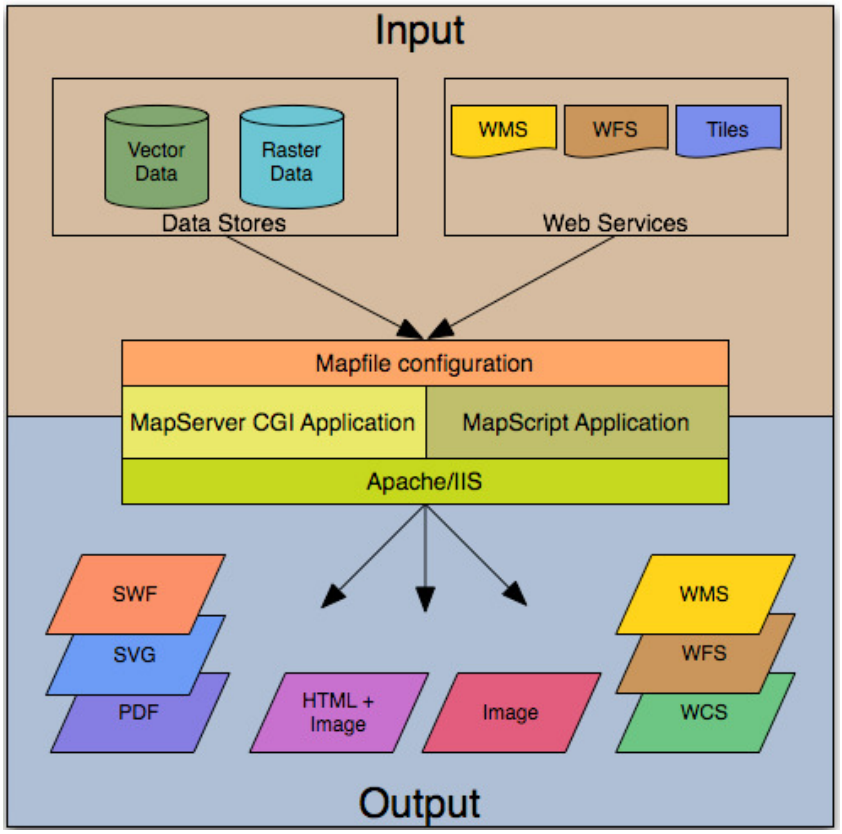

Figura 20 - Arquitetura do Mapserver
Fonte: Reckziegel, 2008.

Fornece uma interface CGI (*Common Gateway Interface*) possuindo diversas funcionalidades para o desenvolvimento de aplicações web. Sua API pode ser acessada através de linguagens como Perl, Python, Java, C (linguagem nativa) e PHP.

Alguma das principais características do Mapserver:

- Suporte a dados vetoriais e matriciais;
- Suporte à fonte *TrueType*;
- Customizável através de *templates*;
- Projeção dinâmica;
- Geração automática de legenda e barra de escala;
- Geração de mapas temáticos usando expressões lógicas ou regulares baseadas em classes;
- Configuração dinâmica através de URLs.

Um dos fatores mais importantes para o sucesso do Mapserver é a quantidade de colaboradores que ele possui, dentre comunidades e grupos de estudo, que contribuem com o crescimento da ferramenta.

3.3 OPENLAYERS

"OpenLayers é uma biblioteca JavaScript de código aberto, do lado cliente, de criação de mapas interativos para a web" (HAZZARD, 2011). Por ser do lado cliente, não tem dependência nenhum com o lado servidor. "Possui uma vasta API que possibilita a construção de sofisticadas aplicações geográficas web, além de implementar os protocolos OpenGIS WMS e WFS" (OLIVEIRA, 2009).

O OpenLayers permite a customização de todos os aspectos do mapa, camadas, controles, eventos dentre outros.

O fato de ser uma biblioteca de lado cliente, não exclui a alternativa que a biblioteca oferece de interação e atuação junto a um servidor de mapas. Isso significa que uma visualização de uma ou mais camadas pode partir de recursos previamente criados em um servidor de mapas

A biblioteca também interage com outras bibliotecas de mapas como o Google Maps API, Yahoo!, Bing e ESRI Mappings API.

A Figura 22 mostra um exemplo de visualização de um mapa WMS utilizando OpenLayers.

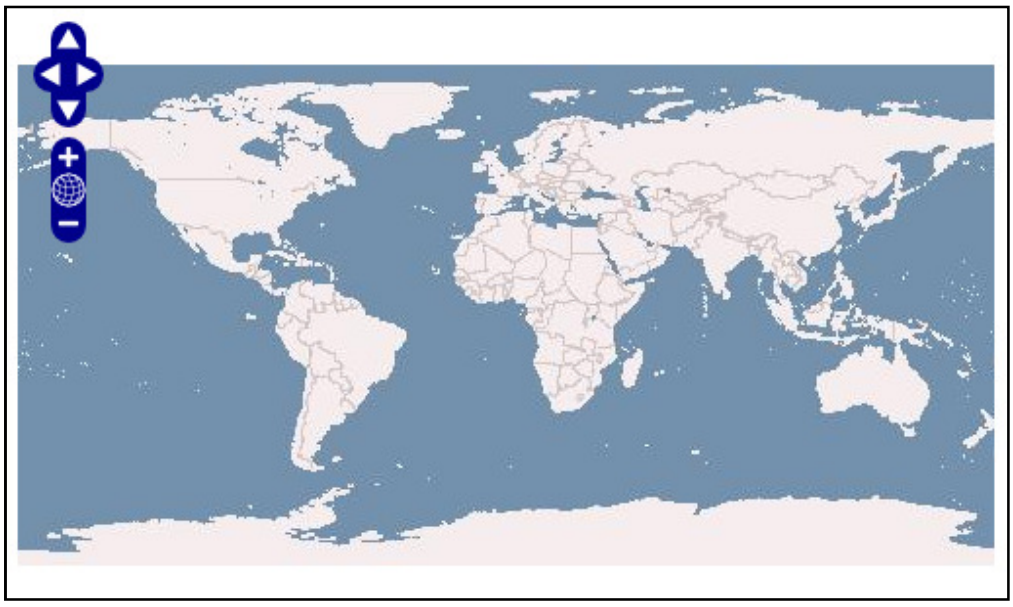

Figura 21 - Visualização WMS com OpenLayers

O Quadro 6 apresenta a implementação, considerada simples e de fácil entendimento.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  <meta charset="utf-8"/>
5  <title>OpenLayers - Primeiro Mapa</title>
6  <script type="text/javascript" src="OpenLayers.js"></script>
7      <script type="text/javascript">
8          var map;
9
10         function init(){
11             map = new OpenLayers.Map('map_element');
12             var wms = new OpenLayers.Layer.WMS(
13                 'OpenLayers WMS',
14                 'http://vmap0.tiles.osgeo.org/wms/vmap0',
15                 {layers: 'basic'},
16                 {}
17             );
18
19             map.addLayer(wms);
20             if(!map.getCenter()){
21                 map.zoomToMaxExtent();
22             }
23         }
24
25     </script>
26 </head>
27 <body onload="init();">
28     <div id="map_element" style="width: 500px; height: 500px;"></div>
29 </body>
30 </html>
```

Quadro 6 - Exemplo simples do uso de OpenLayers

O evento *onload* da marcação *body* chama a função *init()* inicializa uma variável chamada *map* (linha 8) com o construtor OpenLayers.Map que recebe o *id* de visualização do mapa (linha 11). A variável *wms* é inicializada com o construtor de camadas *OpenLayer.Layer.WMS*, que cria uma camada de visualização do tipo WMS (linha 12) e recebe o nome de exibição do mapa (linha 13), a URL do serviço ou servidor que fornece o mapa (linha 14) e as camadas que serão exibidas (linha 15). O quarto parâmetro se refere a algumas opções de visualização que podem ser fornecidas já no construtor.

A camada é adicionada no mapa (linha 19) e é definido o espaço que será ocupado pela mesma (linha 21).

Na linha 6, a biblioteca OpenLayers é acessada localmente, mas há a possibilidade da utilização da biblioteca sem a necessidade do download da mesma, através da URL: http://openlayers.org/api/OpenLayers.js.

3.3.1 Camadas

O conceito de camadas diz respeito à representação de vários níveis de informação geográfica independentes uns dos outros (HAZZARD, 2011).

A Figura 23 exemplifica diversas formas de se representar o mundo real através de camadas sobrepostas, correspondentes a aspectos físicos e sociais.

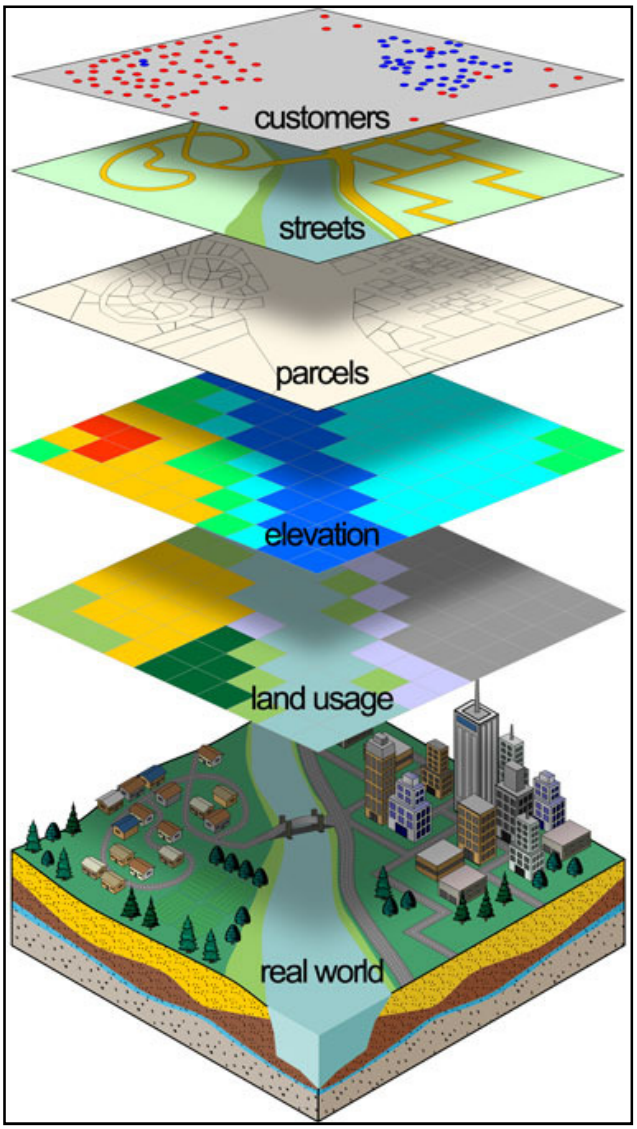

Figura 22 - Exemplo de representação usando camadas
Fonte: Gem Mapping and Design, 2009.

Cada mapa no OpenLayers pode ter uma ou mais camadas conforme a necessidade de representação e podem ser adicionadas em forma de vetor (Ex.: *map.addLayers([layer_1, layer_2])*) ou uma a uma.

O Quadro 7 apresenta uma implementação do uso de duas camadas, tendo uma camada como base.

```
var wms_base = new OpenLayers.Layer.WMS(
        'Base Layer',
        'http://vmap0.tiles.osgeo.org/wms/vmap0',
        {layers: 'basic'},
        {isBaseLayer: true}
    );

var wms_layerLabels = new OpenLayers.Layer.WMS(
            'Location Labels',
            'http://vmap0.tiles.osgeo.org/wms/vmap0',
            {layers: 'clabel,ctylabel,statelabel',
            transparent:true},
            {opacity: .5}
    );

map.addLayers([wms_base, wms_layerLabels]);
```

Quadro 7 - Exemplo do uso de mais de uma camadas

A segunda camada sobrepõe a primeira exibindo os nomes dos países, cidades e países (Figura 24).

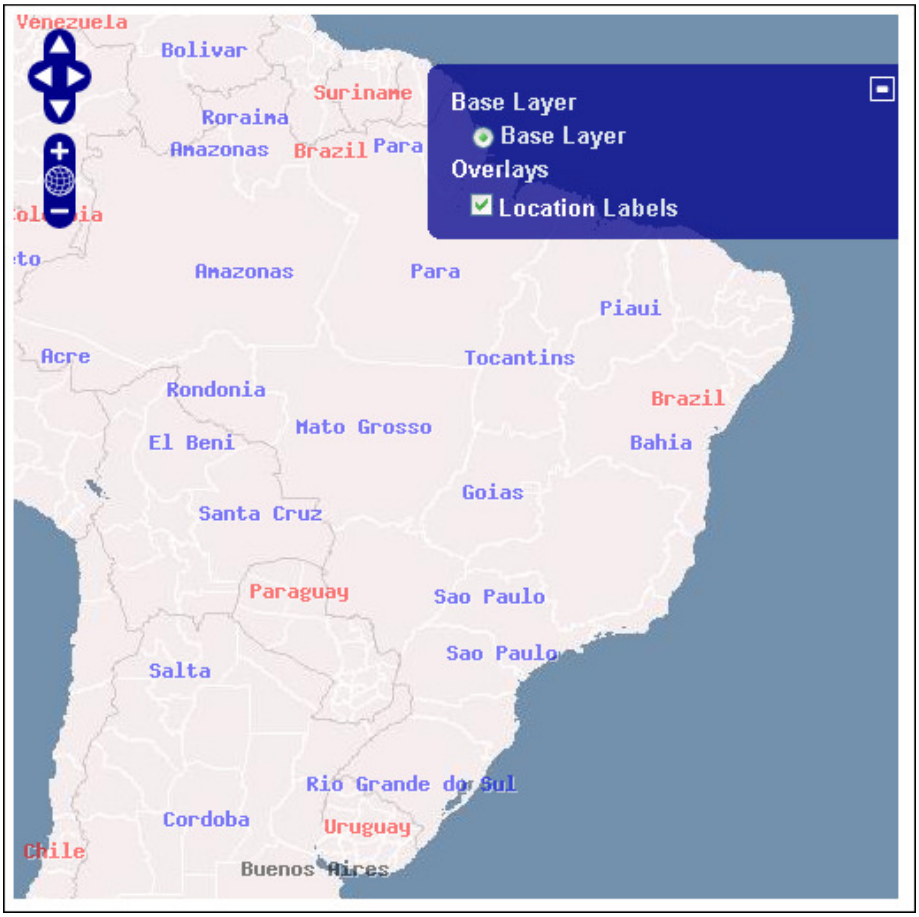

Figura 23 - Visualização do exemplo com duas camadas

As camadas podem ser apresentadas em diversos tipos de representação, que são descritos a seguir.

- **Camadas WMS**

O primeiro exemplo mostrado na Figura 22 utiliza uma camada WMS, que corresponde a um protocolo definido pela OGC para comunicação de dados espaciais renderizados como imagem. A construção de uma camada WMS é feita através de quatro parâmetros, sendo que só o último é opcional (QUADROS, 2009):

- *name {String}*: Nome da camada;
- *url {String}*: URL do serviço WMS (ex.: Quadro 6, linha 14);
- *params {Object}*: Um objeto com informações para o método *GetMap*;
- *options {Object}*: Opções extras para configuração.

O Quadro 8 apresenta um exemplo com praticamente todas as propriedades setadas:

```
<script type="text/javascript" src="OpenLayers.js"></script>
<script type="text/javascript">
    var map;
    function init(){
        map = new OpenLayers.Map('map');
        var wms = new OpenLayers.Layer.WMS(
            "Propriedades da Camada",
            "http://vmap0.tiles.osgeo.org/wms/vmap0",
            {layers: 'basic',
             format: 'image/PNG',
             srs: 'EPSG:4326',
             style: 'mundo'
            },
            {isBaseLayer: 'true',
             numZoomLevels: 5,
             displayInLayerSwitcher: 'false',
             visibility: 'true',
             transistionEffect: 'resize',
             minResolution: 0.04394,
             maxResolution: 0.1757,
             maxExtent: new OpenLayers.Bounds(-51,17.34,8),
             minExtent: 'auto';
             minScale: 50000000,
             maxScale: 1000000000,
             units: 'degree'
            }
        );
        map.addLayer(wms);
        map.zoomToMaxExtent();
    }
</script>
```

Quadro 8 - Propriedades de um construtor de camadas

- *layers* – define a(s) camada(s) que serão utilizadas;
- *format* – formato na qual será gerada a imagem;

- *srs* – define a projeção específica da camada;
- *style* – estilo que será utilizado na camada;
- *isBaseLayer* – define se a camada será a camada base;
- *numZoomLevels* – define a quantidade de zooms permitida pela camada;
- *displayInLayerSwitcher* – define se a camada será visível na lista de camadas;
- *visibilty* – define se a camada estará visível no mapa;
- *transactionEffect* – define o efeito utilizado nas operações de *pan* e *zoom*;
- *minResolution* – valor mínimo de resolução;
- *maxResolution* – valor máximo de resolução;
- *maxExtent* – valor máximo de extensão;
- *minExtent* – valor mínimo de extensão;
- *minScale* – valor mínimo de escala;
- *maxScale* – valor máximo de escala;
- *units* – unidade de medida.

- **Camadas Comerciais**

Como dito anteriormente, o OpenLayers interage com outras bibliotecas de mapas. Algumas das camadas suportadas são: Google Maps API, Yahoo Maps, Bing, ESRI Mappings API, Virtual Earth e Multimap.

O Quadro 9 exemplifica a interação com a biblioteca Google Maps API.

```
2  <html>
3  <head><meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
4  <title>Interação OpenLayers com Google Maps API</title>
5  <script src="http://maps.google.com/maps?file=api"></script>
6  <script type="text/javascript" src="OpenLayers.js"></script>
7      <script type="text/javascript">
8          var map;
9          function init(){
10             map = new OpenLayers.Map('map');
11             map.addControl(new OpenLayers.Control.LayerSwitcher());
12
13             var gphy = new OpenLayers.Layer.Google(
14                 "Google Physical",
15                 {type: G_PHYSICAL_MAP}
16             );
17             var gmap = new OpenLayers.Layer.Google(
18                 "Google Streets", // the default
19                 {numZoomLevels: 20}
20             );
21             var ghyb = new OpenLayers.Layer.Google(
22                 "Google Hybrid",
23                 {type: G_HYBRID_MAP, numZoomLevels: 20}
24             );
25             var gsat = new OpenLayers.Layer.Google(
26                 "Google Satellite",
27                 {type: G_SATELLITE_MAP, numZoomLevels: 22}
28             );
29             map.addLayers([gphy, gmap, ghyb, gsat]);
30             map.setCenter(new OpenLayers.LonLat(10.2, 48.9), 5);
31         }
32     </script>
33 </head>
34 <body onload="init();">
35     <div id="map" style="width: 500px; height: 500px;"></div>
36 </body>
37 </html>
```

Quadro 9 - Exemplo de interação OpenLayers com Google Maps API

No exemplo, são adicionadas quatro camadas referentes aos quatro tipos de visualização do Google Maps. A interação é feita através do acesso a API do Google Maps (linha 5) e através dos construtores (*OpenLayers.Layer.Google*).

A linha 11 adiciona um controle para troca de camada (*LayerSwitcher*). O resultado pode ser visualizado na Figura 25.

Figura 24 - Exemplo de interação OpenLayers com Google Maps API

- **Camadas Vetoriais**

A opção de uso de camadas vetoriais (*OpenLayer.Layer.Vector*) é uma das mais poderosas características do OpenLayers. Ela permite a criação de pontos, linhas e polígonos no mapa bem como a personalização dos estilos. Com essa opção é possível ainda ler e escrever em diversos formatos de dados vetoriais serializáveis como o GeoJSON, KML, GeoRSS, GML e WKT (QUADROS, 2009).

3.3.2 Controles

O OpenLayers disponibiliza os controles como forma de interação com o mapa. Alguns controles tem representação visual e outros não (QUADROS, 2009). Os controles podem ser adicionados ao mapa de duas formas (HAZZARD, 2011):

- Durante a construção do mapa, passando um *array* de objetos do tipo *OpenLayers.Control*;
- Através das funções *addControl()* e *addControls()* que recebem um *array* de objetos de controle e são chamadas após a criação do objeto mapa.

Quatro controles já vêm adicionados por padrão quando da criação do mapa:

- *OpenLayers.Control.Navigation*: responsável pelo tratamento das interações de eventos do mouse como arrastar e zoom através do clique duplo e rolagem;
- *OpenLayers.Control.PanZoom*: adiciona um controle de pan e zoom no canto superior esquerdo do mapa.
- *OpenLayers.Control.ArgParser*: controla e atribui as variáveis passadas via URL;
- *OpenLayers.Control.Attribution*: mostra uma descrição referente a cada camada separadamente, um texto exibido no mapa descrevendo as camadas. Apesar de vir adicionado por padrão só será exibido alguma descrição se setada na construção da camada.

A Figura 26 mostra o uso dos principais controles OpenLayers seguidos da descrição de cada controle.

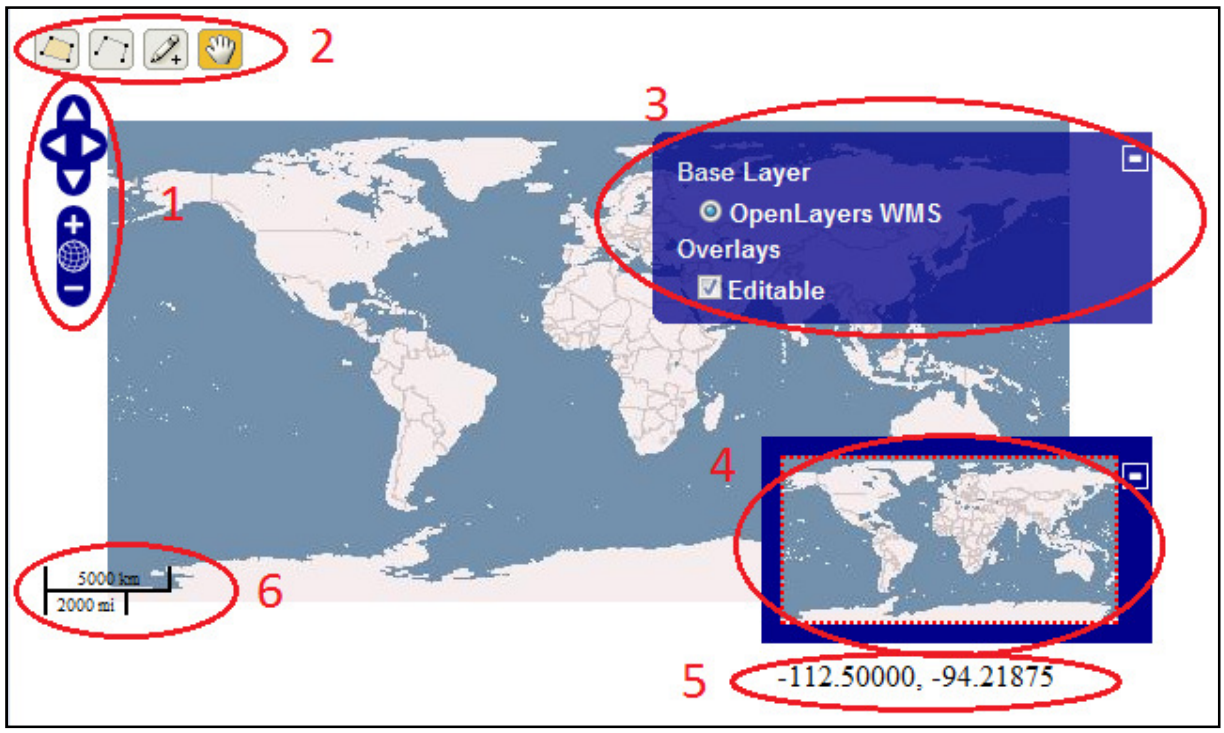

Figura 25 - Utilizando controles OpenLayers

1. Controles padrão (*Pan e Zoom*);
2. Ferramentas de Edição (*OpenLayers.Control.EditingToolbar*): utilizadas quando há a necessidade de editar ou criar novas geometrias diretamente no mapa;
3. Lista de Camadas (*OpenLayers.Control.LayerSwitcher*): mostra as camadas disponíveis, a camada base e oferece opções para ativar ou desativar uma camada;
4. Referência (*OpenLayers.Control.Overview*);
5. Coordenadas (*OpenLayers.Control.MousePosition*): indica o par de coordenadas conforme o posicionamento do mouse;
6. Escala (*OpenLayers.Control.ScaleLine*): insere a escala no canto inferior esquerdo do mapa.

O Quadro 10 apresenta a implementação do exemplo anterior.

```
map = new OpenLayers.Map('map_element');
var wms = new OpenLayers.Layer.WMS(
    'OpenLayers WMS',
    'http://vmap0.tiles.osgeo.org/wms/vmap0',
    {layers: 'basic'},
    {isBaseLayer: true}
);

var vlayer = new OpenLayers.Layer.Vector("Editable");
map.addLayer(vlayer);
map.addLayer(wms);


map.addControl(new OpenLayers.Control.ScaleLine());
map.addControl(new OpenLayers.Control.OverviewMap());
map.addControl(new OpenLayers.Control.LayerSwitcher());
var container = document.getElementById("panel");
map.addControl(new OpenLayers.Control.EditingToolbar(
                    vlayer, {div: container}));
map.addControl(new OpenLayers.Control.MousePosition(
                    {div: cordenadas}));
if(!map.getCenter()){
    map.zoomToMaxExtent();
}
```

Quadro 10 - Trecho de implementação de alguns controles OpenLayers

3.4 OPENLAYERS FOR JSF

O OpenLayers, conforme apresentado anteriormente, é uma biblioteca desenvolvida em JavaScript utilizada para criação e manipulação de mapas que são visíveis nos principais navegadores. No entanto, o desenvolvimento com essa biblioteca resulta numa grande quantidade de código, sem contar a carência de recursos para o trabalho com JavaScript nos ambientes de desenvolvimento (IDEs) (OLIVEIRA, 2009).

Neste cenário, surge o OL4JSF (*OpenLayers for JSF*). Um framework brasileiro que possui como característica principal o encapsulamento dos recursos do OpenLayers para uma plataforma JavaServer Faces, obtendo o máximo de vantagens das duas tecnologias.

"JavaServer Faces é a tecnologia Java Enterprise Edition (JEE) responsável por fornecer um *framework* de componentes de interface com o usuário para construção de aplicações Web" (SUN MICROSYSTEMS, 2008).

As características principais do JSF são:

- Arquitetura MVC (Model-View-Controller) que permite a separação entre a camada de apresentação e a lógica;
- Componentes pré-fabricados e extensíveis;
- Tratamento de eventos;
- Validação e conversão dos dados;
- Internacionalização;
- Controle de navegação.

O OL4JSF, portanto, agrega a opção de renderização de mapas do OpenLayers aos recursos de alta produtividade do JSF. Praticamente todas as funções do OpenLayers são transformadas em recursos JSF. A Tabela 2 mostra todos os componentes implementados no OL4JSF com seus respectivos equivalentes no OpenLayers.

Tabela 2 - Relação de componentes OL4JSF com seus equivalentes OpenLayers

OL4JSF	OpenLayers	Função
<m:arcGIS93RestLayer/>	OpenLayers.Layer.ArcGIS93Rest	Camada ESRI ArcGIS Server 9.3
<m:attributionControl/>	OpenLayers.Control.Attribution	Adiciona atribuição das camadas do mapa
<m:control/>	OpenLayers.Control	Afeta a aparência ou comportamento do mapa
<m:drawFeatureControl/>	OpenLayers.Control.DrawFeature	Controle para desenhar sobre a camada
<m:featureInfoPopup/>	É necessário se combinar vários componentes	Exibe informações sobre a camada em um popup
<m:gmlLayer/>	OpenLayers.Layer.GML	Camada GML
<m:geoRSSLayer/>	OpenLayers.Layer.GeoRSS	Camada GeoRSS
<m:getFeatureInfo/>	É necessário se combinar vários componentes	Obtém informações da camada
<m:googleLayer/>	OpenLayers.Layer.Google	Camada GoogleMaps
<m:keyboardDefaultsControl/>	OpenLayers.Control.KeyboardDefaults	Interage com o mapa pelo teclado
<m:layerSwitcherControl/>	OpenLayers.Control.LayerSwitcher	Controla a exibição das camadas do mapa
<m:map/>	OpenLayers.Map	Define o mapa propriamente dito
<m:mapServerLayer/>	OpenLayers.Layer.MapServer	Camada MapServer
<m:mousePositionControl/>	OpenLayers.Control.MousePosition	Mostra as coordenadas do ponteiro do mouse
<m:navigationControl/>	OpenLayers.Control.Navigation	Permite navegar pelo mapa
<m:overviewMapControl/>	OpenLayers.Control.OverviewMap	Miniatura do mapa
<m:panZoomBarControl/>	OpenLayers.Control.PanZoomBar	Controle de zoom
<m:panelControl/>	OpenLayers.Control.Panel	Container de componentes
<m:permalinkControl/>	OpenLayers.Control.Permalink	Link permanente
<m:registerEvent/>	Implementado através de função	Registra eventos
<m:scaleLineControl/>	OpenLayers.Control.ScaleLine	Exibe a escala do mapa
<m:script/>		Adição de código JavaScript
<m:vectorLayer/>	OpenLayers.Layer.Vector	Camada vetorial
<m:wfsLayer/>	OpenLayers.Layer.WFS	Camada WFS
<m:wmsLayer/>	OpenLayers.Layer.WMS	Camda WMS
<m:zoomBoxControl/>	OpenLayers.Layer.ZoomBox	Caixa de zoom
<m:zoomToMaxExtenControl/>	OpenLayers.Control.ZoomToMaxExtent	Maximiza o zoom do mapa

Fonte: Oliveira, 2009.

Para exemplificar o uso do framework de forma simples, o Quadro 11 apresenta a implementação mostrada no Quadro 10 utilizando o OL4JSF.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "h
3  <html xmlns="http://www.w3.org/1999/xhtml"
4        xmlns:h="http://java.sun.com/jsf/html"
5        xmlns:f="http://java.sun.com/jsf/core"
6        xmlns:ui="http://java.sun.com/jsf/facelets"
7        xmlns:m="http://www.ol4jsf.org">
8
9  <h:head>
10 </h:head>
11 <body>
12     <m:map width="512px" height="256px" options="{controls:[]}">
13         <m:wmsLayer name="OL4JSF WMS"
14                     url="http://labs.metacarta.com/wms/vmap0"
15                     params="{layers:'basic'}" />
16         <m:vectorLayer name="Editing" jsVariable="vlayer"/>
17         <m:navigationControl></m:navigationControl>
18         <m:panZoomBarControl />
19         <m:layerSwitcherControl />
20         <m:scaleLineControl />
21         <m:mousePositionControl />
22         <m:keyboardDefaultsControl />
23         <m:overviewMapControl/>
24         <m:editingToolbarControl layer="vlayer"/>
25     </m:map>
26
27 </body>
28 </html>
```

Quadro 11 - Código exemplo de uso do OL4JSF

Assim como o padrão de marcações do JSF, as marcações do OL4JSF nesse exemplo são acessadas pelo prefixo *m* que referencia o framework (linha 7). Logo na declaração do mapa já estão especificados a largura e altura, as opções e parâmetros iniciais. A Figura 27 mostra a visualização do exemplo.

Figura 26- Visualização do exemplo de uso do OL4JSF

O OL4JSF ainda prevê a adição de códigos JavaScript através da marcação *<m:script/>* caso haja necessidade ou particularidade não interpretada pelo framework.

Além de todos os componentes apresentados, o OL4JSF possui classes utilitárias que auxiliam no desenvolvimento de SIGs. Uma das classes mais utilizadas é a *org.ol4jsf.util.WKTFeaturesCollection*, que é responsável pela conversão de uma coleção de coordenadas para formato compatível com uma camada vetorial exibida no mapa (OLIVEIRA, 2009) .

O Quadro 12 exibe um exemplo de utilização dessa classe contendo uma lista de geometrias em formato WKT que são devolvidas à página para exibição.

```
@Model
public class VectorBean {

    private List<String> wkts;

    public VectorBean(){
        wkts = new ArrayList<String>();

        wkts.add("POLYGON((-74.864584 50.456448, -60.11511 33.77877, " +
                         "-81.5625 46.7578125, -74.864584 50.456448))");
        wkts.add("POLYGON((-121.654644 80.448477, -85.11635 23.686958, " +
                         "-74.5555 48.323254, -121.654644 80.448477))");
        wkts.add("LINESTRING(21.796875 50.2734375, 53.4375 56.6015625, " +
                         "51.328125 39.0234375, 75.234375 43.9453125, " +
                         "98.4375 42.5390625, 96.328125 27.7734375, " +
                         "106.875 34.8046875)");
        wkts.add("POINT(-39.25 -11.998988)");
    }

    public String getWkts() {
        WKTFeaturesCollection<String> features =
                                    new WKTFeaturesCollection<String>();
        features.addAllFeatures(wkts);

        return features.toMap();
    }
}
```

Quadro 12 - ManagedBean utilizando WKTFeaturesCollection

O Quadro 13 mostra a utilização de uma camada vetorial para exibição dos dados (Figura 28).

```
<m:map width="512px" height="256px">
      <m:vectorLayer name="Features"
                     value="#{vectorBean.wkts}"
                     options="{isBaseLayer: true}"/>
      <m:layerSwitcherControl />
</m:map>
```

Quadro 13 - Camada vetorial WKT

Figura 27 -Visualização de camada vetorial

4 ESTUDO DE CASO

A ferramenta escolhida para estudo de caso dos resultados obtidos consiste num editor de geometrias que resultarão em dados espaciais e descritivos relacionados, disposto em ambiente web. O editor permite a criação de geometrias tendo como base de edição o GoogleMaps.

Os dados espaciais criados podem servir de base para criação de mapas temáticos através do tratamento das informações e relação com funções.

As tecnologias utilizadas na ferramenta são:

- PostgreSQL/PostGIS: utilizado para o armazenamento dos dados gerados;
- Servidor de mapas Geoserver: utilizado na recuperação e estilização dos dados armazenados no banco;
- OpenLayers integrado com a GoogleMaps API;
- OL4JSF para utilizar as funcionalidades do OpenLayers (visualização e controles) e implementar os requisitos de lógica de negócio e aplicação e de interação com o modelo de dados;
- Google Geocoding API: para obter informações de localização;
- Primefaces: responsável pelos componentes AJAX de visualização e interação com o usuário.

A Figura 29 apresenta a ferramenta em sua forma inicial.

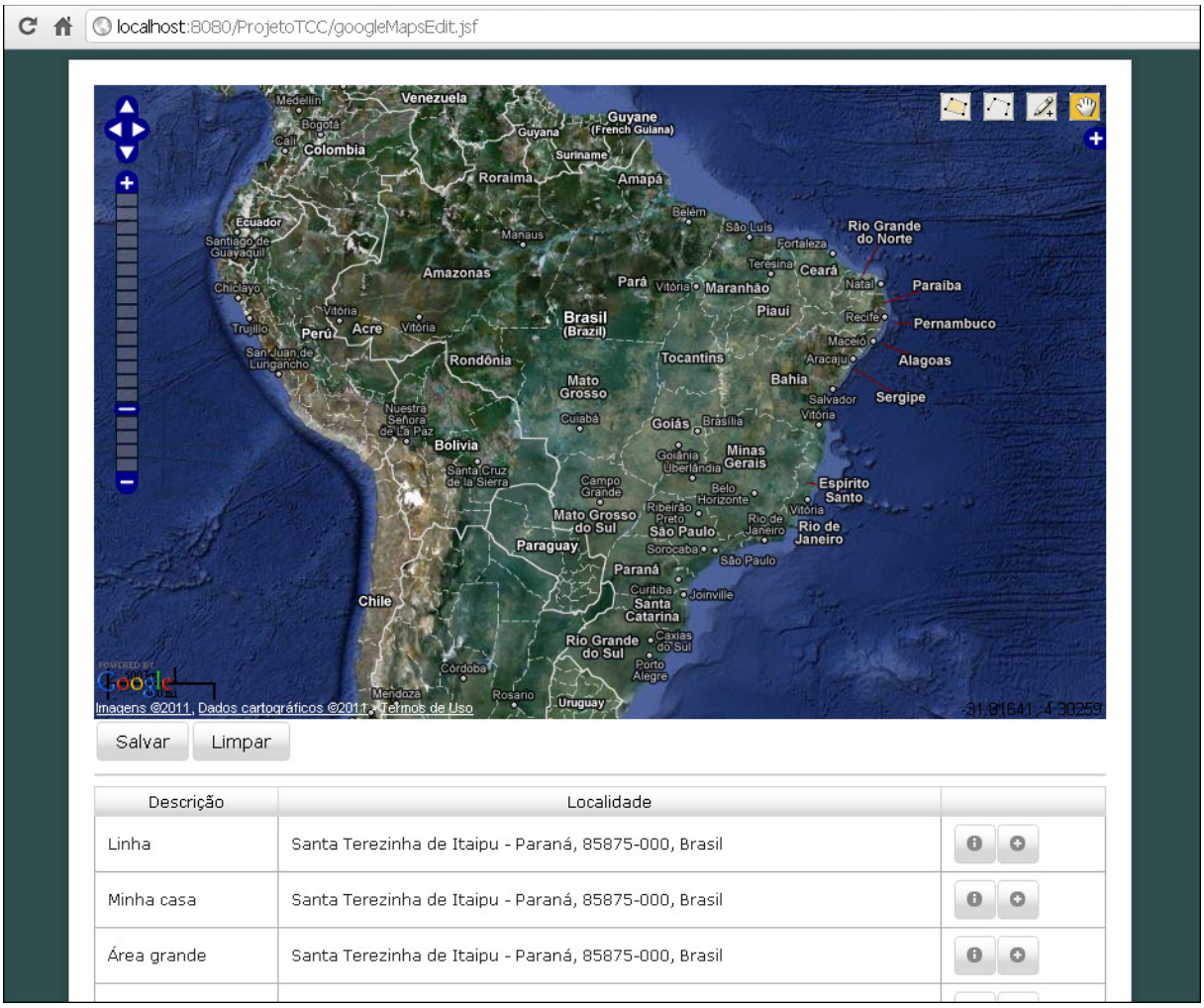

Figura 28 - Ambiente web para criação de geometrias

O primeiro passo para criação da ferramenta foi a criação da tabela do banco de dados responsável pelo armazenamento das geometrias geradas (Quadro 14).

```
CREATE TABLE googlegeoms(
        id serial PRIMARY KEY,
        descricao character varying(60),
        cidade character varying(60),
        estado character varying(60),
        cep character varying(15),
        pais character varying(60)
);

SELECT AddGeometryColumn('googlegeoms', 'the_geom', 4326, 'GEOMETRY', 2);
```

Quadro 14 - Criação da tabela googlegeoms

- *id:* identificador de chave primária;
- *descricao*: descrição da geometria;
- *cidade*: cidade referente ao centróide da geometria ou informada pelo usuário;

- *estado*: estado referente ao centróide da geometria ou informada pelo usuário;
- *cep*: CEP referente ao centróide da geometria ou informada pelo usuário;
- *pais*: país referente ao centróide da geometria ou informada pelo usuário.

4.1 CAMADAS E CONTROLES

O ambiente é composto por quatro camadas base, conforme mostrado no Quadro 15 e visualizado na Figura 30.

```
<m:googleLayer name="Google Hybrid" options="{type: G_HYBRID_MAP}"/>
<m:googleLayer name="Google Streets"/>
<m:wmsLayer name="Geometrias criadas"
            url="http://localhost:8084/geoserver/wms"
            params="{layers: 'googlegeom', transparent:true}"
            options="{visibility: false}"/>
<m:inputVectorLayer name="Editing" jsVariable="vlayer"
            value="#{googleMapEditBean.wkts}"/>
```

Quadro 15 - Camadas da visualização

Figura 29 - Controle com as camadas disponíveis

As duas primeiras são interações com a biblioteca GoogleMaps e são do tipo *G_HYBRID_MAP* (integração dos tipos *G_SATTELITE_MAP* e *G_STREET_MAP*) e *G_STREET_MAP* (endereços, ruas, rodovias, etc.).

A terceira camada, criada em formato WMS, utiliza um recurso do Geoserver criado para trazer todos os dados da tabela *googlegeoms* de forma estilizada.

A quarta camada (*inputVectorLayer*) é a que recebe as geometrias criadas e atribui a um objeto do *managedBean*.

Os controles para interação do usuário utilizados no mapa foram implementados conforme o trecho mostrado no Quadro 16. O mais importante deles, *editingToolbarControl* que aponta para a camada responsável pela criação das geometrias.

```
<m:navigationControl />
<m:panZoomBarControl />
<m:layerSwitcherControl />
<m:scaleLineControl />
<m:mousePositionControl />
<m:keyboardDefaultsControl />
<m:editingToolbarControl layer="vlayer"></m:editingToolbarControl>
<m:zoomToMaxExtentControl></m:zoomToMaxExtentControl>
```

Quadro 16 - Controles da aplicação

4.2 MANAGEDBEAN

A página *googleGeomEdit.xhtml* (Anexo A), seguindo o padrão de aplicações JSF, interage com o *managedBean GoogleMapEdit.java* (Anexo B) através da chamada de métodos e utilização de objetos.

O botão *Salvar* (Quadro 17) chama o método *prepararDados* (Quadro 18) que converte o conteúdo criado no mapa para um objeto geográfico que será persistido no banco.

```
<p:commandButton action="#{googleMapEditBean.prepararDados}" value="Salvar"
    update="formDialog" oncomplete="confirmacao.show();" />
```

Quadro 17 - Botão que chama a função prepararDados

```
public void prepararDados(){
    WKTReader reader = new WKTReader();
    if (wkts!=null){
        try {
            Geometry geom = reader.read(wkts);
            geom.setSRID(4326);
            geomEdit.setGeometria(geom);
            useGeocoding();
            geomEdit.setLocalidade(localidade);
        } catch (ParseException e) {
            e.printStackTrace();
        }
    }else{
        MessageUtil.addErrorMessage("Informe uma geometria");
    }
}
```

Quadro 18 - Método GoogleMapEdit.prepararDados

O método *useGeocoding* (Quadro 19) chama o método estático *BuscaLocalidade.busca* (Anexo C) que acessa a biblioteca Google Geocoding utilizando o centróide da geometria criada para obter detalhes aproximados referente a localização da geometria.

```
public void useGeocoding(){
    Point point = geomEdit.getGeometria().getCentroid();
    double latitude = point.getCoordinate().y;
    double longitude = point.getCoordinate().x;

    localidade = new Localidade();
    try {
        localidade = BuscaLocalidadeUtil.busca(latitude, longitude);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Quadro 19 - Método GoogleMapsEditBean.useGeocoding

Após a preparação dos dados uma caixa de diálogo é exibida com as informações obtidas e com a opção para salvar a geometria e os dados descritivos (Figura 31).

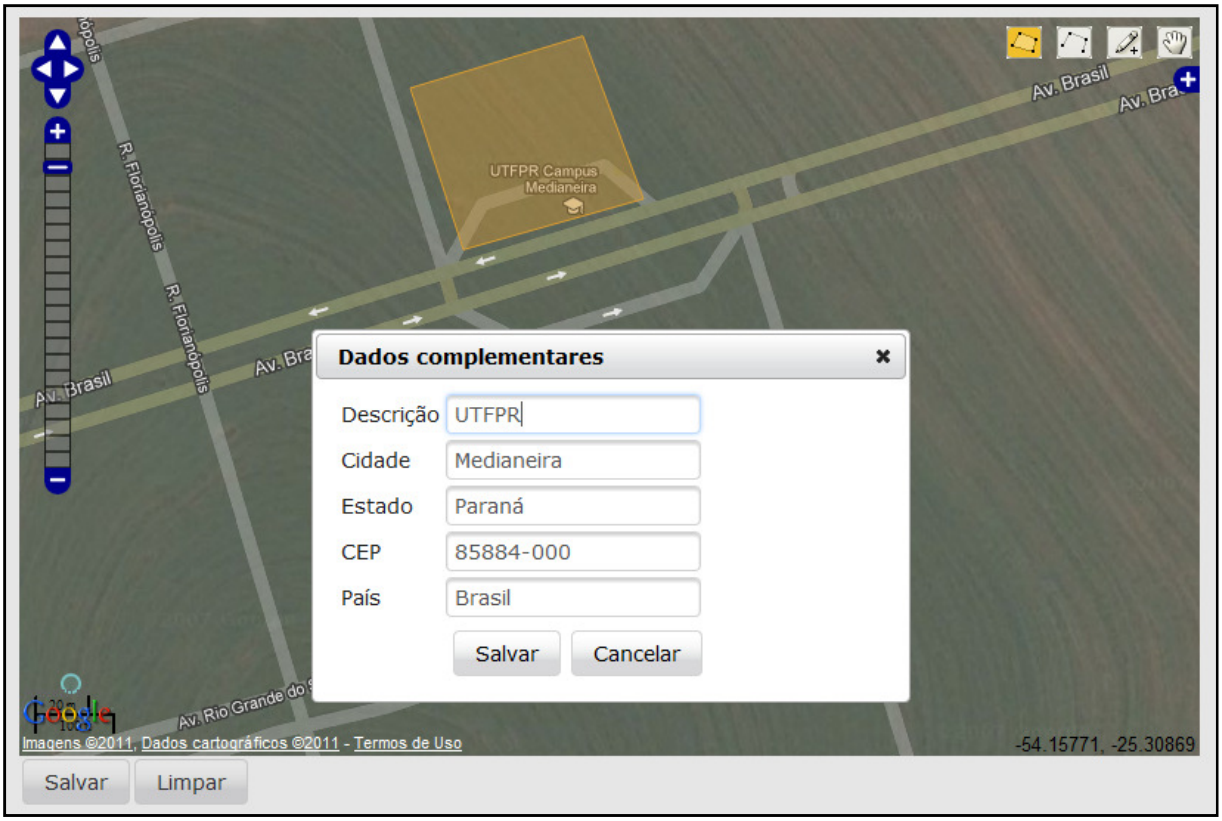

Figura 30 - Caixa de diálogo de confirmação

O botão *Salvar* chama o método *salvarGeometria* do *managedBean* que persiste os dados descritivos juntamente com a geometria criada. Após essa operação, uma mensagem informa que os dados foram inseridos e são listados na tabela disposta abaixo do mapa que contém todos os registros da tabela *googlegeoms* (Figura 32).

Figura 31 - Geometria salva e adicionada na tabela

A tabela mostra a descrição informada, informação completa sobre a localização e dois botões. O primeiro abre uma caixa de diálogo com as informações referentes ao registro e o segundo botão apresenta o registro no mapa novamente e permite que o mesmo seja alterado.

4.3 INTEGRAÇÃO COM O GEOSERVER

A integração com o servidor de aplicação, nesse caso o Geoserver, é feita através da camada que representa todos os registros inseridos no banco. Para isso,

o primeiro passo foi criar a interação do Geoserver com a tabela no PostGIS e criar a camada responsável para exibir os dados.

Ainda no servidor de mapas foi pré-definido um estilo (*green*), apenas para efeito de exemplo. Numa aplicação que tem a função de gerar mapas temáticos, por exemplo, esse estilo poderia ser personalizado. O Geoserver oferece a opção de estilização da camada baseado nos valores particulares dos registros. Um exemplo seria uma aplicação que quisesse expor dados referente a densidade demográfica de uma região. Para cada faixa de valores uma cor diferente seria exibida. Tudo isso definido no arquivo SLD mencionado anteriormente.

No caso dessa aplicação, a camada “Geometrias criadas” reflete todos os registros do banco, estilizados com a cor verde e invisível em um primeiro momento.

Ao ser selecionada no *SwitchLayer*, pode-se visualizar todos as geometrias inseridas (Figura 33).

Figura 32 - Visualização da camada Geometrias criadas

5 CONSIDERAÇÕES FINAIS

5.1 CONCLUSÃO

Após o conteúdo apresentado, é possível observar a tamanha importância dos sistemas que auxiliam no tratamento da informação geográfica e a contribuição que esses sistemas podem oferecer para diversas áreas corporativas e de pesquisa. Agilizar processos e descomplicar tarefas são conceitos chave da área de Sistemas de Informação.

Os SIGs representam a parcela mais importante da área de Geoprocessamento por propiciarem o ambiente onde as análises e diversas operações são realizadas. Por isso a necessidade de que esses sistemas sejam cada vez mais robustos e flexíveis e que acompanhem as tendências da inovação tecnológica, porém sem perder suas principais características.

Partindo de uma necessidade urgente e atual desses sistemas, o fato de conhecer a fundo as possíveis integrações com bibliotecas e *frameworks* já desenvolvidos é muito importante, pois diminui prazos e contribui para o crescimento dessas soluções, seguindo o conceito de reusabilidade, tornando o desenvolvimento muito mais produtivo.

Os objetivos desse trabalho, portanto, foram alcançados por conseguir apresentar de forma clara a possibilidade de utilização de SIGs em ambiente web, sem perder as principais vantagens de um SIG tradicional e, também, por exibir alguns exemplos de integração entre as principais tecnologias utilizadas para esse fim. Essas integrações agregam a esses sistemas recursos poderosos e opções características de uma aplicação web.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Sistemas com responsabilidades relacionadas à informação geográfica possuem uma complexidade considerável devido à quantidade de funcionalidades que eles oferecem. A área tecnológica como um todo está em constante mudança e

atualização, exigindo dos profissionais da área de Tecnologia da Informação a busca constante pelo conhecimento.

Portanto, continuar a pesquisa voltada para SIGs é um dos objetivos futuros, bem como melhorar o desempenho dessas aplicações que geralmente possuem processos custosos e dados relativamente “pesados”.

Há um interesse especial em “desvendar” as diversas funcionalidades dos SIGs-WEB relacionadas à estilização de camadas (SLD), dentre outras, com o objetivo de atender em grande escala aplicações que necessitam de mapas temáticos.

Outro objetivo futuro é pesquisar sobre outra opção de integração importante e muito útil que é o Gmaps4jsf, um *framework* semelhante ao OL4JSF que trata da integração da GoogleMaps API com o JavaServer Faces.

De uma forma geral, pretende-se ampliar o conhecimento na área de forma a acompanhar e contribuir com essas tecnologias.

REFERÊNCIAS BIBLIOGRÁFICAS

BAPTISTA, Cláudio. 2006. **Banco de Dados Espaciais**. Disponível em: www.dsc.ufcg.edu.br/~baptista/cursos/SIG/bdespacial.ppt. Acesso em: 27 set. 2011.

CAMARA, Gilberto. 2005. **Representação computacional de dados geográficos**. Disponível em: http://www.dpi.inpe.br/gilberto/livro/bdados/cap1.pdf. Acesso em: 14 set. 2011.

CAMARA, Gilberto; DAVIS, Clodoveu; MONTEIRO, Antônio M. V. 2001. **Introdução a Ciência da Geoinformação**. Disponível em: http://www.dpi.inpe.br/gilberto/livro/introd/index.html. Acesso em: 17 set. 2011.

CAMARA, Gilberto; MONTEIRO, Antônio M. V. 2001. **Conceitos Básicos em Ciência da Geoinformação. Introdução a Ciência da Geoinformação**. Disponível em: http://www.dpi.inpe.br/gilberto/livro/introd/cap2-conceitos.pdf. Acesso em: 18 set. 2011.

CAMARA, Gilberto; QUEIROZ, Gilberto R. 2001. **Arquitetura de Sistemas de Informação Geográfica. Introdução a Ciência da Geoinformação.** Disponível em: http://www.dpi.inpe.br/gilberto/livro/introd/index.html. Acesso em: 17 set. 2011.

CASANOVA, Marco; CÂMARA, Gilberto; DAVIS, Clodoveu; VINHAS, Lúbia; QUEIROZ, Gilberto R. **Bancos de Dados Geográficos**. Curitiba: MundoGeo, 2005.

COWEN, D.J. **GIS Versus CAD Versus DBMS: What Are the Differences? Photogrammetric Engineering and Remote Sensing**, 1988.

ESRI. 1998. **ESRI Shapefile Technical Description**. Disponível em <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>. Acesso em: 23 set. 2011.

FERREIRA, Nilson C. **Apostila de Sistema de Informações Geográficas**. Goiânia, 2006.

Fundamentos de Geoprocessamento. Disponível em: http://www.ltc.ufes.br/geomaticsce/Modulo%20Geoprocessamento.pdf. Acesso em: 10 out. 2011.

GEM Mapping and Design. 2009. **Gis Page**. Disponível em: http://www.gembc.ca/GIS_Page.htm. Acesso em: 11 out. 2011.

GEOSERVER. 2011. **GeoServer User Manual. Release 2.1.2**. Disponível em: http://docs.geoserver.org/stable/en/user/introduction/index.html. Acesso em: 25 set. 2011.

HAZZARD, Erik. **OpenLayers 2.10 Guide**. Birmingham, UK: Packt Publishing, 2011.

INPE. **Modelo de Dados em Geoprocessamento**. In: Curso – Banco de Dados Geográficos.

OLIVEIRA, Robert A. N. **Criação de um framework open source para construção de sistemas geoespaciais**. 2009. 82. Monografia - Universidade Tiradentes, Aracaju, 2009.

ORACLE SPATIAL. 2006. **User's Guide and Reference 10g Release 2**. Disponível em: http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14255.pdf. Acesso em: 27 set. 2011.

OZEMOY, V.M.; SMITH, D.R.; SICHERMAN, **A. Evaluating Computerized Geographic Information Systems Using Decision Analysis**. Interfaces, 1981.

PALMARES GEOPROCESSAMENTO. **Migração das bases cartográficas para PostgreSQL/PostGIS**. Foz do Iguaçu, 2006.

POSTGIS. 2010. **Manual PostGIS**. Disponível em: http://www.webgis.com.br/postgis/index.html. Acesso em: 25 set. 2011.

POSTGRESQL. 2005. **Documentação do PostgreSQL**. Disponível em: http://pgdocptbr.sourceforge.net/pg80/index.html. Acesso em: 25 set. 2011.

QUADROS, Fernando. 2010. **Introdução ao PostGIS**. Disponível em: http://www.fernandoquadro.com.br/files/PostGIS/Introducao_ao_PostGIS.pdf. Acesso em: 15 set. 2011.

RECKZIEGEL, Mauricio. 2008. **Protótipo de um Sistema de Roteamento Urbano integrando PostgreSQL e Geoserver**. Disponível em: https://sites.google.com/site/mauricioreckziegel/pos-graduacao. Acesso em: 25 set. 2011.

SUN MICROSYSTEMS. 2008. **The Java EE 5 Tutorial**. Disponível em: http://download.oracle.com/javaee/5/tutorial/doc. Acesso em: 18 out. 2011.

ANEXOS

ANEXO A – Arquivo *googleGeomEdit.xhtml*

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:m="http://www.ol4jsf.org"
      xmlns:p="http://primefaces.prime.com.tr/ui">

<ui:composition template="/template/template.xhtml">
   <ui:define name="conteudo">
      <h:form id="formPrincipal">
         <p:messages></p:messages>
         <m:map width="800px" height="500px"
                renderOnWindowLoad="false" id="mapPrincipal"
                options="{controls: [], numZoomLevels: 20, maxExtent:
                            new OpenLayers.Bounds(
                                 -73.991, -33.751,
                                 -32.378, 5.272
                             ),
                             maxResolution: 0.16255078125,
                             projection: 'EPSG:4326'}">
            <m:googleLayer name="Google Hybrid"
                           options="{type: G_HYBRID_MAP}"/>
            <m:googleLayer name="Google Streets"/>
            <m:wmsLayer name="Geometrias criadas"
                        url="http://localhost:8084/geoserver/wms"
                        params="{layers: 'googlegeom', transparent:true}"
                        options="{visibility: false}"/>
            <m:inputVectorLayer name="Editing" jsVariable="vlayer"
                                value="#{googleMapEditBean.wkts}"/>
            <m:navigationControl />
            <m:panZoomBarControl />
            <m:layerSwitcherControl />
            <m:scaleLineControl />
            <m:mousePositionControl />
            <m:keyboardDefaultsControl />
            <m:editingToolbarControl layer="vlayer">
            </m:editingToolbarControl>
            <m:zoomToMaxExtentControl></m:zoomToMaxExtentControl>
         </m:map>

         <p:commandButton action="#{googleMapEditBean.prepararDados}"
                        value="Salvar"
                        update="formDialog"
                        oncomplete="confirmacao.show();" />
         <p:commandButton action="#{googleMapEditBean.limpar}"
                        value="Limpar" update="formPrincipal"/>
         <p:separator />

         <p:dataTable value="#{googleMapEditBean.listaGeoms}" var="g"
                      styleClass="tabela">
            <p:column headerText="Descrição">
                  <h:outputText value="#{g.descricao}" />
            </p:column>
```

```
      <p:column headerText="Localidade">
            <h:outputText value="#{g.localidade}" />
      </p:column>
      <p:column>
            <p:commandButton id="btnExibeDetalhes"
                        image="ui-icon ui-icon-info"
                        oncomplete="detalhes.show()" title="Detalhes"
                        update="formDetalhes">
               <f:setPropertyActionListener value="#{g}"
                  target="#{googleMapEditBean.geomEditSelect}" />

            </p:commandButton>
            <p:commandButton id="btnVisualizar"
                        image="ui-icon ui-icon-circle-plus"
                        action="#{googleMapEditBean.selecionar}"
                        title="Adicionar a visualização"
                        update="formPrincipal mapPrincipal">
               <f:setPropertyActionListener value="#{g}"
                  target="#{googleMapEditBean.geomEditSelect}" />
            </p:commandButton>
      </p:column>
   </p:dataTable>
</h:form>
<p:dialog widgetVar="confirmacao" width="400" modal="true"
      appendToBody="true" resizable="false"
      header="Dados complementares">
   <h:form id="formDialog">
      <h:panelGrid columns="2">
         <h:outputText value="Descrição" />
         <p:inputText
               value="#{googleMapEditBean.geomEdit.descricao}" />
         <h:outputText value="Cidade" />
         <p:inputText value="#{googleMapEditBean.geomEdit.cidade}" />
         <h:outputText value="Estado" />
         <p:inputText value="#{googleMapEditBean.geomEdit.estado}" />
         <h:outputText value="CEP" />
         <p:inputText value="#{googleMapEditBean.geomEdit.cep}" />
         <h:outputText value="País" />
         <p:inputText value="#{googleMapEditBean.geomEdit.pais}" />
         <p:spacer />
         <h:panelGrid columns="2">
            <p:commandButton value="Salvar" update=":formPrincipal"
                  action="#{googleMapEditBean.salvarGeometria}"
                  oncomplete="confirmacao.hide()" />
            <p:commandButton value="Cancelar" type="button"
                  onclick="confirmacao.hide()" />
         </h:panelGrid>
      </h:panelGrid>
   </h:form>
</p:dialog>
<p:dialog widgetVar="detalhes" appendToBody="true" modal="true"
            width="400" header="Detalhes" resizable="false">
   <h:form id="formDetalhes">
      <h:panelGrid columns="2" cellpadding="5">
         <h:outputText value="Descrição:" />
         <h:outputText
            value="#{googleMapEditBean.geomEditSelect.descricao}"
            style="font-weight: bold;"/>
         <h:outputText value="Cidade:" />
         <h:outputText
            value="#{googleMapEditBean.geomEditSelect.cidade}"
```

```
                  style="font-weight: bold;"/>
               <h:outputText value="Estado:" />
               <h:outputText
                  value="#{googleMapEditBean.geomEditSelect.estado}"
                  style="font-weight: bold;"/>
               <h:outputText value="CEP:" />
               <h:outputText
                  value="#{googleMapEditBean.geomEditSelect.cep}"
                  style="font-weight: bold;"/>
               <h:outputText value="País:" />
               <h:outputText
                  value="#{googleMapEditBean.geomEditSelect.pais}"
                  style="font-weight: bold;"/>
            </h:panelGrid>
            <h:outputText value="Informações referentes à geometria"
                  style="font-style:italic; line-height: 40px;"/>
            <h:panelGrid columns="2" cellpadding="5">
               <h:outputText value="Centróide:" />
               <h:outputText
                  value="#{googleMapEditBean.geomEditSelect.centroid}"
                  style="font-weight: bold;"/>
               <h:outputText value="Tipo de Geometria" />
               <h:outputText
                  value="#{googleMapEditBean.geomEditSelect.geometria.geome
                  tryType}" style="font-weight: bold;"/>
            </h:panelGrid>
            <center><p:commandButton value="Fechar"
                        onclick="detalhes.hide()" /></center>
         </h:form>
      </p:dialog>
   </ui:define>
</ui:composition>
</html>
```

ANEXO B – Classe *GoogleGeomEditBean.java*

```
import helper.HibernateUtil;
import java.io.IOException;
import java.util.List;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.persistence.EntityManager;
import javax.persistence.Query;
import modelo.GoogleGeoms;
import modelo.Localidade;
import util.BuscaLocalidadeUtil;
import util.MessageUtil;
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.Point;
import com.vividsolutions.jts.io.ParseException;
import com.vividsolutions.jts.io.WKTReader;

@ManagedBean
@SessionScoped
public class GoogleMapEditBean {

   private EntityManager em = HibernateUtil.getEntityManager();
   private String            wkts;
   private GoogleGeoms geomEdit = new GoogleGeoms();
   private GoogleGeoms geomEditSelect = new GoogleGeoms();
```

```
private List<GoogleGeoms> listaGeoms;

private Localidade localidade;

public GoogleMapEditBean(){
   geomEdit = new GoogleGeoms();
   geomEditSelect = new GoogleGeoms();
   localidade = new Localidade();
}

public void prepararDados(){
   WKTReader reader = new WKTReader();
   if (wkts!=null){
      try {
         Geometry geom = reader.read(wkts);
         geom.setSRID(4326);
         geomEdit.setGeometria(geom);
         useGeocoding();
         geomEdit.setLocalidade(localidade);
      } catch (ParseException e) {
         e.printStackTrace();
      }
   }else{
      MessageUtil.addErrorMessage("Informe uma geometria");
   }
}

public void salvarGeometria(){
   try{
      em.getTransaction().begin();
      geomEdit = em.merge(geomEdit);
      em.getTransaction().commit();
      limpar();
      MessageUtil.addInfoMessage("Geometria salva com sucesso!");
   }catch(Exception e){
      MessageUtil.addErrorMessage("Erro: "+e.getMessage());
   }
}

public void useGeocoding(){
   Point point = geomEdit.getGeometria().getCentroid();
   double latitude = point.getCoordinate().y;
   double longitude = point.getCoordinate().x;

   localidade = new Localidade();
   try {
      localidade = BuscaLocalidadeUtil.busca(latitude, longitude);
   } catch (IOException e) {
      e.printStackTrace();
   }
}

public void limpar(){
   setWkts("");
   geomEdit = new GoogleGeoms();
}

public void selecionar(){
   geomEdit = geomEditSelect;
   setWkts(geomEdit.getGeometria().toString());
}
```

```
   public String getWkts() { return wkts; }
   public void setWkts(String wkts) { this.wkts = wkts; }

   public Localidade getLocalidade() { return localidade; }
   public void setLocalidade(Localidade localidade) {
      this.localidade = localidade;
   }

   public GoogleGeoms getGeomEdit() { return geomEdit; }
   public void setGeomEdit(GoogleGeoms geomEdit) {
      this.geomEdit = geomEdit;
   }

   public List<GoogleGeoms> getListaGeoms() {
      Query q = em.createQuery("from googlegeoms");
      listaGeoms = q.getResultList();
      return listaGeoms;
   }

   public void setListaGeoms(List<GoogleGeoms> listaGeoms) {
      this.listaGeoms = listaGeoms;
   }

   public GoogleGeoms getGeomEditSelect() {
      return geomEditSelect;
   }

   public void setGeomEditSelect(GoogleGeoms geomEditSelect) {
      this.geomEditSelect = geomEditSelect;
   }

}
```

ANEXO C – Classe *BuscaLocalidadeUtil.java*

```
import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.Iterator;
import modelo.Localidade;
import org.codehaus.jackson.JsonNode;
import org.codehaus.jackson.map.ObjectMapper;

public class BuscaLocalidadeUtil {
   private static final String URL4JSON =
            "http://maps.google.com/maps/api/geocode/json";

   private static final String ESTADO = "administrative_area_level_1";
   private static final String CIDADE = "locality";
   private static final String CEP = "postal_code";
   private static final String PAIS = "country";

   public static Localidade busca(double latitude, double longitude)
            throws IOException {
      Localidade localidade = new Localidade();
      URL url = new URL(URL4JSON+"?latlng="+latitude+","
                        +longitude+"&sensor=true&language=pt_BR");
      HttpURLConnection conn = (HttpURLConnection)url.openConnection();
```

```
        ObjectMapper om = new ObjectMapper();

        JsonNode root = om.readValue(conn.getInputStream(), JsonNode.class);;
        JsonNode results = root.get("results");
        JsonNode address_components =
              results.get(0).get("address_components");
        Iterator<JsonNode> i = address_components.getElements();
        while(i.hasNext()){
              JsonNode item = i.next();
              JsonNode type = item.get("types").get(0);

              if(type.getTextValue().equals(ESTADO))
                 localidade.setEstado(item.findValue("long_name").asText());
              else if(type.getTextValue().equals(CIDADE))
                 localidade.setCidade(item.findValue("long_name").asText());
              else if(type.getTextValue().equals(PAIS))
                 localidade.setPais(item.findValue("long_name").asText());
              else if(type.getTextValue().equals(CEP)){
                 localidade.setCep(item.findValue("long_name").asText());
              }
        }
        return localidade;
    }
}
```

ANEXO D – Classe *GoogleGeoms.java*

```
import java.io.Serializable;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import org.hibernate.annotations.Type;
import com.vividsolutions.jts.geom.Geometry;

@Entity(name="googlegeoms")
public class GoogleGeoms implements Serializable{

private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    private int id;

    private String descricao;
    private String cidade;
    private String estado;
    private String cep;
    private String pais;

    @Column(name="the_geom" )
    @Type(type = "org.hibernatespatial.GeometryUserType")
    private Geometry geometria;

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getDescricao() { return descricao; }
```

```
    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public Geometry getGeometria() { return geometria; }
    public void setGeometria(Geometry geometria) {
        this.geometria = geometria;
    }
    public String getCidade() { return cidade; }
    public void setCidade(String cidade) { this.cidade = cidade; }
    public String getEstado() { return estado; }
    public void setEstado(String estado) { this.estado = estado; }
    public String getCep() { return cep; }
    public void setCep(String cep) { this.cep = cep; }
    public String getPais() { return pais; }
    public void setPais(String pais) { this.pais = pais; }

    public void setLocalidade(Localidade localidade){
        this.cidade = localidade.getCidade();
        this.estado = localidade.getEstado();
        this.cep = localidade.getCep();
        this.pais = localidade.getPais();
    }

    public Localidade getLocalidade(){
        Localidade localidade = new Localidade();
        localidade.setCidade(this.cidade);
        localidade.setEstado(this.estado);
        localidade.setCep(this.cep);
        localidade.setPais(this.pais);
        return localidade;
    }

    public String getCentroid(){
        if (this.geometria!=null)
            return this.geometria.getCentroid().getX()+","+
                this.geometria.getCentroid().getX();
                return "";
    }

}
```