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

JULIANO CEZAR CARNIELETTO

# ESTUDO SOBRE A API GEOTOOLS PARA APLICAÇÕES BASEADAS NA PLATAFORMA JAVA SE

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA – PR
2011

JULIANO CEZAR CARNIELETTO

# ESTUDO SOBRE A API GEOTOOLS PARA APLICAÇÕES BASEADAS NA PLATAFORMA JAVA SE

Trabalho de Diplomação apresentado a disciplina de Trabalho de Diplomação graúdo curso Superior de Tecnologia em Desenvolvimento de Sistemas de Informação da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Msc. Claudio Leones Bazzi.

MEDIANEIRA – PR
2011

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

## TERMO DE APROVAÇÃO

## ESTUDO SOBRE A API GEOTOOLS PARA APLICAÇÕES BASEADAS NA PLATAFORMA JAVA SE

Por

**Juliano Cezar Carnieletto**

Este Trabalho de Diplomação (TD) foi apresentado às 14:40 h do dia 16 de junho de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Desenvolvimento de Sistemas de Informação, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

| | |
|---|---|
| Prof. (Claudio Leones Bazzi)<br>UTFPR – *Campus* Medianeira<br>(Orientador) | Prof. (Neylor Michel)<br>UTFPR – *Campus* Medianeira<br>(Convidado) |
| Prof. (Marcio Angelo Matté)<br>UTFPR – *Campus* Medianeira<br>(Convidado) | Prof. Juliano Rodrigo Lamb<br>UTFPR – *Campus* Medianeira<br>(Responsável pelas atividades de TCC) |

**DEDICATÓRIA**

*A minha filha Juliana*

# AGRADECIMENTOS

Agradeço aos meus familiares, por terem apoiado minhas decisões, me auxiliando nos momentos difíceis, dando-me bons conselhos que levarei para toda vida. Agradeço também a minha namorada Grasiele Mônica Matté, que tem ficado ao meu lado dando-me forças e um novo sentido a vida.

Obrigado ao professor orientador Claudio Leones Bazzi, por ter oferecido ajuda quando necessitei e por ter me dado à oportunidade de trabalhar junto com ele em seu projeto de doutorado. Obrigado também ao amigo Davi Marcondes Rocha e aos professores da UTFPR – Campus Medianeira, Everton Coimbra de Araújo e Marcio Angelo Matté, pelo auxilio prestado não só na elaboração desse trabalho mas também no decorrer de toda minha vida acadêmica.

*“A preguiça anda tão devagar, que a pobreza facilmente a alcança.”*

Confúcio

## RESUMO

Esse trabalho tem como objetivo realizar um estudo sobre a API (*Application Programming Interface*) Geotools e banco de dados PostgreSQL, juntamente com sua extensão espacial PostGIS. Este estudo contempla a apresentação da API, como também a demonstração de suas funcionalidades, no que tange a criação de interfaces para visualização de dados geograficamente referenciados. Objetivou-se ainda uma breve revisão de literatura sobre o Sistema Gerenciador de Banco de Dados (SGDB) PostgreSQL e de sua extensão PostGIS, demonstrando a forma de conexão com a API Geotools.



**Palavras-chave:** Sistemas de Informação Geográfica, Java, Geotools

## ABSTRACT

This work aims to conduct a study on the API (Application Programming Interface) Geotools and PostgreSQL database, along with its spatial extension PostGIS. This study includes the presentation of the API, as well as a demonstration of its features, regarding the creation of interfaces to display geographically referenced data. The objective is also a brief literature review on the System Manager Database (DBMS) PostgreSQL PostGIS and its extension, demonstrating the means of connecting to the API Geotools.



**Keywords**: Geographic Information Systems, Java, Geotools

# LISTA DE FIGURAS

# LISTA DE QUADROS

# LISTA DE SIGLAS

| | | |
|---|---|---|
| API | - | *Application Programming Interface* |
| BLOB | - | *Binary Large Object* |
| ESRI | - | *Environmental Systems Research Institute* |
| GB | - | *Gigabyte* |
| GIS | - | *Geographic Information System* |
| GNU | - | *GNU is Not Unix* |
| GPL | - | *General Public License* |
| GRASS | - | *Geographic Resources Analysis Suport* |
| JAR | - | *Java Archive* |
| JDK | - | *Java Development Kit* |
| JRE | - | *Java Runtime Environment* |
| OGC | - | *Open Geospatial Consortium* |
| SGBD | - | *Sistema Gerenciador de Banco de Dados* |
| SGBDOR | - | *Sistema Gerenciador de Banco de Dados Objeto Relacional* |
| SIG | - | *Sistemas de Informação Geográfica* |
| SQL | - | *Structured Query Language* |
| SO | - | *Sistema Operacional* |
| TB | - | *Terrabyte* |
| XML | - | *Extensible Markup Language* |

# SUMARIO

# 1 INTRODUÇÃO

*Softwares* de geoprocessamento estão cada vez mais sendo utilizados pelas diversas áreas do conhecimento (Agricultura, Meio Ambiente, Saúde, Transportes e estudos climáticos), passando a ser considerados fundamentais na busca de resultados mais relevantes e precisos. A procura por este tipo de software tem feito com que desenvolvedores se especializem e passem a incorporar funções geográficas em seus softwares tradicionais, dando suporte para alcançar resultados que até então não eram possíveis sem a consideração da localização dos dados no espaço. Neste contexto, se faz necessário o uso de bibliotecas específicas para se trabalhar com este tipo de dado, considerando a complexidade de representação do espaço e suas características. Geotools e uma biblioteca desenvolvida em Java[1], seguindo os padrões internacionais do *Open Geospatial Consortium* (OGC) e permite desenvolver soluções adaptadas aos padrões internacionais em vigor, para desenvolvimento de *softwares* voltado a informações geográficas. Trata-se de uma API (*Application Programming Interface*) de código aberto, que conta com um grande número de colaboradores atuando no seu continuo desenvolvimento. Esse estudo tem por finalidade demonstrar algumas funcionalidades dessa API, considerando o aspecto de apresentação de dados espaciais, bem como a forma de conexão com o SGDB PostgreeSQL, que faz uso da sua extensão PostGIS para a manipulação de dados espaciais.

[1] Java é uma linguagem de programação orientada a objetos desenvolvida na década de 90 por uma equipe de programadores da Sun Microsystens coordenados por James Gosling.

## 1.1 OBJETIVO GERAL

Esse trabalho tem por objetivo realizar referencial teórico sobre a API Geotools, e desenvolver uma aplicação que demonstre algumas das funcionalidades dessa API para desenvolvimento de SIG´s (Sistemas de Informação Geográfica).

## 1.2 OBJETIVOS ESPECÍFICOS

- Realizar um referencial teórico sobre funcionalidades da API Geotools.
- Apresentar a estrutura de conexão com o banco de dados PostgreSQL.
- Criar uma interface para apresentação de *Layers* oriundas de arquivos no formato *shapefile* e geometrias do tipo Polígono[2] e Ponto[3].
- Demonstrar ação de funcionalidades fornecidas pela API para manipulação das *Layers*.

[2] Polígono é uma figura geométrica plana limitada por uma linha poligonal fechada
[3] Ponto é um elemento do espaço que indica uma posição

# 2 SISTEMAS DE INFORMAÇÕES GEOGRÁFICAS

Sistemas de Informação Geográfica (SIG´s) são *softwares* para tratamento e a manipulação de dados geográficos, os quais vem sendo utilizados em diversas áreas do conhecimento: Agricultura (Estudos detalhados sobre a variabilidade de atributos químicos e físicos do solo e da produtividade; Mapeamento de doenças; Controle de plantas Invasoras);, Saúde (Controle de focos de doenças); Segurança pública (Mapeamento da criminalidade, apoiada na localização onde os mesmos ocorrem); Transporte (Criação de rotas mais econômicas para as frotas; Rastreamento de Veículos); entre outras. Para Camargo (1997), os dados georreferenciados complementam a representação de objetos e fenômenos sendo indispensável para analisá-la de forma precisa.

Apesar da demanda deste tipo de *software* estar em pleno crescimento, há dificuldades em sua utilização, pois está totalmente baseado em conceitos de Geoprocessamento, que corresponde a uma área multidisciplinar que envolve conhecimentos de diferentes disciplinas (Geografia, Cartografia, Ciência da Computação, Sensoriamento Remoto, entre outras) (CAMARGO, 1997), além da necessidade de conhecimento da área onde é aplicada. Para ser considerado um Sistema de Geoprocessamento, deve ser capaz de capturar, processar e gerenciar dados espaciais, compreendendo softwares como: Sistemas de Cartografias Automatizadas, Sistemas de Processamento de imagens, Sistemas de CAD, mas principalmente os SIGs.

Os SIGs segundo Camargo (1997), apresentam duas características importantes por possibilitarem a integração de dados provenientes de diversas fontes como dados cartográficos (dados de censo e cadastro urbano, imagens de satélites e modelos numéricos do terreno) e por apresentarem a capacidade de recuperar, manipular e visualizar estes dados através de algoritmos de manipulação e análise. Permitem uma forma consistente de consultas envolvendo dados geográficos e alfanuméricos, possibilitando por exemplo o acesso a registros de imóveis a partir de sua localização geográfica (DAVIS, 2003).

Os SIGs são suportados ou desenvolvidos de forma integrada por algum SGDB (ESRI, 1991), sendo que para Ooi (1991), dados gerenciados pelos SIGs podem ser classificados como convencionais, espaciais e pictórios, sendo possível por meio destas estruturas o armazenamento de dados sobre a localização geográfica, características estruturais, geométricas e topologias de entidades pertencentes a um determinado domínio. Como afirma Camara (1995) um dado espacial ou geográfico possui uma localização expressa como

coordenadas de um mapa e atributos descritivos representados num banco de dados convencional, representando-se a natureza dual da informação trafegada em um SIG. Dados geográficos não se apresentam de forma isolada no espaço, sendo tão importante quanto localizá-los, descobrir e representar as relações entre as diversas características representadas.

Sistemas computacionais voltados ao geoprocessamento podem lidar com tipos de dados espaciais distintos, onde podem representar variações espaciais contínuas (Geo-Campos) ou individualizáveis e com identificação (Objetos Geográficos), cada um tendo como foco um tipo de representação que objetiva principalmente a melhor representação do espaço e o desempenho computacional respectivamente.

## 2.1 ARQUITETURA DE UM SIG

Como dito anteriormente SIGs são sistemas computacionais, sendo assim podem ser distribuídos de diversas formas, através de um servidor, no caso de sistemas voltados à internet, através de redes internas, no caso de aplicações Desktop distribuídas, ou até mesmo aplicações que rodam localmente com a base de dados instala na própria máquina, fazendo dessa, uma máquina cliente/servidor. Mas segundo ARONOFF (1989) os seguintes itens descrevem um SIG:

- Entrada de Dados - Estes componentes convertem dados de seu formato original para àquele que pode ser utilizado em um SIG.
- Gerenciamento de Dados – O componente de gerenciamento de dados inclui aquelas funções necessárias para armazenar e recuperar dados de uma base de dados.
- Análise e manipulação de dados – As funções de análise e manipulação de dados determinam as informações que podem ser geradas pelo SIG.
- Saída de dados – As funções de saída ou de geração de relatórios são muito semelhantes nos sistemas de informações geográficas. A variação está mais ligada à qualidade, acurácia e a facilidade de uso. Estes relatórios podem ser no formato de mapas, tabelas de valores, texto impresso ou em texto disponível em arquivo eletrônico.

Outra definição para a estrutura de um SIG pode ser dada por WORBOYS (1995), sendo ela o gerenciamento de dados onde inclui funções, necessárias para armazenar e recuperar informações de uma base de dados. A análise e a manipulação de dados determinam as informações que podem ser geradas pelo SIG.

Em resumo pode se dizer que um SIG é formado pelos seguintes componentes:

- Interface com o usuário;
- Entrada e integração de dados;
- Funções de consultas e análise espacial;
- Visão e plotagem;
- Armazenamento e recuperação de informações.

Estes componentes se relacionam de forma hierárquica. No nível mais próximo ao usuário, a interface homem-máquina define como o sistema é operado e controlado. No nível intermediário, um SIG deve ter mecanismos de processamento de dados espaciais (entrada, edição, análise, visualização e saída). No nível mais interno do sistema, um sistema de gerência de bancos de dados geográficos oferece armazenamento e recuperação dos dados espaciais e seus atributos. De uma forma geral, as funções de processamento de um SIG operam sobre dados em uma área de trabalho em memória principal. A ligação entre os dados geográficos e as funções de processamento do SIG é feita por mecanismos de seleção e consulta que definem restrições sobre o conjunto de dados. Exemplos ilustrativos de modos de seleção de dados são:

- "Recupere os limites da Região Sul" (restrição por definição de região de interesse);
- "Recupere as cidades da Região Sul com população entre 100.000 e 500.000 habitantes" (consulta por atributos não-espaciais).
- "Mostre os postos de saúde num raio de 15 km do hospital Ministro Costa Cavalcante de Foz do Iguaçu - PR" (consulta com restrições espaciais).

A Figura 1 indica o relacionamento dos principais componentes ou subsistemas de um SIG. Cada sistema, em função de seus objetivos e necessidades implementa estes componentes de forma distinta, mas todos os subsistemas citados devem estar presentes num SIG.

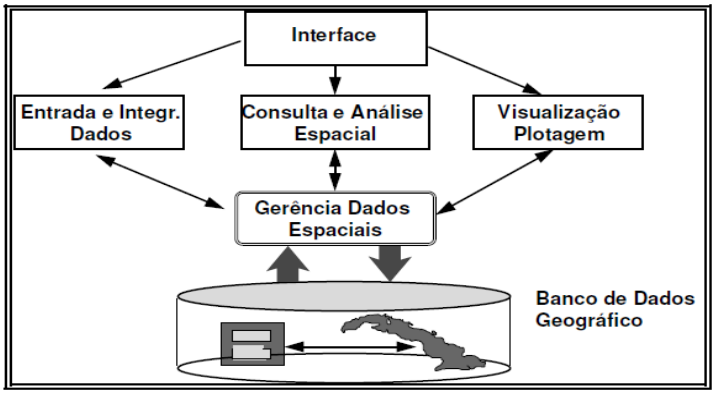


Figura 1 - Componentes de um SIG
Fonte: Marco Antonio Casanova, et. al (2005).

A principal diferença entre os SIGs é a forma como os dados geográficos são gerenciados. Existem três diferentes arquiteturas de SIG que utilizam os recursos de um SGBD, são elas:

- Dual;
- Integrada baseada em SGBD relacionais; e
- Integrada baseada em extensões espaciais sobre SGBDOR (Sistema Gerenciador de Bando de Dados Objeto Relacionais).

## 2.2 DUAL

Na arquitetura dual, utiliza-se um SGBD relacional para armazenar os atributos convencionais dos objetos geográficos e arquivos para guardar as representações geométricas destes objetos. No modelo relacional, os dados são organizados na forma de uma tabela onde as linhas correspondem aos dados e as colunas correspondem aos atributos. A entrada dos atributos não-espaciais é feita por meio de um SGBD relacional e para cada entidade gráfica inserida no sistema é imposto um identificador único ou rótulo, através do qual é feita uma ligação lógica com seus respectivos atributos não-espaciais armazenados em tabelas de dados no SGBD. A Figura 2 demonstra a ligação entre a entidade gráfica e seus atributos não-espaciais.

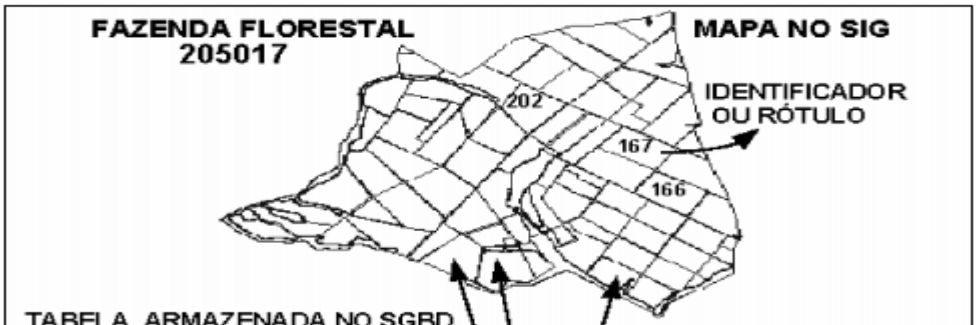


TABELA ARMAZENADA NO SGBD

| CODPAR | CODFAZ | CODPROJ | TALHAO | AREATOT | DTPLAN |
|---|---|---|---|---|---|
| 5 | 205017 | 105 | *147* | 9,91 | 31/10/89 |
| 5 | 205017 | 105 | *148* | 25,66 | 18/12/89 |
| 5 | 205017 | 068 | *152* | 26,34 | 7/10/93 |
| 5 | 205017 | 068 | *153* | 21,65 | 14/10/93 |
| 5 | 205017 | 068 | *154* | 27,90 | 21/10/93 |
| 5 | 205017 | 068 | *155* | 23,52 | 23/11/93 |
| 5 | 205017 | 109 | *162* | 26,29 | 5/11/89 |
| 5 | 205017 | 109 | *163* | 27,57 | 9/11/89 |

CODPAR = código do parque florestal ; CODFAZ = código da fazenda
CODPROJ = código do projeto; TALHÃO = número do talhão
AREATOT = Área total plantada; DTPLAN = Data do plantio

Figura 2 - Esquema de ligação entre dados em uma arquitetura Dual
Fonte**:** http://www.dpi.inpe.br/gilberto/livro/introd/cap3-arquitetura.pdf.

- Pode ser citado como principal vantagem dessa estratégia a possibilidade de utilização de SGDBs convencionais de marcado, tornando-se possível a escolha por bancos livres e de código aberto. Mas a estratégia Dual também possui desvantagens como as citadas abaixo:
- Dificuldade na otimização de consultas;
- Dificuldade na gerência de transações;
- Dificuldades no controle e manipulação dos dados espaciais;
- Dificuldade em manter a integridade entre componentes espaciais e componentes alfanuméricos;
- Consultas mais lentas, pois são processadas separadamente;
- Falta de interoperabilidade entre os dados.

Cada sistema produz seu próprio arquivo proprietário sem seguir um formato padrão, o que dificulta a integração destes dados.

A Figura 3 apresenta a arquitetura Dual.

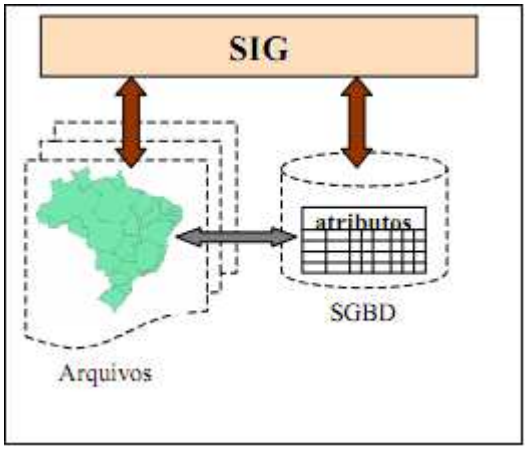


Figura 3 - Sistema de arquitetura Dual
Fonte: Marco Antonio Casanova, et. al. (2005).

## 2.3 ARQUITETURA INTEGRADA

Na arquitetura integrada tanto os dados espaciais quanto os dados alfanuméricos são armazenados em um SGDB qualquer, isso tem como principal vantagem a utilização dos recursos da base de dados para o controle e manipulação de dados espaciais, como gerência de transações, controle de integridade e concorrência. Sendo assim, a manutenção de integridade entre os componentes espaciais e alfanuméricos é feita pelo SGBD. Há duas alternativas para a arquitetura integrada, baseada em SGBDs relacionais e baseada em extensões espaciais de SGBDORs. A Figura 4 demonstra a arquitetura integrada.

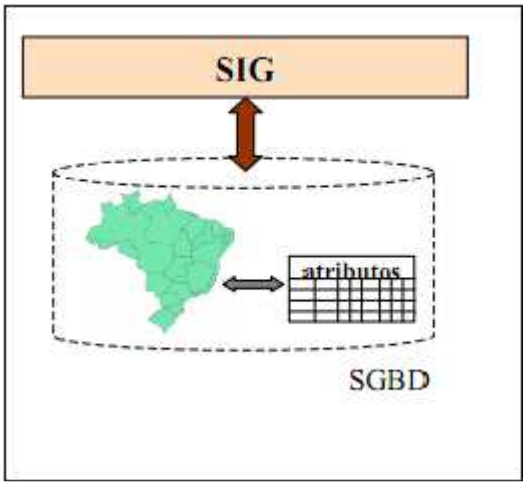

Figura 4 - Sistema de arquitetura integrada
Fonte: Marco Antonio Casanova, et. al. (2005).

A arquitetura integrada baseada em um SGBD relacional utiliza campos BLOB (*Binary Large Object*), para armazenar os dados espaciais. Porém isso traz desvantagens, pois como o SGBD trata o campo longo como uma cadeia binária, não é possível conhecer a semântica do seu conteúdo, além disso, os métodos de acesso espacial e as consultas devem ser implementadas pelo SIG, devido ao fato de o SGBD tratar os dados espaciais como uma cadeia binária, por isso não possui mecanismos satisfatórios para o seu tratamento. Soma-se ainda o fato da linguagem SQL ser limitada para tratamentos de campos longos.

O outro tipo de arquitetura integrada consiste em utilizar extensões espaciais desenvolvidas sobre SGBDORs. Estas extensões contêm funcionalidades e procedimentos que permitem armazenar, acessar e analisar dados espaciais de formato vetorial. Como desvantagens dessa arquitetura podem ser citadas as faltas de mecanismos de controle de integridade sobre os dados espaciais e a falta de padronização das extensões da linguagem SQL. Os SGBDORs, também chamados de SGBDs extensíveis, oferecem recursos para a definição de novos tipos de dados e de novos métodos ou operadores para manipular esses tipos, estendendo assim seu modelo de dados e sua linguagem de consulta. Por isso, um SGBDOR é mais adequado para tratar dados complexos, como dados geográficos, do que um SGBD, o qual não oferece esses recursos.

Um SGBDOR que possui uma extensão para tratar dados espaciais deve conter as seguintes características:

- Fornecer tipos de dados espaciais (TDEs), como ponto, linha e região, em seu modelo de dados e manipulá-los assim como os tipos alfanuméricos básicos (inteiros, string, etc).

- Estender a linguagem de consulta SQL para suportar operações e consultas espaciais sobre TDEs;
- Adaptar outras funções de níveis mais internos para manipular TDEs eficientemente, tais como métodos de armazenamento e acesso, e métodos de otimização de consultas.

Portanto, além dos TDEs, as extensões espaciais fornecem operadores e funções que são utilizados, juntamente com a linguagem de consulta do SGBD, para consultar relações espaciais e executar operações sobre TDEs. Além disso, fornecem métodos de acesso eficiente de TDEs através de estruturas de indexação, como R-tree e QuadTree.

# 3 POSTGRE SQL

Trata-se de um SGDBOR de código aberto e pode ser executado em vários Sistemas Operacionais (Baseados em arquitetura livres[4] ou sobre uma licença paga). Possui funcionalidades como replicação assincrona, registrador de transações sequencial, um sofisticado planejador de consultas, recuperação em um ponto no tempo, controle de concorrência multiversionado, entre outras funcionalidades (Postgres, 2010). Dentre suas principais características estão o tamanho ilimitado, possibilidade de criação de tabelas com capacidade de até 32 TB, tuplas com capacidade de até 1.6 TB e atributos com limite de 1GB de capacidade de armazenamento. Além disso, não possui restrição quanto a quantidade de tuplas que podem ser armazenadas em uma tabela e podendo ser criadas em um única tabela até 1600 colunas (dependendo do tipo de dado armazenado).

[4] Softwares livres são sistemas que podem ser usados, copiados, estudado e redistribuído sem qualquer restrição

# 4 POSTGIS

PostGis é uma extensão espacial do SGBD PostgreSQL, que fornece suporte para o trabalho com objetos geográficos e permite que o servidor do PostgreSQL, seja usado como apoio para execução de funções espaciais para sistemas GIS (*Geographic Information Systen)*, assim como ocorre como a extenção do *Oracle Spatial* da Oracle. O PostGIS segue o padrão OGC e foi certificado como compatível com os tipos e função de perfil geo espacial (Postgis, 2010). Vem sendo desenvolvido como software de código aberto e é liberado sobre a licensa GNU (*GNU is Not Unix*) GPL (*General Public License*). A comunidade colaboradora do PostGIS continua seu desenvolvimento, trabalhando na parte de interface com usuários, suporte a topologias básicas, a validação de dados e transformação de coordenadas.

A extensão espacial do SGDB PostgreSQL, vem se tornando cada vez mais conhecida e utilizada em sistemas GIS. Sistemas como o GRASS(*Geographic Resources Analysis Suport*) e GeoServer são exemplos de sistemas que fazem uso da poderosa extensão espacial (Postgis, 2010).

# 5 SHAPEFILES

Arquivos com extensão *Shape* correspondem a um formato de arquivo que contém informações geo-espacial em formato vetorial para serem usados nos sistemas de informações geográficas. Desenvolvido e mantido pela ESRI (*Environmental Systems Research Institute*) o *shapefile* é um formato de arquivo proprietário, porém mantém uma interoperabilidade com diversos softwares, fazendo com que ganhe grande popularidade (ESRI, 1998).

Um arquivo *.shp* é composto por três arquivos, sendo um arquivo principal, um arquivo de indexação e uma tabela dBASE. No arquivo principal (*.shp*) são guardadas as feições geométricas, no arquivo de índices (*.shx*) cada registro contém uma referência dos dados do arquivo principal, e o arquivo dBASE (*.dbf*) contém uma tabela com os dados alfanuméricos dos atributos espaciais (ESRI, 1998).

# 6 JAVA SE

A plataforma Java SE é uma ferramenta de desenvolvimento para a plataforma *Java*, que contém todo o ambiente necessário para a criação e execução de aplicações na linguagem *Java*. A tecnologia é composta de dois produtos, pelo *JDK* (*Java Development Kit*) e pelo *JRE* (*Java Runtime Environment*). O *JDK* é um kit de desenvolvimento *Java* fornecido pela *Oracle* e é constituído por um conjunto de programas que engloba compilador, interpretador e utilitários que fornecem um pacote de ferramentas para o desenvolvimento de aplicações *Java*. Atualmente se encontra na versão *6 update 21*. O *JRE* fornece bibliotecas, máquina virtual e outros componentes para executar *applets*[5] e aplicativos.

[5] Applet é um software aplicativo que é executado através de um sistema e geralmente executa uma função específica.

# 7 GEOTOOLS

GeoTools é uma API Java de código aberto que implementa especificações do *Open Geospatial Consortium* (OGC) para trabalhar com dados vetoriais e do tipo *raster*. Fornece várias Classes[6] e Interfaces[7] para introduzir as funcionalidades espaciais a todas as oferecidas pelo JavaSE.

É composta por um grande numero de módulos que permitem:

- Acessar dados espaciais em formatos de arquivos e banco de dados espaciais;
- Trabalhar com um grande número de projeções;
- Filtrar e analisar dados espaciais e não espaciais;
- Montar e exibir mapas com estilos complexos;
- Criar e analisar gráficos e redes.

É uma API utilizada no desenvolvimento de aplicações voltadas ao Geoprocessamento, tornando possível a criação de sistemas que vão do simples, podendo citar como exemplo pequenas aplicações SIG até mesmo aplicações maiores como servidores de Mapas. Geotools permite a exploração de dados contidos em arquivos armazenados fisicamente em discos, como também a comunicação com SGDBs, dessa maneira permite o desenvolvimento tanto de aplicações com arquitetura DUAL, quanto com arquitetura integrada. Proporciona ainda a separação entre:

- Os recursos responsáveis pela exploração de valores, ou modelo de dados;
- Os modelos de consultas que correspondem aos filtros; e
- Expressões usadas para consultas e recuperação de dados dos modelos e construção de consultas de modelos e metadados[8].

[6] Classes representam conjuntos de objetos com características afins

[7] Interfaces podem ser consideradas um contrato entre a classe e o mundo externo ela fornece métodos e atributos para a classe que a implementa

[8] Metadados são dados que representam informações a respeito de outro dado.

## 7.1 LICENÇA

A licença LGPL (*Lesser General Public License*) permite que sejam construida aplicações comerciais que façam uso da API Geotools e redistribuir a aplicação sob qualquer licença a critério do desenvolvedor. Os usuários, porém devem receber o código-fonte da biblioteca conforme a licença LGPL especifica. Se o usuário optar por realizar alterações na API, o código fonte deve ser publicado, tendo como maneira mais fácil a apresentação das alterações para o projeto Geotools para que possam se incorporads ao núcleo da API. Da mesma maneira que pode ser utilizadas em softwares proprietários a API também pode ser utilzada e softwares livres, sob a licença GPL, seguindo os mesmo termos da licença LGPL.

## 7.2 CONHECENDO A API

A API como demonstrada na página oficial do projeto (Figura 5), pode ser melhor entendida quando distribuída em camadas. As camadas representam JARs[9] (Java Archive), responsáveis pelas funcionalidades das camadas que compõem a API.

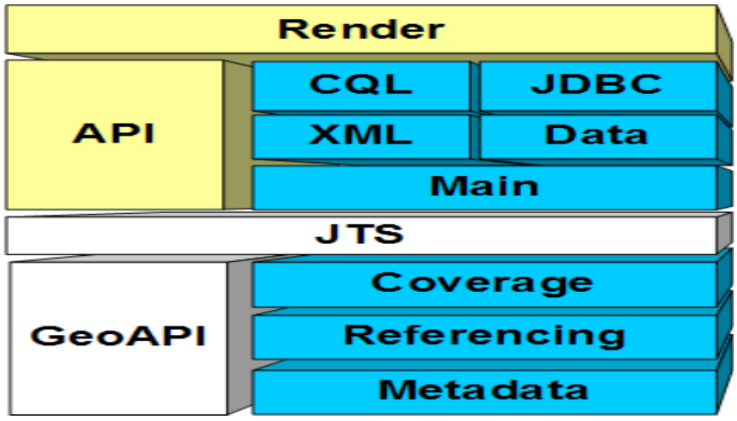


Figura 5 - Representação em camadas da API Geotools
Fonte: http://docs.geotools.org/latest/userguide/welcome/geotools.html

A camada API que nesse caso não faz referencia a *Application Programming Interface*, mas ao módulo de Geotools, é onde ficam armazenadas as interfaces estáveis que são implementadas pela biblioteca. Essas interfaces fornecem modelos para criação de componentes de conexão e apresentação das camadas. A camada *Renderer* é responsável por

[9] JARs são arquivos compactados utilizados para distribuir um conjunto de classes Java.

suportar o processamento de informações espaciais fazendo uso da API Java2D[10]. Ela possui interface como a *Style* responsável por definir como uma camada será apresentada ao usuário. Alguns conceitos precisam ser compreendidos ao trabalhar com essa API, como por exemplo, que os Dados apresentados ao usuário são vistos como camadas, a classe de Geotools que representa uma camada é chamada *Layer*.

Para realizar a apresentação de um mapa completo, muitas vezes são necessárias várias camadas. Por exemplo, se formos representar o mapa de criminalidade de uma cidade qualquer, seria preciso pelo menos duas camadas, uma para representar a área municipal delimitando as fronteiras do município (um polígono) e outra mostrando os locais onde os crimes ocorreram (pontos). Essa é uma simples situação hipotética do uso de camadas em se trabalhando com a API Geotools. Essas camadas precisaram aparecer visualmente para o usuário da aplicação, sendo assim, a elas precisam ser inseridos componentes gráficos que devem definir os atributos dessas camadas, como a cor por exemplo.

*Feature* é a interface responsável por criar objetos que representem as estruturas de dados espaciais. Geotools fornece uma série de implementações prontas para o uso de *Feature*, como poderá ser observado no decorrer desse estudo. Essas estruturas de dados possuem um tipo, ou "*FeatureType*" como é tratado pela API. *FeatureType* fornece um modelo de *metadados* que descrevem as informações representadas no *Feature*. Continuando a usar a situação do mapa de criminalidade seria como se a camada que representa a cidade teria seus dados armazenados em um *FeatureType* ou seja, um conjunto de informações a respeito de um determinado tipo de dado e a camada que representa os crimes em outro, dois conjuntos de dados distintos para dois tipos de dados também distintos, tanto no que se trata de tipo de dado como de informações que contém.

O fato de se trabalhar com objetos Java torna o sistema dinâmico, pois assim os dados espaciais contidos nessas estruturas podem ser alterados e manipulados em tempo de execução.

[10] API Java2D é um conjunto de classes para o trabalho com imagens em duas dimensões

## 7.3 PLUGINS

São pacotes de funcionalidades especificas para que as aplicações construidas com ela suportem diversos formatos de dados. Podem ser citados como exemplos os plugins postgis e oracle, para comunicação com as extensões espaciais oferecidas pelos SGDBs PostgreSQL e Oracle respectivamente. A Figura 6 apresenta alguns plugins fornecidos pela API.

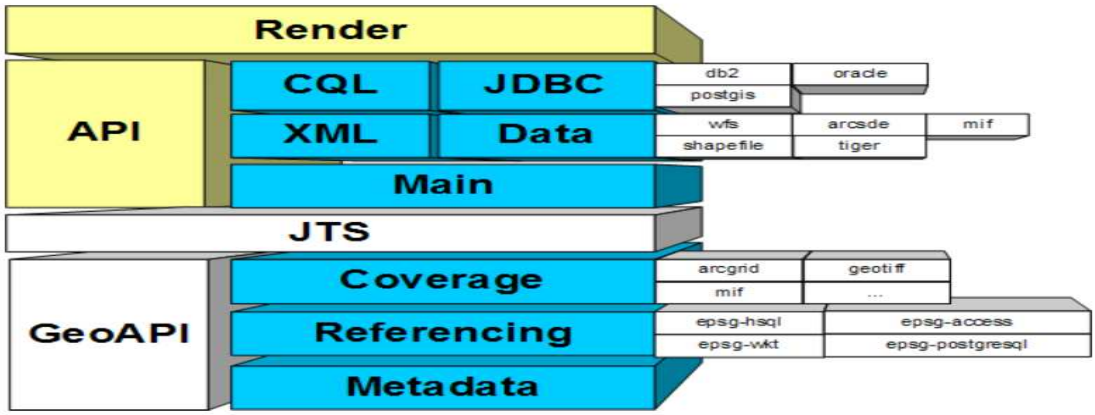


Figura 6 - Sistema de camadas dos *plugins* da API Geotools
Fonte: http://docs.codehaus.org/display/GEOTDOC/02+Meet+the+GeoTools+Library

# 8 CRIAÇÃO DE UMA INTERFACE GRÁFICA PARA APRESENTAR ALGUMAS FUNCIONALIDADES DA API GEOTOOLS

A interface que será desenvolvida para esse estudo tem por finalidade apresentar algumas das funcionalidades da API Geotools. Como citado anteriormente, Geotools fornece métodos para acesso e manipulação de dados espaciais de maneira facilitada, fornecendo ainda acesso direto a arquivos que contenham essas informações e SGDBs que armazenam e manipulam esses dados através de suas extensões espaciais. Foi elaborado um caso de uso exemplo para demonstrar as funcionalidades desta API, compreendendo:

- Criação de uma tela principal que irá manter todas as funcionalidades da aplicação;
- Carregamento de dados oriundos de arquivos *shapefile*;
- Carregamento de dados oriundos do SGDB PostgreSQL;
- Criação de estilos para as camadas que serão apresentadas;
- Aplicação de funcionalidades nas camadas apresentadas.

Para o desenvolvimento da aplicação foi utilizada a versão 2.6.5 da API Geotools, foi usado também o banco de dados PostgreSQL versão 8.4 e a sua extensão PostGIS versão 1.5.2. Para o desenvolvimento da interface gráfica foi utilizado o conjunto de classes Java Swing[11].

Essa interface exemplo foi dividida em três pacotes, formando o modelo MVC[12] (*Model View Controller*). Para os serviços exigidos pela aplicação foi criado o pacote "*controle*" que é responsável por manter toda a parte de serviços, conexões com banco de dados, busca de arquivos *shape*, etc. Para as classes de camada visual foi criado o pacote "*visao*", nele ficam contidas a classes que formam as telas e componentes como botões, menus e itens de menus. Para manter interfaces, classe de exceções, arquivos de configuração e pacote de imagens, foi criado o pacote "*modelo*". No sub-pacote de modelo, "*interfaces"*, é onde fica localizada a interface "*Command"* responsável por manter o método "*execute*", ela é implementada pelas classes que representam itens do menu e os botões da aplicação para o

[11] Swing é um conjunto de classes Java que permite a criação de interfaces gráficas de alto nível.

[12] MVC é um padrão de desenvolvimento de software que visa separar a lógica de negocio da lógica de apresentação.

tratamento de eventos segundo o padrão de projeto[13] (*Design Pattern*) *Command*[14]. Outro sub-pacote de modelo, o "*facade*", é responsável por armazenar a classe "*FacadeServicos*", essa é utilizada na implementação de outro padrão de projeto o Facade[15], responsável pelo acesso da interface gráfica aos métodos oferecidos pelas classes do pacote de serviços. A Figura 7 demonstra a estrutura de pacotes da aplicação.

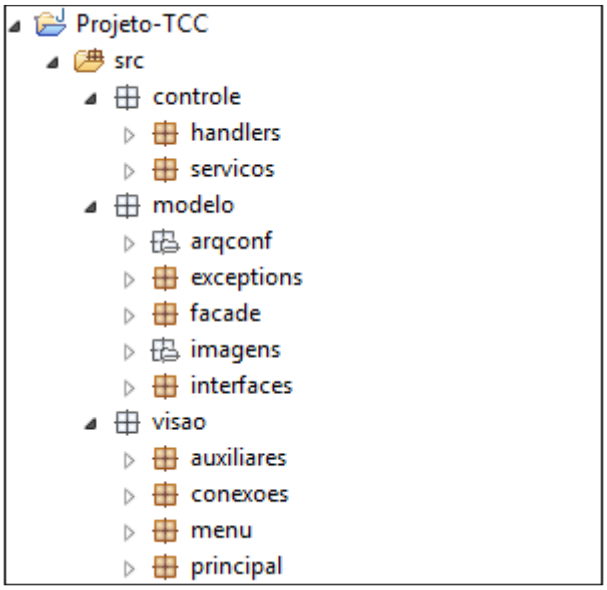


Figura 7 - Estrutura da aplicação exemplo
Fonte: Autoria própria

## 8.1 CRIAÇÃO DA TELA PRINCIPAL

A API Geotools fornece uma classe chamada *JMapFrame*, essa classe é derivada da classe *JFrame* do pacote "*javax.swing*", ela fornece as funcionalidades da classe JFrame, aliado a funcionalidades específicas para o trabalho com objetos da API utilizados na apresentação das camadas. *JMapFrame* possibilita facilmente a criação de uma interface para apresentação dessas camadas como é demonstrado no Quadro 1.

[13] Padrões de projetos descrevem soluções para problemas decorrentes no desenvolvimento de sistemas orientados a objetos.

[14] Command é um padrão de projeto que tem por objetivo encapsular uma requisição como um objeto.

[15] Facade é um padrão de projeto que consiste em fornecer uma interface para acesso a funcionalidades de um módulo ou sistema.

```
111     public void criaMapFrame() throws Exception {
112
113          File file = JFileDataStoreChooser.showOpenFile("shp", null);
114          if (file == null) {
115              return;
116          }
117
118          FileDataStore store = FileDataStoreFinder.getDataStore(file);
119          featureSource = store.getFeatureSource();
120          MapContext map = new DefaultMapContext();
121          Style style = createDefaultStyle();
122          map.addLayer(featureSource, style);
123
124          JMapFrame mapFrame = new JMapFrame(map);
125
126          mapFrame.enableToolBar(true);
127          mapFrame.enableStatusBar(true);
128
129          mapFrame.setSize(600, 600);
130          mapFrame.setVisible(true);
131      }
```

Quadro 1 - Criação de um *JMapFrame*
Fonte: Autoria própria.

Como pode ser observado no Quadro 1 entre as linhas 124 e 130, é simples a criação de um *JMapFrame*, inclusive incluindo controles que respondem a todas as camadas apresentadas, isso pode ser observado na linha 126 onde o método "*enableToolBar*" inclui automaticamente controles que permitem aumento e diminuição de *zoom*, arrastar as camadas por evento do mouse, recuperar informações a respeito da camada apresentada e voltar a camada para seu tamanho original. Pode ser observado ainda na Quadro 1 que o objeto contendo a camada, o *MapContext*, pode ser passado diretamente no construtor da classe *JMapFrame*, que irá interpretá-lo como o componente que contém as camadas que devem ser apresentadas. A inclusão de um *MapContext* em um *JMapFrame* também pode ser feita através do método "*setMapContext*". A Figura 8 apresenta o resultado da criação da interface. O objeto *MapContext* como também todos os outros que o compõem, serão abordados de maneira mais profunda no decorrer deste trabalho.

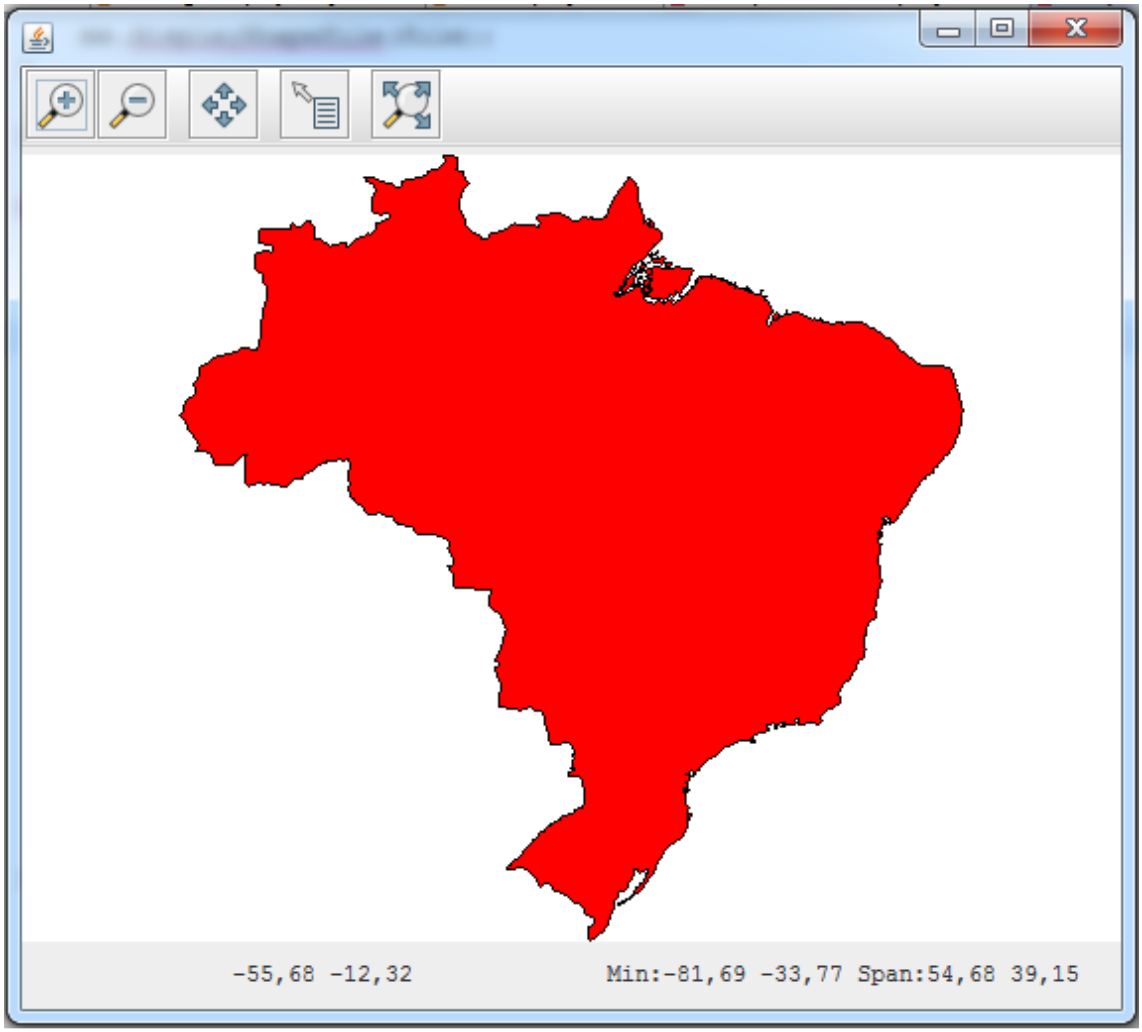

Figura 8 - Interface criada pela classe *JMapFrame*
Fonte: Autoria Própria.

Pode ser observado na Figura 8 que o mapa apresentado ocupa toda a janela criada, não deixando espaço para que sejam inseridos outros componentes dentro dela. Isso ocorre devido a fato do objeto *MapContext* ter sido passado para a classe *JMapFrame* através de seu construtor, ou mesmo quando utilizado o método *setMapContext* para isso. Assim o objeto tela principal irá interpretar o *MapContext* como o único componente que a interface irá apresentar. Uma alternativa para isso, é a utilização de outra classe fornecida por Geotools a classe *JMapPane*, que também deriva de uma classe do pacote "*javax.swing*",.

*JMapPane* fornece um painel[16] que pode ser posicionado dentro de um *JMapFrame* podendo assim possibilitar que seja utilizado qualquer layout[17] suportado pela classe *JFrame*.

[16] Painel nesse caso pode ser definido como um quadro, contendo elementos visuais ao usuário final, que pode ser posicionado em qualquer parte da tela principal.

Em um *JMapPane* podem ser incluídas camadas através do método "*setMapContext*" que também é implementado por essa classe.

Para a criação da classe principal da aplicação de exemplo desse trabalho, foi optado por utilizar a classe *JFrame*, sendo que os mapas serão apresentados dentro de um *JMapPane*, torna-se dispensável o uso da classe *JMapFrame*, sendo assim por opção, foi tomada a decisão pela classe *JFrame*.

A classe principal da aplicação é onde estarão os acesso para as funcionalidades da interface exemplo implementada, nela também estarão os comandos para manipulação das camadas carregadas e o mapa propriamente dito que será formado pelas camadas. Será apresentado na tela principal o componente *MapLayerTable*, encontrado no pacote "*org.geotools.swing*" da API Geotools, esse componente é utilizado para controle de estilos das camadas carregadas na aplicação, e também permite a manipulação das ordens de apresentação e a visibilidade.

A classe principal é formada por um *JSplitPane*[18], que divide a tela em dois lados. O lado *Oeste* é onde ficaram posicionados o componente *MapLayerTable* e uma árvore trazendo informações a respeito da camada carregada (mapa que está sendo apresentado), esse lado é dividido por outro *JSplitPane*, formando dentro dele os painéis *Norte* e *Sul*. Sendo que *MapLayerTable* ocupa o painel *Norte* e o painel para apresentação dos dados o painel *Sul*. O painel *Leste* da tela principal é onde será incluído o *JMapPane* que estará incumbido da apresentação dos mapas.

Utilizando o método "*setJMenuBar*" da própria classe *JFrame*, é carregado o menu que dá acesso as funcionalidades. A Figura 9 demonstra a tela principal.

---

[17] Layout são formas de apresentação de componentes gráficos em uma tela.

[18] JSplitPane oferece um componente que pode ser redimensionado facilitando o trabalho com dois componentes visuais na mesma tela.

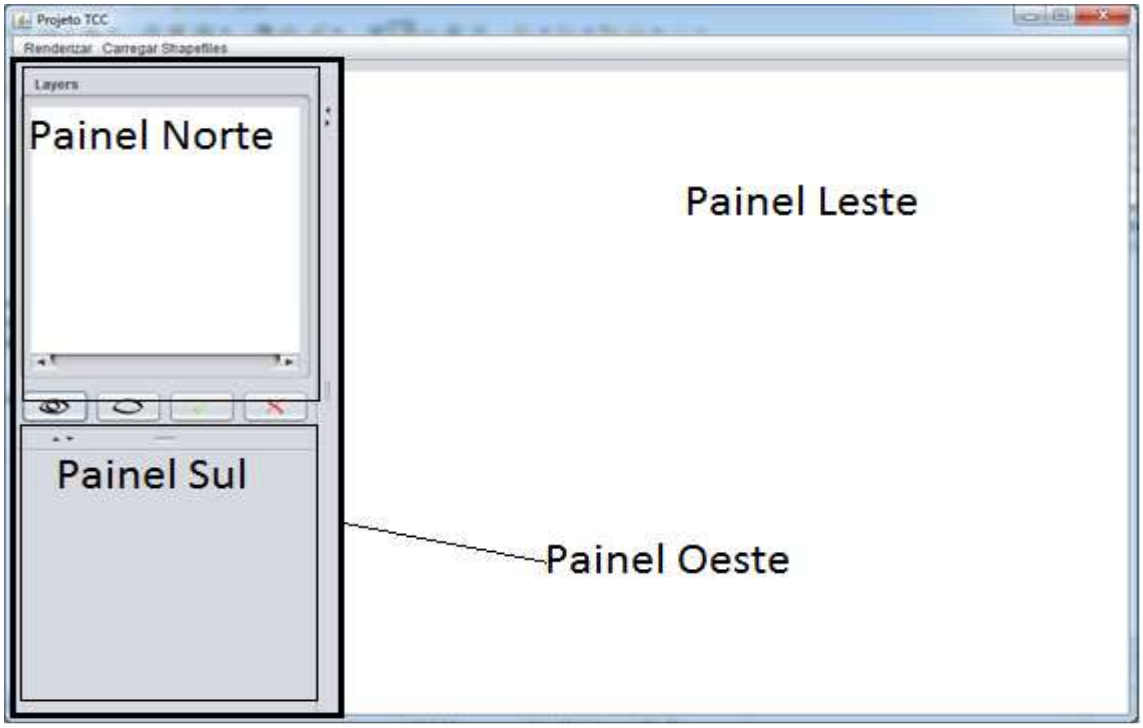

Figura 9 - Tela Principal
Fonte: Autoria própria.

### 8.1.1 Menu

O menu é criado através da classe "*MenuBar*_Principal" que utiliza funcionalidades da classe *JMenuBar* do pacote "*javax.swing*" através do principio conhecido como Herança[19]. Também utilizando o principio de Herança são criados para o menu principal, os itens de menu, esses lançando mão dos métodos herdados da classe *JMenuItem*, pertencentes ao mesmo pacote da classe *JMenu*. Os eventos disparados pelos itens do menu são manipulados através do padrão de projeto *Command*, na classe "*Handler*_Menu".

### *8.1.2 MapLayerTable*

*Na parte no*rte do painel *Oeste*, encontrasse um componente muito interessante da API Geotools, o *MapLayerTable* que facilita e bastante o trabalhos de criação de estilos, visibilidade, e posicionamento das camadas. Em uma aplicação SIG, para formar um objeto

[19] Herança consiste no fato de uma classe qualquer possuir atributos e métodos de outra, essa é tida como superclasse da classe anterior

de estudo (mapa) são necessárias várias camadas isso faz com que o sentido de posicionamento dessas camadas para visualização se torne importante, sendo que uma camada pode esconder outra embaixo dela. O componente *MapLayerTable* através de eventos *Drag-And-Drop*[20], permite uma fácil manipulação das camadas permitindo visualizá-las na ordem desejada pelo usuário. Esse componente também é responsável por controlar a visibilidade, podendo tornar camadas específicas visíveis ou não. Além das funcionalidades apresentadas, o componente apresenta outros três eventos muito úteis, o fechamento da camada o que a exclui da lista de visualização, a mudança no nome que é apresentado e a manipulação da forma como a camada é apresentada.

O objeto que irá controlar as camadas é criado pelo construtor da classe *MapLayerTable*. Na seqüência ele é adicionado ao *JMapPane* que é onde irão ser apresentadas as camadas que ele irá processar. O Quadro 2 apresenta o método que cria o componente de controle das camadas e a Figura 10 apresenta o componente com duas camadas carregadas.

```
private MapLayerTable getMapLayerTable() {
    if(map == null){
        return null;
    }else{
        layerTable = new MapLayerTable();
        mapPane.setMapLayerTable(layerTable);

        return layerTable;
    }
}
```

Quadro 2 - Método utilizado para criar o componente *MapLayerTable*
Fonte: Autoria Própria.

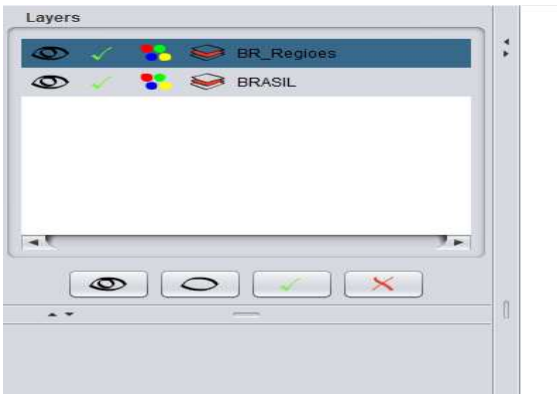


Figura 10 - Componente *JMapLayerTable*
Fonte: Autoria própria.

[20] Drag-And-Drop é a ação de clicar em um objeto virtual e deslocá-lo a uma posição diferente ou sobre outro objeto virtual

## 8.2 CARREGAMENTO DAS CAMADAS

Na aplicação exemplo dois tipos de dados podem ser carregados, vindos de um arquivo *shapefile* e dados buscados em SGDBs. Para isso precisam de interfaces distintas, sendo que precisam de configurações diferentes para a recuperação desses dados.

### 8.2.1 Carregamento de arquivos *shapefile*

*Shapefiles*, por se tratarem de arquivos armazenados em disco por meio de um sistema de diretórios, foi necessária uma interface de acesso aos diretórios da maquina local, para que o arquivo possa ser carregado. A API Geotools oferece um componente para isso, através da classe *JFileDataStoreChooser*. Para o acesso a essa funcionalidade foi preciso a criação de um item no menu principal, o "*JMenuItem_CarregarShapefile*". O Quadro 3 apresenta o método "*execute*", do item de menu, onde pode ser observada a utilização da classe *JFileDataStoreChooser*, através de seu método "*showOpenFile*", para o carregamento de um arquivo.

```
34    @Override
35    public void execute() {
36        File f = new File("C:\\Users\\Cko-Not\\Documents\\ROOT\\TCC\\Shapes");
37        File file = JFileDataStoreChooser.showOpenFile("shp", f, principal);
38        if (file != null) {
39            facade = new FacadeServicos(principal);
40            facade.apresentarShapeFile(file);
41        }
42    }
```

Quadro 3 - Método *execute* demonstrando a utilização da classe *JFileDataStore*
Fonte: Autoria própria.

No Quadro 3 pode ser observada na linha 37 que é criado o componente através do método "*showOpenFile*", esse método possui três parâmetros, o primeiro uma *String*[21] para trabalhar como um filtro, só permitindo ao usuário a visualização de arquivos com a extensão "shp", extensão dos arquivos *shapefile*, o segundo é um objeto *File* do pacote "*java.io*", que é utilizado para definir o diretório raiz para a busca dos arquivos, o terceiro e último parâmetro é o componente ao qual o janela que será aberta estará ligada, para que seja fornecida a ela

[21] String é uma classe do pacote Java.lang que representa uma seqüência de caracteres.

uma referência de apresentação. A Figura 11 demonstra a tela de carregamento de arquivos *shapefile*.

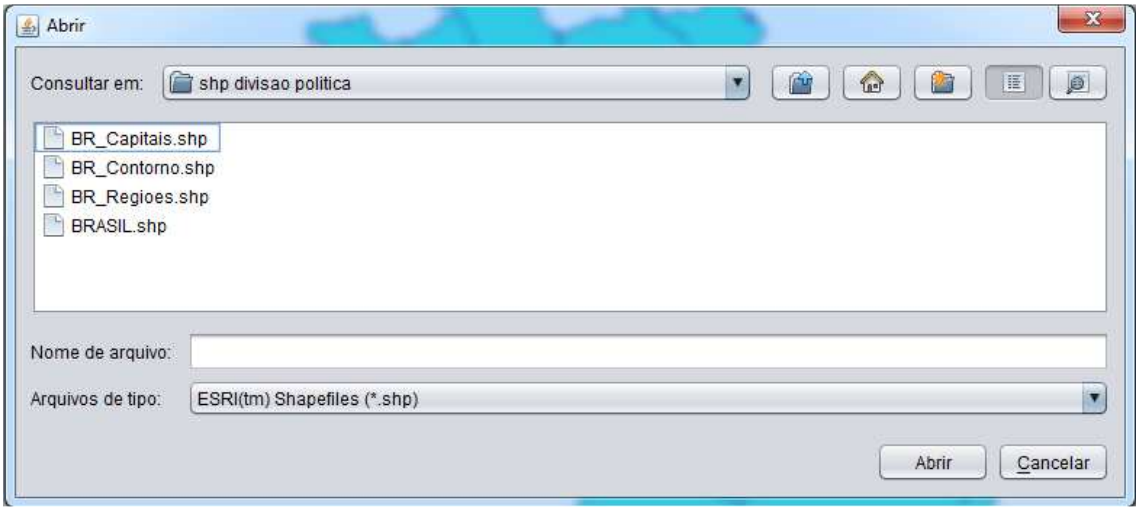


Figura 11 - Tela de carregamento de *shapefiles*
Fonte: Autoria própria.

Após ser carregado o arquivo é enviado para o método responsável pela criação da camada. Isso se dá através da utilização do padrão de projeto *Facade* como pode ser observado ainda no Quadro 3, linha 40. Nesse método é aonde os componentes da API Geotools vão trabalhar com os dados do arquivo com a intenção de transformá-lo sem um componente apresentável.

A primeira coisa a ser feita é extrair do arquivo carregado, os dados que formam a camada. Isso é feito com a utilização de uma *interface* da API Geotools, *DataStore*, que funciona como uma espécie de repositório de dados, tanto dados de arquivos armazenados em disco, como vindos de SGDBs. Essa interface é implementada por algumas outras *interfaces* de Geotools, uma delas, a que foi utilizada nesse estudo é a *FileDataStore*. Um objeto da *interface FildeDataStore*, pode ser criado através da classe *FileDataStoreFinder*, pelo seu método "*getDataStore*", como parâmetro para esse método é passado o arquivo de onde serão recuperados os dados e disponibilizados no repositório criado. Isso fará com que os dados contidos no arquivo agora possam ser acessados.

Esse acesso é feito através de outra *interface* da API Geotools, a *FeatureSource* que é uma implementação da *interface Feature*, citada no capitulo 8. Essa também é uma *interface* implementada por outras *interfaces*, que definem especificações para o trabalho que se deseja realizar com os dados contidos no *DataStore*. *SimpleFeatureSource* e *SimpleFeatureStore* são implementações da interface *FeatureSource*. Ambas são *interfaces* que permitem acesso ao conteúdo do *DataStore*, porém as duas são utilizadas de maneira distinta. A primeira é

utilizada simplesmente para recuperar informações contidas no arquivo para que possa ser apresentado, para isso dispõe de muitos métodos que permitem o acesso a esse conteúdo. A segunda vai mais além, não fornece apenas o acesso, mas também permite que seja modificado o conteúdo extraído. Os dados de um *DataStore* são recuperados pelo método "*getFeatureSource*".

Para que uma camada seja realmente criada, é necessária a utilização de mais alguns recursos oferecidos pela API Geotools. Mais uma coisa que se precisa saber sobre camadas em Geotools é que não são formadas apenas por dados extraídos de algum lugar, necessitam de "estilos". Um estilo pode ter como definição, a forma como a camada será apresentada e são definidos pela *interface Style* do pacote "*org.geotools.styling*".

Uma maneira fácil de definir estilos é através de uma classe de Geotools que fornece para o trabalho com *Styles* a classe *JSimpleStyleDialog*, também do pacote "*org.geotools.swing.styling*", ela fornece uma interface para as várias configurações que podem ser aplicadas sobre uma camada. O método estático "*showDialog*" cria uma tela (interface gráfica) que tem como retorno um objeto *Style* definido e pronto para ser aplicado a uma camada, podendo ser configurados atributos referente a cor e espessura da linha de contorno da geometria, cor e transparência do contorno, tamanho e símbolo de geometrias do tipo ponto e definir valores que serão apresentados junto a camadas, contidos no arquivo, ou na tabela que gerou a geometria.

A Figura 12 demonstra a interface criada pela classe *JSimpleStyleDialog* através do método *showDialog*.

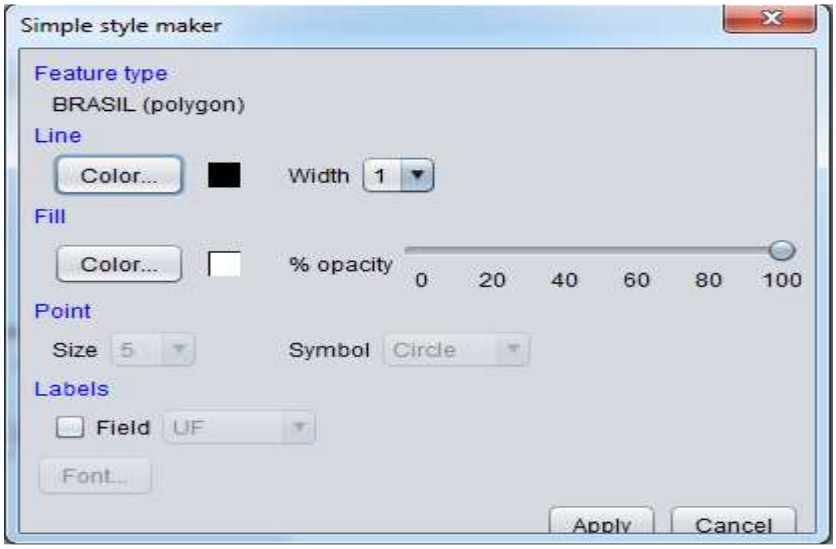


Figura 12 - Tela criada pela classe *JSimpleStyleDialog*
Fonte: Autoria própria.

O desenvolvedor pode também optar por não usas a classe oferecida pela API para criação de estilos, para isso devera trabalhar com a interface *Style* além de precisar do apoio de outras classes e *interfaces* oferecidas por Geotools. Ao se trabalhar com a criação de estilos, surge a necessidade do desenvolvedor saber de que tipo de geometria se trata. Para o carregamento do *shapefile* nesse estudo foram utilizados dois tipos de geometria, Polígono e Ponto, sendo que os dados utilizados fornecem estes dois tipos de geometria. Os dois possuem características semelhantes, ambos são desenhados com uma cor e um contorno, porém um Ponto necessita também de outras características para o *Style* que irá representá-lo como uma imagem para formar seu símbolo, e esse podendo ter um tamanho variável, por exemplo.

Geotools utiliza a interface *Rule* para definir regras baseadas no tipo de geometria que será apresentada. Objetos de *Rule*, são criados através método "*createRule*" da interface *StyleFactory*. *StyleFactory*, é uma interface muito importante, pois é através dela que todo o estilo da camada será criado. Quando é criada uma regra são necessário que sejam definidos alguns atributos para o *StyleFactory*. Primeiramente é preciso que se saiba qual o tipo de geometria a camada representa para daí poder ser adicionados atributos específicos. Isso em Geotools pode ser conseguido de maneira simples com a utilização da *interface SimpleFeatureType* que define qual o tipo de geometria o objeto *FeatureSource* representa, o objeto dessa *interface* pode ser conseguido através da utilização de apenas um método da classe *FeatureSource*, o método "*getSchema*". Através do objeto dessa classe é possível recuperar um segundo objeto do tipo *Class* do pacote "*java.lang*" e com uma comparação simples definir de que tipo de geometria se trata. Isso pode ser observado no Quadro 4.

```
182         SimpleFeatureType schema = (SimpleFeatureType)featureSource.getSchema();
183         Class geomType = schema.getGeometryDescriptor().getType().getBinding();
184
185         if (Polygon.class.isAssignableFrom(geomType)
186                 || MultiPolygon.class.isAssignableFrom(geomType)) {
187             return createPolygonStyle();
188
189         } else if (LineString.class.isAssignableFrom(geomType)
190                 || MultiLineString.class.isAssignableFrom(geomType)) {
191             return createLineStyle();
192
193         } else {
194             return createPointStyle();
195         }
196     }
```

Quadro 4 - Utilização da interface *SimpleFeatureType* para definir tipos de geometria
Fonte: Autoria própria.

Definido o tipo de geometria agora podem ser atribuídas regras especificas para elas. Pode ser observado no Quadro 4 que para cada tipo de geometria é chamado um método

específico onde são definidos os estilos e as regras. Para a definição de regras são utilizados filtros definidos pela interface *FilterFactory*. Filtros em Geotools são utilizados para recuperar dados com características específicas, como por exemplo, “desenhar todos os pontos de uma tabela que contenha o valor de determinado atributo superior a 10 (dez)”. Os filtros podem ser utilizados também de maneira mais complexa através de seus muitos métodos. Pode ser ter como exemplos da utilização de filtros, restrições por intervalos de valores, junção entre geometrias e o cruzamento de geometrias. Porém pode e são utilizados de maneira mais simples, pelo objeto *StyleFactory* para a criação do contorno e do preenchimento da geometria. Nesse caso os filtros são usados para definir as cores e transparência dos objetos apresentados. Para definição das características de contorno e preenchimento, Geotools fornece as interfaces *Stroke* e *Fill*, respectivamente.

Esses dois objetos são criados através do objeto *StyleFactory*. Para a criação do *Fill* é utilizado o método “*createFill*”, esse método possui dois parâmetros, ambos passados através de objetos da *interface FilterFactory*, o primeiro define a cor do preenchimento o segundo a transparência. O mesmo caso pode ser observado para a criação do contorno, *Stroke*, porém esse criado através do método “*createStroke*”, que possui uma pequena diferença no seu segundo parâmetro, sendo que se trata de uma linha que delimita a área da figura desenhada, o segundo parâmetro é utilizado para definir a espessura dessa linha. Além do contorno e do preenchimento, para uma geometria do tipo Ponto devem ser definidos outro atributo o símbolo, ou seja, a imagem que irá representar esse ponto. Essas definições também são feitas através do *StyleFactory*. O Quadro 5 demonstra a definição de atributos para uma geometria Ponto.

```
143     stroke = sf.createStroke(ff.literal(Color.BLACK), ff.literal(LINE_WIDTH));
144     fill = sf.createFill(ff.literal(Color.YELLOW), ff.literal(OPACITY));
145     Mark mark = sf.getCircleMark();
146     mark.setFill(fill);
147     mark.setStroke(stroke);
148     Graphic graphic = sf.createDefaultGraphic();
149     graphic.graphicalSymbols().clear();
150     graphic.graphicalSymbols().add(mark);
151     graphic.setSize(ff.literal(POINT_SIZE));
152     symbolizer = sf.createPointSymbolizer(graphic, null);
153
```

Quadro 5 - Definição dos atributos de *Style* para uma geometria Ponto
Fonte: Autoria Própria.

No Quadro 5 as linhas 143 e 144 são definidos o contorno e o preenchimento respectivamente, nas linhas seguintes 145 a 147, pode se observar a definição de símbolo que

irá representar a geometria, nesse caso foi utilizado um símbolo padrão da *StyleFactory*, o método "*getCircleMark*" define que será apresenta o ponto como um circulo. Por último na linha 152, aparece o objeto "*Symbolizer*" que será adicionado ao objeto *Rule* para que a regra fique formada e possa ser aplicada no *Style*. A interface *Symbolizer* define os atributos para uma geometria específica. *PointSymbolizer* e *PolygonSimbolizer* são interfaces que implementam *Symbolizer* definindo atributos específicos para o *Style* de cada uma das geometrias. Um objeto de *Symbolizer* é criado pelo *StyleFactory* através de métodos específicos, são eles, "*createPointSymbolizer*" e "*createPolygonSymbolizer*".

Definidos todos os atributos para o StyleFactory a regra pode ser criada. Uma regra não é adicionada diretamente em um estilo, isso é feito através de um objeto da interface *FeatureTypeStyle*, que define um tipo de estilo para o objeto *Style*, nesse objeto são armazenadas as regras, e ele é adicionado no objeto *Style* que será utilizado na camada. Estando os *StyleFactory* criado e as regras adicionadas no *FeatureTypeStyle* é hora, mais uma vez, de utilizar o *StyleFactory* para a criação do *Style* através de seu método "*createStyle*" e por ultimo adicionar as regras que estão contidas no *FeatureTypeStyle*. O Quadro 6 apresenta a criação do objeto *Style*.

```
74          Rule rule = createRule(featureSource);
75
76          FeatureTypeStyle fts = sf.createFeatureTypeStyle();
77
78          fts.rules().add(rule);
79
80          Style style = sf.createStyle();
81          style.featureTypeStyles().add(fts);
82
```

Quadro 6 - Criação do objeto *Style*
Fonte: Autoria Própria.

Observando no Quadro 6, a linha 74 chama o método responsável por criar a regra passando o objeto *FeatureSource*. Esse método irá definir o tipo da geometria que esse objeto representa e aplicar ao objeto *Rule* as regras especificas. Seguindo na linha 76 é criado o *FeatureTypeStyle* que irá compor o *Style*, na linha 78 é adicionado ao objeto recém criado a regra e nas linhas 80 e 81 é criado o *Style* e adicionado a ele o *FeatureTypeStyle*.

O módulo *render-gt* define uma estrutura de dados para representar o conteúdo de uma camada. A classe *MapContext* desse módulo é utilizada para capturar o conteúdo de uma camada para a apresentação. O *MapContext* é o objeto que será incluído na interface gráfica através do *JMapPane*, como foi dito um mapa pode ser formado por diversas camadas. Através de objetos *MapLayer* é que essas camadas vão ser adicionados no *MapContext*.

Para a criação de uma camada é criado um objeto, *MapLayer* utilizando o construtor da classe *DefaultMapLayer*. É nesse construtor que são passados o *FeatureSource* representando os dados da camada e o *Style* criado para a representação gráfica. Esse é o momento em que os dados extraídos do *DataStore* são transformados em camada. O Quadro 7 demonstra a criação de um objeto *MapLayer*.

```
MapLayer map = new DefaultMapLayer(featureSource,
        criaStilo(featureSource));
```

Quadro 7 - Criação do objeto *MapLayer*
Fonte: Autoria Própria.

Quando se tem a camada formada com todas suas regras e estilos, o passo seguinte é apresentá-la ao usuário, como citado no parágrafo anterior isso é feito através de um objeto *MapContext* ligado ao *JMapPane*. Mais especificamente isso é feito pelo método "*addLayer*" da classe *MapContext*. Isso feito o mapa é apresentado para o usuário. Aqui aparece outra facilidade oferecida pela API, sendo que o objeto *JMapPane* já está ligado ao *MapLayerTable*, e o objeto *MapContext* ligado ao *JMapPane*, nada mais é preciso ser feito para que as funcionalidades de controle oferecidas por *MapLayerTable* estejam disponíveis para a camada carregada.

### 8.2.2 Aplicação de funcionalidades as camadas apresentadas

Com a camada representada na interface, pode ser feito e geralmente é, a disponibilização de componentes para interação com as camadas, como aumento e diminuição de *zoom* por exemplo. A API Geotools fornece um conjunto de funcionalidades pré-implementadas tornando fácil a inclusão delas nas camadas. Esse conjunto é formado por cinco funcionalidades, implementadas através de classes do pacote "*org.geotools.swing.action*":

- *ZoomInAction*: implementa funcionalidades de aumento de *zoom*;
- *ZoomOutAction*: implementa funcionalidades de diminuição de *zoom*;
- *ResetAction*: retorna o mapa ao seu tamanho original;
- *InfoActio*: recupera informações a respeito do mapa apresentado; e
- *PanAction*: implementa o evento de arrastar e soltar para que se possa manipular o mapa dentro da área onde está desenhado.

Essas classes descendem de *MapAction* do mesmo pacote delas, essa por sua vez descende de *AbstractAction* do pacote "*javax.swing*", isso torna simples a utilização desses componentes bastando que uma instância deles seja inserida no construtor de um *JButton*[22] para que suas funcionalidades sejam aplicadas a esse botão. A ligação entre esses componentes e os mapas são feitos através do *JMapPane* que é passado no construtor das classes que irão implementar as funcionalidades. Por último, basta adicionar os botões na interface da maneira que desejar. No caso da interface de exemplo desse estudo foi criada uma barra de botões através de um objeto *JToolBar* de "*javax.swing*" e adicionado logo abaixo do menu principal. O Quadro 8 apresenta a criação das funcionalidades através dos *JButtons*.

```
148         buttonZoomIn = new JButton(new ZoomInAction(mapPane));
149         buttonZoomOut = new JButton(new ZoomOutAction(mapPane));
150         buttonResetMap = new JButton(new ResetAction(mapPane));
151         buttonInfoActionMap = new JButton(new InfoAction(mapPane));
152         buttonPanActionMap = new JButton(new PanAction(mapPane));
```

Quadro 8 - Criação dos componentes para as camadas
Fonte: Autoria própria.

A Figura 13 apresenta a interface exemplo criada, com duas camadas carregadas, uma de tipo Polígono, que representa o território da Republica Federativa do Brasil divida por estados, e a outra do tipo Ponto, que representa as capitais estaduais.

[22] JButton é um componente da API Swing que representa botões para ações na interface com o usuário.

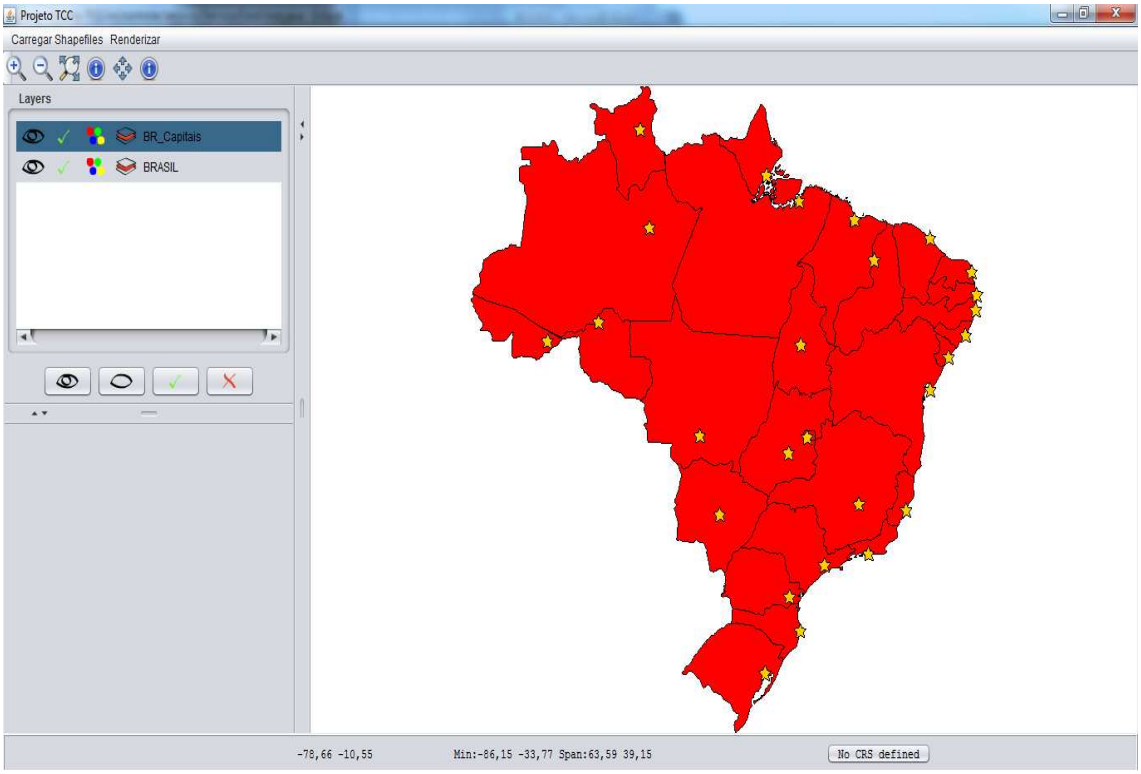

Figura 13 - Tela principal com duas camadas carregadas
Fonte: Autoria própria.

## 8.3 CRIANDO UMA CONEXÃO COM O SGDB POSTGRESQL

Conexões com o SGDBs também são criadas pela interface *DataStore*. Sendo *DataStore* um repositório de dados, os dados retornados do SGDB também serão adicionados a ele para que os objetos *FeatureSource* possam acessá-los e transformá-los em camadas. O que permite a criação de um *FeatureSource* nesse caso é o método "*getFeatureSource*", que para esse caso recebe como parâmetros uma *String*, que define em que tabela os dados estão contidos.

Um *DataStore* que irá conectar-se a um SGDB precisa ter informações para que realize essas conexões, informações como, nome do banco de dados onde estão as tabelas a serem recuperadas, a porta de conexão para esse banco, a máquina onde o banco está, sendo que este pode estar em uma máquina qualquer da rede, entre outros parâmetros que precisam ser adicionados. A classe *DataStoreFinder* mostrada anteriormente no carregamento de arquivos *shapefiles*, possui um método "*getDataStore*", que recebe como parâmetro, um objeto *Map* do pacote "*java.util*". O objeto *Map* deve conter todos os parâmetros necessários para a criação do *DataStore*. O retorno do método *getDataStore* é um o *DataStore* pronto para o acesso aos dados do SGDB. Isso deixa o uso dessas conexões muito dinâmicas, sendo que

podem ser criados vários objetos de conexão para diferentes bancos de dados, assim uma aplicação pode trabalhar com diversas bases de dados.

Para se obter os dados que formarão o *Map* de parâmetros de configuração podem ser utilizadas várias formas, como por exemplo, recuperar os dados de arquivos de configuração, que podem ser arquivos com extensões TXT[23], XML[24] e etc., ou mesmo de tabelas contidas em uma base de dados. O Quadro 9 apresenta a criação de um *DataStore* fazendo uso da leitura de arquivo de configuração.

```
35    public static DataStore createDataStore(){
36        File file = new File("src/conexao/modelo/connection/db.properties");
37        Properties props = new Properties();
38        FileInputStream fis = null;
39        try {
40            fis = new FileInputStream(file);
41            props.load(fis);
42            fis.close();
43        }
44        catch (IOException ex) {
45            System.out.println(ex.getMessage());
46            ex.printStackTrace();
47        }
48        Map<String, Object> parametros = new HashMap<String, Object>();
49        parametros.put(JDBCDataStoreFactory.DBTYPE.key, props.getProperty("dbtype"));
50        parametros.put(JDBCDataStoreFactory.HOST.key, props.getProperty("location"));
51        parametros.put(JDBCDataStoreFactory.PORT.key, new Integer(5432));
52        parametros.put(JDBCDataStoreFactory.DATABASE.key, props.getProperty("db"));
53        parametros.put(JDBCDataStoreFactory.USER.key,props.getProperty("username"));
54        parametros.put(JDBCDataStoreFactory.PASSWD.key,  props.getProperty("password"));
55        try {
56            dataStore = DataStoreFinder.getDataStore(parametros);
57        } catch (IOException e) {
58            JOptionPane.showMessageDialog(null, "Erro");
59            e.printStackTrace();
60        }
61        if (dataStore == null) {
62            JOptionPane.showMessageDialog(null, "Erro");
63            System.exit(0);
64        }
65        return dataStore;
66    }
```

Quadro 9 - Criação de um *DataStore* utilizando parâmetros de um arquivo de configuração
Fonte: Autoria Própria.

Geotools, porém contém a classe *JDataStoreWizard*, que fornece uma interface para a entrada desses parâmetros. Através do método estático "*showModalDialog*" é criada uma janela modal onde o usuário pode inserir os parâmetros de configuração. Primeiramente é exibida uma tela onde o usuário pode selecionar o tipo do SGDB para o qual deseja criar a conexão, em seguida é apresentada a tela com os campos para os parâmetros de configuração para aquele SGDB especifico. A Figura 14 apresenta a interface criada pelo método

[23] Arquivos de texto

[24] XML (*Extensible Markup Language*) é uma recomendação da W3C para gerar linguagens de marcação para necessidades especiais.

*showModalDialog*, para a configuração de conexão com a extensão espacial PostGIS do SGDB PostgreSQL.

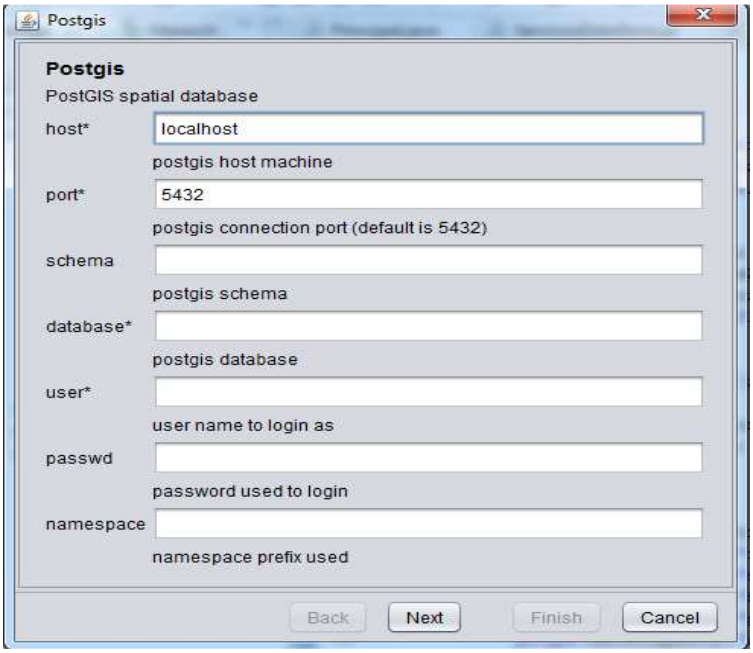


Figura 14 - Tela fornecida pela classe *JDataStoreWizard* para entrada de parâmetros do *DataStore*
Fonte: Autoria própria.

Com a conexão através do *DataStore* criada basta os dados contidos nas tabelas da base de dados serem recuperados. Isso é feito pelo método *getFeatureSource*, que nesse caso necessita de um parâmetro *String* que fornece o nome da tabela a ser pesquisada. Esse método fará com que os dados da tabela sejam buscados e armazenados no *DataStore* para daí serem manipulados por objetos *FeatureSource*.

# 9 CONSIDERAÇÕES FINAIS

Através desse estudo foi possível verificar que a API Geotools é uma interessante ferramenta para desenvolvimento de Sistemas de Informação Geográfica, pois fornece uma infinidade de Classes, Interfaces e métodos responsáveis para o trabalho com dados espaciais. Geotools mostrou-se de uso fácil e com muitas classes pré-implementadas, o que retira do usuário a responsabilidade do desenvolvimento de alguns componentes chave no trabalho com dados espaciais. O fato do desenvolvimento da API separar o modelo de visão do modelo de negócios foi um ponto visto como positivo, pois permite o baixo acoplamento entre os módulos desenvolvidos facilitando a divisão em camadas das aplicações que utilizam a API. O projeto possui uma boa documentação disponível na página oficial e na internet que se mostrou suficiente para o aprendizado de Geotools. O pouco material disponível em português foi uma pequena dificuldade encontrada e facilmente contornada.

Conclui-se que os Sistemas de Informações Geográficas estão ganhando cada dia mais importância no mundo, devido a sua versatilidade de representação computacional de dados geográficos e atributos referentes aos mesmos, fazendo com o sistema tenha um raio de alcance maior disponibilizando uma visão mais completa do serviço que lhe é proposto.

O estudo de caso mostrou como os dados geográficos podem ser representados em uma interface gráfica *JavaSE*, além de demonstrar como se aplicam e funcionam algumas das funcionalidades oferecidas por Geotools, o que veio a comprovar que suas operações suprem o propósito ao qual é designada de maneira produtiva e transparente.

## 10 REFERÊNCIAS BIBLIOGRÁFICAS


Geotools. **Documentation**. Disponível em: < http://docs.geotools.org/> Acesso dia: 20/05/2011

Geotools. **Wiki**. Disponível em: < http://docs.codehaus.org/display/GEOTOOLS/Home> Acesso dia: 25/05/2011

Geotools. **About**. Disponível em: < http://docs.codehaus.org/display/GEOTOOLS/Home> Acesso dia: 25/05/2011

ANTUNES, Alzir Felippe Buffara. **Iniciando em Geoprocessamento**. Disponível em: <http://people.ufpr.br/~felipe/sig.pdf> Acesso dia: 10/04/2011

BORGES, Karla A. V. JÚNIOR, Clodoveu A. Davis. LEANDER, Alberto H. F. **Modelagem Conceitual de Dados Geográfico**. Disponível em: <http://www.dpi.inpe.br/livros/bdados/cap3.pdf> Acesso dia: 10/04/2011

BRANDALIZE, Maria Cecília Bonato. **Apostila Topografia**. Disponível em: <http://www2.uefs.br/geotec/topografia/apostilas/topografia(1).htm> Acesso dia: 10/04/2011

CÂMARA, Gilberto, QUEROZ, Gilberto Ribeiro de. **Arquitetura de Sistemas de Informação Geográfica**. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/introd/cap3-arquitetura.pdf> Acesso dia: 20/04/2011

CÂMARA, Gilberto. Davis, Clodoveu. **Introdução**. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/introd/cap1-introducao.pdf> Acesso dia: 20/4/2011
CÂMARA, Gilberto. FERREIRA, Karine Reis. QUEIROZ, Gilberto Ribeiro de. **Arquitetura de Banco de Dados Geográfico**. Disponível em: <http://mtc-

m12.sid.inpe.br/col/sid.inpe.br/sergio/2004/10.07.15.53/doc/cap2.pdf> Acesso dia: 13/05/2011

CÂMARA, Gilberto. **Representação Computacional de Dados Geográfico**. Disponível em: <http://www.dpi.inpe.br/livros/bdados/cap1.pdf> Acesso dia: 13/05/2011

CAMPOS, Samuel Rodrigues de Sales. MARTINHAGO, Adriana Zanella. CARVALHO, Luis Marcelo Tavares de. SCOLFORO, José Roberto. OLIVEIRA, Antônio Donizete de. VEIGA, Ruben Delly. LIMA, Renato Ribeiro. **Banco de Dados**. Disponível em: <http://www.redeapasul.com.br/publicacoes/banco_dados_zee.pdf> Acesso dia: 04/04/2011

CASANOVA, Marco Antonio; CÂMARA, Gilberto; JR, Clodoveu A. Davis; VINHAS, Lúbia; QUEIROZ, Gilberto Ribeiro. **Bancos de Dados Geográficos**. Curitiba: MundoGEO, 2005.

COSTA, Helen Camargo. SILVA, Marcos Vinícius Alexandre. **Projeções Cartográficas**. Disponível em: <http://www.lapig.iesa.ufg.br/lapig/cursos_online/gvsig/projees_cartogrficas.html> Acesso dia: 03/05/2011

D'ALGE, Julio César Lima. **Cartografia Para o Geoprocessamento**. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/introd/cap6-cartografia.pdf> Acesso dia: 16/02/2011

DEMARQUI, Edgar Nogueira. **Características de um SIG**. Disponível em: <http://www.unemat-nfet.br/prof/foto_p_downloads/sig_aula2.pdf> Acesso dia: 14/03/2011

DEMARQUI, Edgar **Nogueira.Gerência de Dados em SIG**. Disponível em: <http://www.unemat-net.br/prof/foto_p_downloads/sig_aula7.pdf> Acesso dia: 14/03/2010

ESRI. **ESRI Sshapefile Technical Description**. Disponível em: <http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf> Acesso dia: 16/04/2011

FRANCISCO, Cristiane. **Estudo Dirigido em SIG**. Disponível em:<http://www.professores.uff.br/cristiane/Estudodirigido/Cartografia.htm> Acesso dia: 15/02/2011

HARA, Lauro Tsutomu. **Técnicas de Apresentação de Dados em Geoprocessamento**. Disponível em: <http://www.obt.inpe.br/pgsere/Hara-L-T-1997/cap2.pdf> Acesso dia: 02/02/2011

IRRIGART. **Banco de Dados**. Disponível em: <http://www.agenciadeaguapcj.org.br/download/RS-04-06_Capitulo-10.pdf/> Acesso dia: 13/04/2011

JUNIOR, Clodoveu A. Davis. BORGES, Karla A. V. SOUZA, Ligiane Alves de. CASANOVA, Marco Antônio. JÚNIOR, Paulo de Oliveira Lima**. O Open Geospatial Consortium**. Disponível em: <http://www.dpi.inpe.br/livros/bdados/cap11.pdf> Acesso dia: 14/04/2011

JÚNIOR, Clodoveu A. Davis. LEANDER, Alberto H. F. **Extensões ao modelo OMT-G para Produção de Esquemas Dinâmicos e de Representação**. Disponível em: <http://www.dpi.inpe.br/cursos/ser300/Referencias/davis_geoinfo.pdf> Acesso dia: 18/04/2011

MEDEIROS, Anderson Maciel Lima de. **Padrões Open Geospatial Consortium**. Disponível em: < http://blog.geoprocessamento.net/2010/03/ogc-parte1/> Acesso dia: 14/04/2011

MELO, Cléber Hostalácio; GUERRA, Marcio Azevedo de Menezes. **SGBD com Extensão Espacial e Sistema de Geoinformação: Um Casamento Perfeito**. Disponível em:< http://felipe.manauense.com.br/geomatica/artigo-comparando-sgbds-com-extensao-espacial> Acesso dia 26/03/2011.

PERCIVALL, George. **OGC Reference Model**. Disponível em: <portal.opengeospatial.org/files/?artifact_id=3836> Acesso dia: 15/04/2011

PONTES, Marco A. G**. GIS e Geoprocessamento**. Disponível em: <http://www.cedupcua.com.br/~jucemar/Apostilas_Diversas_Areas/Construcao_Civil/FACENS%20-%20Topografia%20-%20Geoprocessamento.pdf> Acesso dia: 08/02/2011

POSTGIS DOC. **PostGIS 1.5.0 Manual**. Disponível em : <http://postgis.refractions.net/docs/> Acesso dia 02/02/2011.

TIMES, Valéria. **Sistema de BD Geográfico PostGIS**. Disponível em: <http://www.cin.ufpe.br/~mcts/BDA/Geografico/> Acesso dia: 23/04/2011

UCHOA, Helton Nogueira; COUTINHO, Renata Juliana Cristal; FERREIRA, Paulo Roberto; FILHO, Luiz Carlos Teixeira; BRITO, Jorge Luís Nunes e Silva. **Análise do módulo PostGIS(OpenGIS) para armazenamento e tratamento de dados geográficos com alta performance e baixo custo**. Disponível em <http://www.opengeo.com.br/download/postgis-sbc-v13-06102005.pdf> Acesso em 02/05/2011.