UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM SOFTWARE PARA VISUALIZAÇÃO DE INFORMAÇÕES ESPACIAIS UTILIZANDO BANCO DE DADOS ORACLE COM INTERAÇÃO PELA INTERNET

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

SIDNEI BUNDE

BLUMENAU, JULHO/2002

2002/1-70

PROTÓTIPO DE UM SOFTWARE PARA VISUALIZAÇÃO DE INFORMAÇÕES ESPACIAIS UTILIZANDO BANCO DE DADOS ORACLE COM INTERAÇÃO PELA INTERNET

SIDNEI BUNDE

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Dalton Solano dos Reis - Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Dalton Solano dos Reis

_____

Prof. Alexander R. Valdameri

_____

Prof. Marcel Hugo

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Este Trabalho de Conclusão de Curso visa realizar um estudo sobre Sistemas de Informação Geográfica (SIG), mais especificamente *Automated Mapping/Facilities Management (*AM/FM*)* que permita obter como resultado final uma aplicação *Web* que utilize Java e *Applets* para realizar consultas sobre dados espaciais. Tem-se também como resultado a implementação de um protótipo que permite visualizar via *Web* informações do tipo *Raster* e vetorial, bem como operações de *Zoom In/Out* e *Pan*.

ABSTRACT

This work aims to carry a study on Geographical Information Systems (GIS), more specifically Automated Mapping/Facilities Management (AM/FM) that it allows to get as final result a Web application that uses Java and Applets to carry the query criteria on spatial data. It also had as result the implementation of Web software prototype that allows to visualize vectorial the raster information, as well as operations of *Zoom In/Out and Pan*.

1 INTRODUÇÃO

Entre as aplicações que representam informação espacial estão os sistemas de informação geográfica (SIG) e as bases de dados de imagens, onde é importante descrever o espaço, os objetos (e as suas características) e as posições por estes ocupadas ao longo do tempo (Madeira, 1999).

Uma das áreas do SIG é a Mapeamento Automatizado e Gerenciamento de Serviços (AM/FM), que segundo Jacob (1999) o AM/FM utiliza a capacidade de base de dados para armazenar informações sobre objetos mapeados e juntar estes dados as informações do mapa, mas geralmente não possuem capacidades de análise espacial ou estrutura de dados topológica.

Outro fator está em poder combinar uma parte das capacidades dos sistemas CADD (*Computer Aided Design and Drafting*, ou Projeto Assistido por Computador) para gráficos interativos, entrada e técnicas de armazenamento com a capacidade da base de dados. O objetivo dos sistemas do tipo AM/FM é converter mapas manuais e registros em base de dados digitais para seleção e processamento de ordem de trabalho. Sistemas AM/FM são grandemente utilizados em redes de utilidades ou facilidades (redes de esgoto, de água, luz, telefonia, etc.), pois possuem atributos de conectividade.

Uma das consultas mais utilizadas em aplicações de AM/FM é a determinação do menor caminho, podendo citar como exemplo a determinação de um caminho mais curto entre duas cidades num mapa de estradas. Tipicamente este exemplo é conhecido como o problema do caminho mínimo, onde para cada arco associa-se um custo gasto para percorrê-lo, e através de um algoritmo de menor caminho encontra-se a solução (Leal, 2001?). Este problema pertence a área da teoria dos grafos, o qual representa uma estrutura de dados constituída por um conjunto de vértices (ou nodos) e arestas (ou arcos), onde um par de vértices formam uma aresta e o caminho é uma seqüência de vértices (Salgados 2001).

O problema do caminho mais curto é o mais importante problema relacionado com a busca de caminhos em grafos, em vista de sua ligação direta com uma realidade encontrada a todo momento. É comum a existência de distâncias físicas associadas, como no caso das redes de utilidades ou facilidades (Netto, 1979).

Um dos pontos mais relevante na definição de aplicações AM/FM esta relacionado com a gerência de banco de dados, pois este deve permitir entre outras coisas, representar informações espaciais.

Um dos bancos de dados que atualmente possui esta capacidade é o Banco de Dados *ORACLE 8i*, que permite armazenar e fazer consultas sobre mapas no formato vetorial, conceito denominado no *ORACLE 8i* como *Spatial Data Option*. Outra característica do *ORACLE 8i,* é que este fornece ferramentas avançadas para gerenciar todos os tipos de dados em *sites* da web. O *Internet File System (iFS)* combina a potência do *ORACLE 8i* com a facilidade de uso de um sistema de arquivos, permitindo que os dados possam ser armazenados e gerenciados com mais eficiência. Os usuários finais podem acessar facilmente arquivos e pastas no *ORACLE iFS* através de vários protocolos, como HTML, FTP e IMAP4, que fornecem acesso universal aos dados (Oracle, 1999).

A *ORACLE* oferece uma plataforma Internet abrangente e de alto desempenho para comércio eletrônico e armazenamento de dados. Essa plataforma integrada inclui tudo o que é necessário para desenvolver, depurar e gerenciar aplicações para Internet. A Plataforma Internet da *ORACLE* é organizada em três partes principais: clientes baseados em navegadores para processar apresentações; servidores de aplicações para executar a lógica comercial e a lógica de apresentação de servidor aos clientes baseados em navegadores; bancos de dados para executar a lógica comercial com uso intensivo do banco de dados e dos dados do servidor (Oracle, 1999). A *ORACLE* oferece ainda várias ferramentas de desenvolvimento orientadas a *Graphical User Interface* (*GUI)* para criar aplicações comerciais, além de um grande conjunto de aplicações de software para várias áreas de comércio e indústria (Oracle, 1999).

Desta forma, pretende-se montar rotas de menor caminho entre um lugar que se deseja partir e um que se deseja chegar via web. Para a especificação será utilizado a UML com a ferramenta Rational Rose e na parte da implementação será utilizada o *ORACLE 8i.*

1.1 OBJETIVOS DO TRABALHO

O objetivo principal do trabalho é desenvolver o protótipo de uma aplicação que permitirá um usuário via web fazer a montagem de rotas de menor caminho entre pontos de

um mapa, utilizando as ferramentas *ORACLE* (banco de dados e ferramentas de desenvolvimento).

Os objetivos específicos do trabalho são:

a) permitir armazenar no banco de dados mapas e informações sobre este;
b) permitir que via web o usuário interaja com o sistema através de um mapa e que seja possível informar pontos (de origem e de destino) e que seja retornado o menor caminho para o percurso do trajeto entre estes mesmos pontos.

1.2 ESTRUTURA DO TRABALHO

No capítulo 2 é apresentado uma revisão bibliográfica sobre Sistemas de Informações Geográficas (SIG), *Automated Mapping/Facilities Management* (AM/FM) que é um dos sistemas especializados de SIG e sobre modelagem da dados geográficos em SIG utilizando-se Geo-OMT.

O capítulo 3 descreve banco de dados espaciais, citando-se alguns existentes atualmente no mercado e conceitua ORACLE 8i, que foi o banco de dados utilizado neste trabalho para armazenamento de informações (mapas e informações do mesmo).

O capítulo 4 relata a publicação de dados geográficos pela web e sobre a*pplets* e *servlets* que são programas escritos em Java e foram utilizados para o desenvolvimento deste protótipo.

O capítulo 5 apresenta o desenvolvimento do protótipo, começando pela especificação em *Unified Modeling Language* (UML) e Geo-OMT e seguindo até a implementação, demonstrando as classes e funcionalidades do protótipo.

No sexto capítulo são apresentados as conclusões, limitações e trabalhos futuros.

2 SISTEMAS DE INFORMAÇÕES GEOGRÁFICAS

Sistema de Informação Geográfica (SIG) são sistemas automatizados usados para armazenar, analisar e manipular dados geográficos, dados estes que representam objetos e fenômenos em que a localização geográfica é características indispensável à informação (Câmara, 1996).

O termo SIG é aplicado para sistemas que realizam o tratamento computacional de dados geográficos. Um SIG armazena a geometria e os atributos dos dados que estão geo-referenciados, isto é, localizados na superfície terrestre segundo uma projeção cartográfica (Medeiros, 1999).

Os Sistemas de Informações Geográficas comportam diferentes tipos de dados e aplicações, em várias áreas de conhecimento e têm sido concebidos com o principal objetivo de desenvolver novas tecnologias na gestão de cidades. Principalmente as grandes aglomerações urbanas vêem-se às voltas com problemas de abastecimento (água tratada, gás, energia elétrica, etc.), telecomunicações, esgotamento sanitário, controle das condições ambientais, controle de tráfego, cadastro imobiliário e outros (Camargo, 1997).

Como principais características de SIG é possível indicar (Medeiros, 1999):

a) inserir e integrar, numa única base de dados, informações espaciais provenientes de dados cartográficos, dados censitários, cadastro urbano e rural, imagens de satélite, redes e modelos numéricos de terreno;
b) oferecer mecanismos para combinar as várias informações, através de algoritmos de manipulação e análise, bem como para consultar, recuperar, visualizar e graficar o conteúdo da base de dados geo-referenciados.

Num SIG, há duas formas de organizar um ambiente de trabalho: organização baseada num banco de dados geográficos e organização baseada em projetos.

No primeiro, o usuário define inicialmente o esquema conceitual associado às entidades do banco de dados geográficos, indicando para cada tipo de dados seus atributos não-espaciais e as representações geométricas associadas. Procede-se de maneira similar em um banco de dados tradicional (como o dBASE, ACCESS ou ORACLE), onde a definição da estrutura do banco precede a entrada dos dados. Por exemplo, o SPRING e o MGE são

exemplos de sistemas organizados com base na modelagem do banco de dados, antes da inserção dos dados geográficos.

No segundo, o usuário define inicialmente um referencial geográfico, que delimita uma região de trabalho, e a seguir define as entidades geográficas que compõem o projeto. Por exemplo, o ARC/INFO, IDRISI e SGI são exemplos desta classe de sistemas (Medeiros, 1999).

Existem vários tipos de SIG, cada um apresentando propósitos distintos e servindo a diferentes tipos de tomada de decisão. Uma variedade de nomes tem sido aplicadas para diferentes tipos de SIG, distinguindo suas funções e papéis. Um dos sistemas especializados mais comuns, por exemplo, é normalmente citado como sistema AM/FM. AM/FM é projetado especificamente para gerenciamento da infra-estrutura (Meneguete, 1998).

2.1 MAPEAMENTO AUTOMÁTICO / GERÊNCIA DE SERVIÇOS (AM/FM)

AM/FM são sistemas de automação do processo de mapeamento e gerenciamento das informações representadas por itens em um mapa (Meneguete, 1998).

Automated Mapping and Facilities Management System (AM/FM) ou Mapeamento Automatizado e Gerenciamento de Serviços possuem a capacidade de banco de dados para armazenar informações sobre objetos mapeados e juntar estes dados as informações do mapa, mas geralmente não possuem capacidades de análise espacial ou estrutura de dados topológica. Combinam uma parte das capacidades dos sistemas CADD para gráficos interativos, entrada e técnicas de armazenamento com a capacidade da base de dados. O objetivo do sistema é converter mapas manuais e registros em base de dados digitais para seleção e processamento de ordem de trabalho (Jacob, 1999).

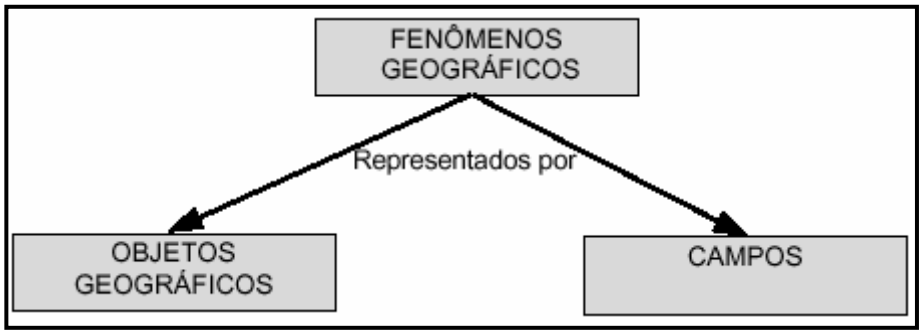
As relações entre os componentes do sistema de utilidade pública são definidas como redes (*Networks*) que são associadas à atributos, permitindo assim modelar e analisar a operação do sistema de utilidade pública. Atributos não-gráficos podem ser ligados aos dados gráficos (Meneguete, 1998).

2.2 CARACTERIZAÇÃO DE DADOS GEOGRÁFICOS

O termo dado espacial significa qualquer tipo de dado que descreve fenômenos aos quais esteja associada alguma dimensão espacial. Os dados utilizados em SIG pertencem a uma classe particular de dados espaciais: os dados geo-referenciados ou dados geográficos. O termo denota dados que descrevem fatos, objetos e fenômenos do globo terrestre associados à sua localização sobre a superfície terrestre, num certo instante ou período de tempo (Thome, 1998).

Para se construir um Sistema de Informação Geográfica torna-se necessário criar um modelo que represente os fenômenos geográficos. Tal modelo representa estes fenômenos como dois grandes tipos de dados: objeto geográfico e campos (Figura 1).

Figura 1 – Representação de Fenômenos Geográficos



Fonte: Thome, 1998

Os objetos geográficos são individualizáveis e possuem identificação com elementos do mundo real. São compostos de duas partes. Uma parte define a representação geométrica do objeto geográfico (ponto, linha, polígono), que é apresentada graficamente na tela do computador e a outra que define sua descrição alfanumérica, como por exemplo seu nome, tamanho, área, perímetro, etc (Thome, 1998).

Campos são variações espaciais contínuas que representam grandezas distribuídas espacialmente no mundo real. Cada localização no espaço está associada a um conjunto de valores da variável espacial representada.

2.3 MODELAGEM DE DADOS GEOGRÁFICOS EM SIG

Um modelo de dados é um conjunto de conceitos que podem ser usados para descrever a estrutura e as operações em um banco de dados. O modelo busca sistematizar o entendimento que é desenvolvido a respeito de objetos e fenômenos que serão representados em um sistema informatizado. Os objetos e fenômenos reais, no entanto, são complexos demais para permitir uma representação completa, considerando os recursos à disposição dos sistemas gerenciadores de bancos de dados (SGBD) atuais. Desta forma, é necessário construir uma abstração dos objetos e fenômenos do mundo real, de modo a obter uma forma de representação conveniente, embora simplificada, que seja adequada às finalidades das aplicações do banco de dados (Borges, 1997).

A orientação a objetos é uma tendência em termos de modelos para representação de aplicações geográficas. Conforme Câmara (1996) “a modelagem orientada a objetos não obriga o armazenamento em um SGBD orientado a objetos, mas simplesmente visa dar ao usuário maior flexibilidade na modelagem incremental da realidade”. Os objetos geográficos se adequam bastante bem aos modelos orientados a objetos ao contrário, por exemplo, do modelo de dados relacional que não se adequa aos conceitos natos que o homem tem sobre dados espaciais.

Uma extensão da *Object Modeling Technique* (OMT), denominado Geo-OMT, para utilização em aplicações geográficas, fornece primitivas para modelar a geometria e a topologia dos dados espaciais, suportando diferentes estruturas topológicas, múltiplas visões dos objetos e relacionamentos espaciais. Dessa forma, ele supre as principais limitações dos modelos hoje mais utilizados, provendo maiores facilidades para a modelagem de aplicações geográficas (Borges, 1997).

O modelo Geo-OMT apresenta as seguintes características (Davis, 2001):

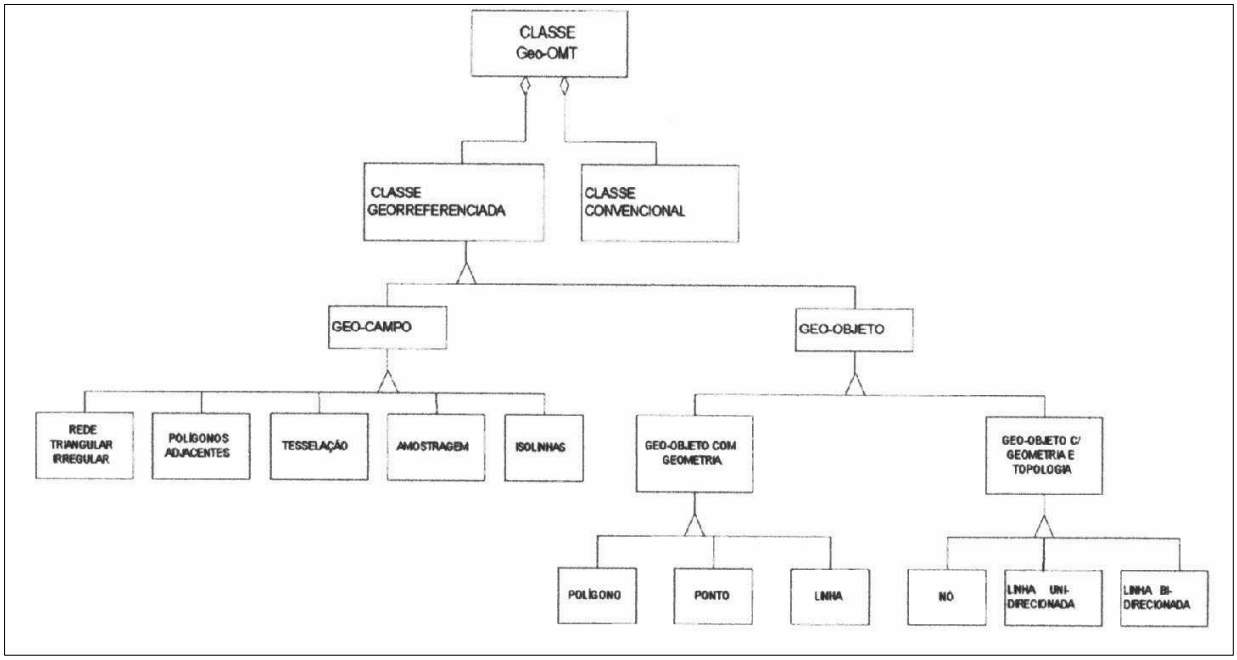
a) segue o paradigma de orientação a objetos suportando os conceitos de classe, herança, objeto complexo e método;
b) representa e diferencia os diversos tipos de dados envolvidos nas aplicações geográficas, fazendo uso de uma representação simbólica que possibilita a percepção imediata da natureza do dado, eliminando assim, a extensa classe de hierarquias utilizada para representar a geometria e a topologia dos objetos

espaciais;

c) fornece uma visão integrada do espaço modelado, representando e diferenciando classes com representação gráfica (geo-referenciadas) e classes convencionais (não-espaciais), assim como os diferentes tipos de relacionamento entre elas;
d) caracteriza as classes em contínuas e discretas, utilizando os conceitos de “visão de campos” e “visão de objetos”;
e) representa a dinâmica da interação entre os vários objetos, explicitando tanto as relações espaciais como as associações simples;
f) representa as estruturas topológicas “todo-parte” e de rede;
g) formaliza as possíveis relações espaciais, levando em consideração a forma geométrica da classe;
h) traduz as relações topológicas e espaciais em restrições de integridade espaciais;
i) representa os diversos fenômenos geográficos, utilizando conceitos natos que o ser humano tem sobre dados espaciais;
j) possibilita a representação de múltiplas visões de uma mesma classe geográfica, tanto baseada em variações de escala, quanto nas várias formas de se perceber o mesmo objeto no mundo real;
k) é de fácil visualização e entendimento, pois utiliza basicamente os mesmos tipos construtores definidos no modelo OMT;
l) não utiliza o conceito de camadas e sim o de níveis de informação (temas), não limitando o aparecimento de uma classe geográfica em apenas um nível de informação;
m) é independente de implementação.

O modelo Geo-OMT trabalha no nível conceitual/representação e suas classes básicas são: Classes Georreferenciadas e Classes Convencionais (Figura 2). Através dessas classes são representados os três grandes grupos de dados (contínuos, discretos e não-espaciais) encontrados nas aplicações geográficas, proporcionando assim, uma visão integrada do espaço modelado, o que é importante na modelagem principalmente de ambientes urbanos.

Figura 2 – Meta modelo parcial do modelo Geo-OMT



Fonte: Davis, 2001.

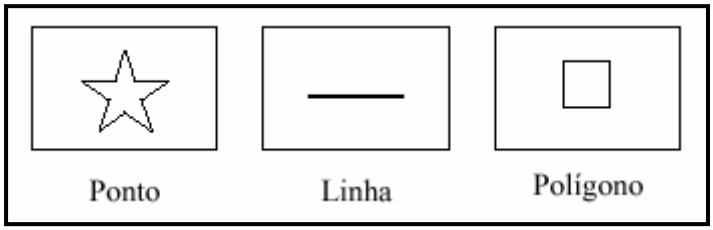
Tanto as classes georreferenciadas como as classes convencionais podem ser especializadas, utilizando o conceito de herança da orientação a objetos. O modelo Geo-OMT formaliza a especialização das Classes Georreferenciadas em dois tipos de classes: Geo-Campo e Geo-Objeto (Davis, 2001).

No primeiro, representam-se objetos distribuídos continuamente pelo espaço, correspondendo a grandezas como tipo de solo, topografia e teor de minerais. No segundo representam-se objetos geográficos individualizáveis, que possuem identificação com elementos do mundo real, como lotes, rios e postes.

Em relação às subclasses georreferenciadas, todas elas apresentam uma representação simbólica, construindo assim um sistema semântico onde cada símbolo possui um significado próprio que incorpora a sua natureza e a geometria (Davis, 2001).

Um Geo-objeto pode ser representado por três pictogramas (Figura 3). O ponto representa um símbolo como por exemplo uma árvore, a linha representa segmentos de reta formados por uma linha simples, um arco ou por uma polilinha (ex. muro, trecho de rua, trecho de circulação) e o polígono representa uma área (ex. edificação, lote).

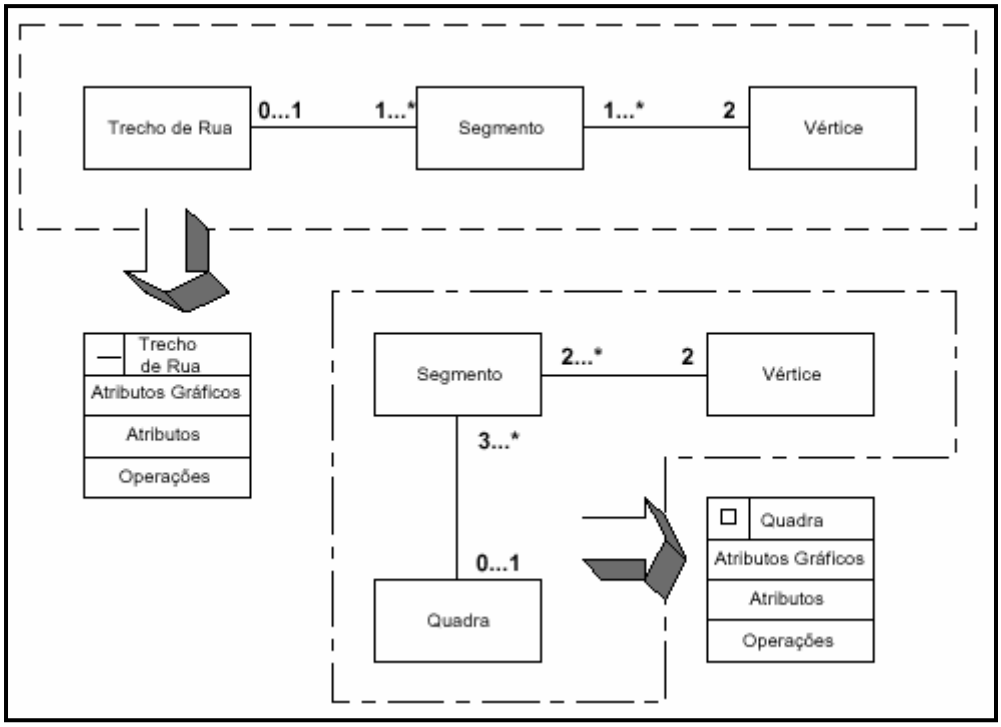
Figura 3 - Pictogramas da classe Geo-Objeto



Fonte: Davis, 2001.

Nas aplicações urbanas, esta simplificação pode ser facilmente percebida devido à grande quantidade de entidades do mundo real presentes em um esquema da aplicação. Na Figura 4 está exemplificada a simplificação por substituição simbólica.

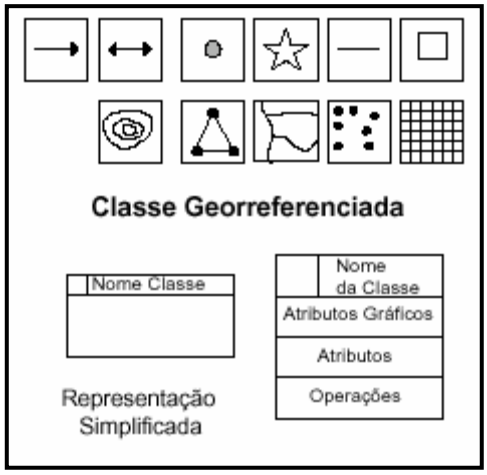
Figura 4 - Exemplo de simplificação por substituição simbólica



Fonte: Davis, 2001.

Uma Classe Georreferenciada é representada graficamente por um retângulo, subdividido em quatro partes. A parte superior contém à direita o nome da classe e à esquerda o símbolo representando a forma gráfica da Classe Georreferenciada. Na segunda parte, aparece a lista dos atributos gráficos. Na terceira parte, a lista dos atributos alfanuméricos e, na última parte, a lista das operações que são aplicadas à classe. Esta terá sempre o atributo gráfico de localização (Figura 5).

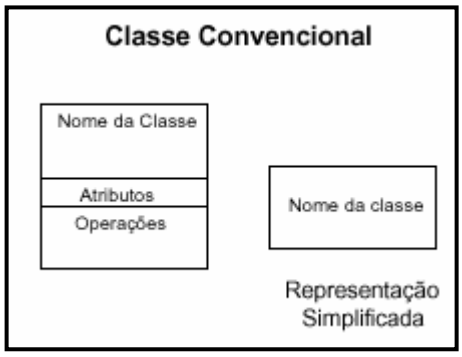
Figura 5 – Classe Georreferenciada



Fonte: Davis, 2001.

A notação gráfica utilizada para *Classes Convencionais* corresponde à notação usada no modelo OMT. Uma classe é representada graficamente por um retângulo subdividido em três partes contendo, respectivamente, o nome da classe, a lista dos atributos e a lista das operações que são aplicadas à classe (Figura 6).

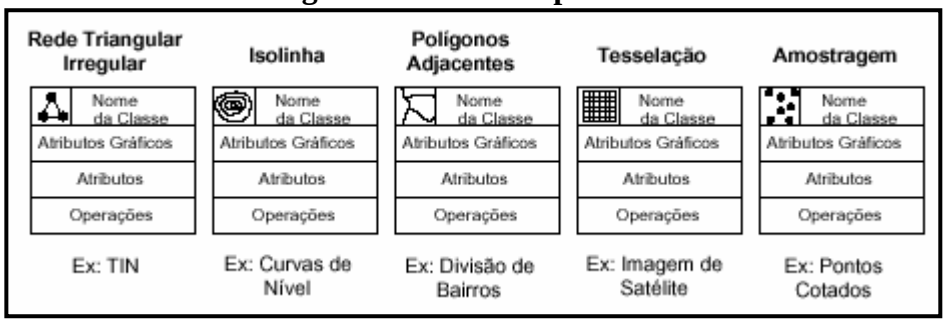
Figura 6 – Classe Convencional



Fonte: Davis, 2001.

O modelo Geo-OMT possui cinco classes do tipo *Geo-Campo: Isolinhas, Polígonos Adjacentes, Tessselação, Amostragem* e *Rede Triangular Irregula*r. Cada uma dessas classes possui um padrão simbólico de representação, conforme representado na Figura 7.

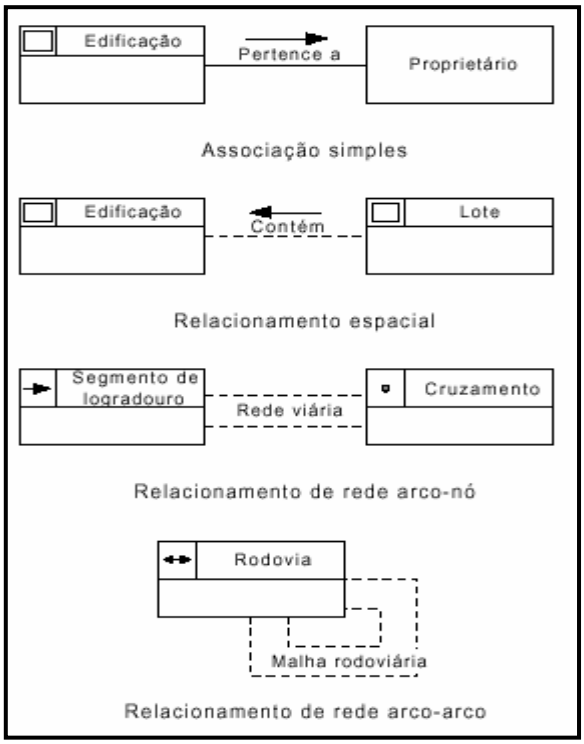
Figura 7 – Geo-Campos



Fonte: Davis, 2001.

Considerando a importância dos relacionamentos espaciais e não espaciais para a compreensão do espaço modelado, o modelo Geo-OMT representa os três tipos de relacionamento que podem ocorrer entre suas classes: associações simples, relacionamentos topológicos em rede e relacionamentos espaciais (Figura 8). Associações simples representam relacionamentos estruturais entre objetos de classes diferentes, convencionais ou georreferenciadas. Relacionamentos espaciais representam relações topológicas, métricas, ordinais e *fuzzy*. Relacionamentos de rede são relacionamentos entre objetos que estão conectados uns com os outros.

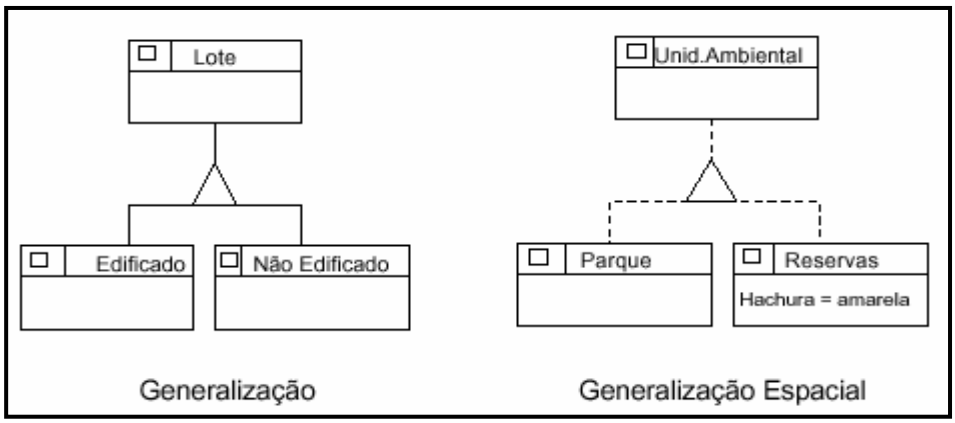
Figura 8 – Relacionamentos



Fonte: Davis, 2001.

Generalização é o processo de definição de classes que são mais genéricas (superclasses) do que classes com características semelhantes (subclasses). No modelo Geo-OMT as abstrações de generalização e especialização se aplicam tanto a classes georreferenciadas quanto a classes convencionais. Se as propriedades gráficas (por exemplo, cor, tipo de linha, etc.) variarem nas subclasses, é utilizada a generalização espacial, onde as subclasses herdam a natureza gráfica da superclasse mas variam suas propriedades gráficas. A notação utilizada na generalização espacial só varia no tipo de linha utilizada na ligação entre a superclasse e as subclasses. A linha contínua é substituída pela pontilhada (Davis, 2001) (Figura 9).

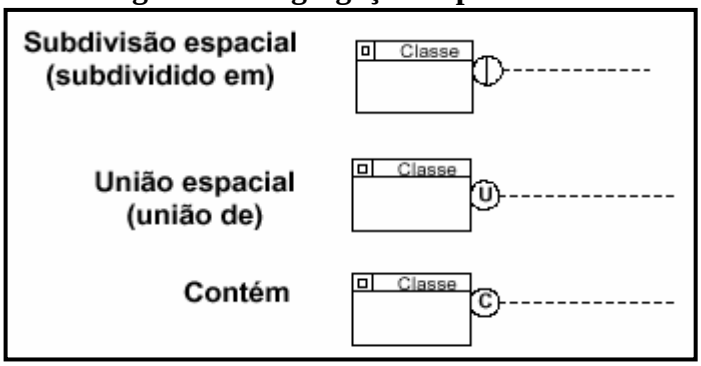
Figura 9 – Generalização



Fonte: Davis, 2001.

A agregação é uma forma especial de associação entre objetos, onde um deles é considerado composto por outros. Uma agregação pode ocorrer entre Classes Convencionais, entre Classes Georreferenciadas e entre Classes Georreferenciadas e Classes Convencionais. A agregação espacial é um caso especial de agregação onde são explicitados relacionamentos topológicos "todo-parte". A estrutura topológica "todo-parte" foi subdividida em: subdivisão espacial, união espacial e contém. A notação das três estruturas é apresentada na Figura 10.

Figura 10 – Agregação Espacial



Fonte: Davis, 2001.

Entre vários modelos de dados geográficos atuais, o Geo-OMT se mostra mais adequado pela sua facilidade de “leitura” e visualização dos relacionamentos, seja dos dados espaciais ou não. Além disto, o GEO-OMT é apenas um modelo conceitual, sem implicações na implementação (Davis, 2001).

3 BANCO DE DADOS ESPACIAIS

O núcleo central de um sistema de informação geográfica é um banco de dados espacial e atualmente a comunidade de Ciência da Computação vem dedicando grande atenção ao tema, com o objetivo geral de aplicar à gestão de dados geográficos paradigmas já estabelecidos em aplicações convencionais (estruturas de dados, linguagens de consulta, controle de concorrência, gerência de transações, bancos de dados distribuídos).

O interesse pelo uso de SIG no ambiente corporativo levou ao aparecimento de gerenciadores de dados geográficos, que armazenam tanto a geometria como os atributos dos objetos dentro de um sistema gerenciador de bancos de dados (SGBD). As principais vantagens desta estratégia são (Câmara, 2000):

a) evitar os problemas de controle de integridade típicos do ambiente "*desktop*", permitindo o acesso concorrente aos dados;
b) facilitar a integração com as bases corporativas já existentes, como sistemas legados[1], que já utilizam SGBDs relacionais.

Estes gerenciadores possuem dois componentes, usualmente de distintos fabricantes: um SGBD com suporte a dados geográficos e uma "camada de acesso", que fornece um ambiente de armazenamento e recuperação, visível externamente ou integrado a um "GIS desktop". Por enquanto, os SGBD geográficos armazenam apenas representações vetoriais (pontos, redes e polígonos), com três classes principais de tecnologia (Câmara, 2000) (Figura 11):

[1] Sistemas de informação desatualizados, que resistem às mudanças tecnológicas.

Figura 11 - SGBD com suporte a dados espaciais

	Armazenamento	Indexação espacial	Interface Progr.	Integrac. Produtos
Oracle Spatial	Tabelas normalizadas	Quad-trees	C, SQL/MM, OpenGIS	GeoMedia, MapInfo, SDE
DB2 Spatial Extender	Tipos de dados espaciais	-	C++, SQL	SDE
Informix Data Blade	Tipos de dados espaciais	-	C++, SQL	SDE, MapInfo SpatialWare

Fonte: Câmara, 2000.

a) SGBD relacional no qual as representações de dados geográficos são armazenadas como atributos numéricos em tabelas. Cada ponto é representado por um par de coordenadas x,y (cada um numa coluna separada) e cada arco é guardado como referência a dois pontos (o inicial e o final). Reconstruir a geometria de um polígono complexo requer a junção de dados de múltiplas tabelas, o que pode limitar o desempenho do sistema para grandes massas de dados. Esta é a estratégia adotada no ORACLE SPATIAL;
b) SGBD relacional com suporte para campos longos (um campo longo é uma cadeia binária aonde se pode armazenar informação gráfica, numérica ou pictórica). O acesso ao conteúdo dos campos longos é deixado a cargo de uma interface de aplicação. Esta é o caso do COMPUTER ASSOCIATES INGRES;
c) SGBD extensível (tipicamente relacional com suporte a orientação a objetos), que permite a definição de novos tipos de dados (como linhas e polígonos). A idéia neste caso é que os dados espaciais são simplesmente um tipo de dados adicional em um SGBD relacional-objeto. Este é o caso do INFORMIX DATABLADE e do IBM DB2 SPATIAL EXTENDER.

As "camadas de acesso" interagem com estes servidores, para fornecer um ambiente de armazenamento e recuperação, através de uma interface de programação. Estes produtos podem ser divididos em duas grandes classes (Câmara, 2000) (Figura 12):

Figura 12 - Gerenciadores de dados geográficos

	Área de Aplicac.	Integrac. Produtos	Gerência Transacão	Interface Program.	SGBDs Suportados
GeoMedia Pro	Geral	GeoMedia	-	VBA	Oracle
MapInfo SpatialWare	Geral	MapInfo	-	C++, VBA	Oracle, Informix
ModelServer Continuum	Geral	Microstation Geographics	-	VBA	Oracle
SDE/Esri	Geral	Arc/Info, Arc/View	-	C, Avenue	Oracle, DB2, Informix
SmallWorld	Gerência Redes	-	Controle versões	Smallworld Magik	Oracle
VISION*	Gerência Redes	AutoDesk World	"Check-in, check-out"	Express (VBA-like)	Oracle

Fonte: Câmara, 2000.

a) Ambiente de programação genéricos que fornecem funções de acesso aos bancos de dados geográficos. Como exemplo tem-se SDE/ESRI e MAPINFO/SPATIALWARE.

b) Sistemas de Gerência de Dados, especialmente para aplicações de redes (AM/FM), que oferecem, além do acesso aos dados, mecanismos sofisticados de controle de transações (como *check-in/check-out* ou gerência de versões). Como exemplo tem-se VISION e SMALLWORLD GIS.

As interfaces oferecidas pelo "software de acesso", na maior parte dos casos, implicam o uso de soluções proprietárias. Como alternativa, alguns fabricantes estão adotando interfaces padronizadas, como as preconizadas pelo consórcio OpenGIS, que definiu, para o caso de dados vetoriais, um conjunto de tipos de dados espaciais e um conjunto de funções para acesso e manipulação (Câmara, 2000).

3.1 BANCO DE DADOS ORACLE

O ORACLE possui diversos tipos primitivos de dados como `caracter, number, date e LOB`. Além destes ele possui tipos definidos pelo usuário como os tipo objeto (`object type`) e coleção (`colection type`). Um tipo objeto é uma estrutura definida pelo usuário que representa uma abstração das entidades do mundo real, possuindo um conceito semelhante ao de classes em Orientação a Objeto. Uma coleção é um grupo de elementos ordenados ou não, sendo todos os elementos do mesmo tipo podendo ser constituídas de tipos primitivos, tipos objeto ou referência.

3.1.1 ORACLE SPATIAL

Oracle8i Spatial é uma extensão do *Oracle8i* que permite armazenamento geométrico, indexação e funções de buscas espaciais. Previamente especificado como *Oracle Spatial Cartridge*, tem estado disponível para Oracle8 desde 1997. Na versão Oracle7 o produto era conhecido como *Spatial Data Option* e tinha a mesma funcionalidade básica. Ele é projetado para dois grupos de usuários:

a) Banco de dados tradicionais para adicionar consultas espaciais para suas aplicações;
b) Suportam sistemas de informações geográficas que necessitam armazenar, recuperar, e gerenciar grandes bancos de dados espaciais que contenham centenas de gigabytes de dados geográficos.

Oracle8i Spatial consiste de funções e procedimentos destinados para o armazenamento e recuperação de primitiva (ponto, linha, polígono) e composto (conjunto de pontos, polígonos com buracos) de recursos geométricos bidimensionais. Esses objetos são armazenados em um formato predefinido. Os objetos são espacialmente ordenados com um algoritmo de decomposição hierárquica espacial conhecido como um *quad-tree linear* (Oracle, 1998).

Oracle8i Spatial consiste de quatro componentes principais:

a) Uma estrutura que prescreve a armazenamento, sintaxe e semântica de suporte geométrico de tipos de dados;

b) Um mecanismo de indexação espacial;

c) Um conjunto de operadores espaciais;

d) Um conjunto de utilitários administrativos.

O *Oracle8*i *Spatial* utiliza tabelas *ORACLE 8i* que podem ser acessadas e carregadas com comandos SQL. As informações são armazenas em colunas do tipo SDO_GEOMETRY (Oracle, 1998).

O quadro 1 demonstra o armazenamento dos pontos de um polígono em uma tabela *ORACLE* utilizando o *Oracle8*i *Spatial*.

Quadro 1 – Armazenamento de um polígono no Banco ORACLE

```
// Polígono de exemplo, que possui 8 pontos
POLIGONO1 = [P1(6,15), P2(10,10), P3(20,10), P4(25,15), P5(25,35),
             P6(19,40), P7(11,40), P8(6,25), P1(6,15)]

// criação da tabela para armazenamento das informações
CREATE TABLE TABELA1 (NOME_OBJ  VARCHAR2(32),
                      PONTOS    MDSYS.SDO_GEOMETRY);

// inserção dos pontos que formam o polígono
INSERT INTO TABELA1 VALUES
       ('POLIGONO1', MDSYS.SDO_GEOMETRY(3, NULL,NULL,
        MDSYS.SDO_ELEM_INFO_ARRAY(1,3,1),
        MDSYS.SDO_ORDINATE_ARRAY(6,15, 10,10, 20,10, 25,15, 25,35,
                                 19,40, 11,40,  6,25,  6,15)));
```

Após armazenar os objetos no banco de dados, é possível efetuar pesquisas utilizando as funções geométricas do *Oracle8*i *Spatial*. Entre várias funções disponíveis, pode-se citar (Oracle, 1998):

a) SDO_GEOM.LENGTH: retorna o tamanho ou perímetro da geometria;

b) SDO_GEOM.SDO_POLY_DIFFERENCE: gera um polígono representando a diferença entre dois polígonos;

c) SDO_GEOM.RELATE: determina se dois objetos interagem;

d) SDO_GEOM.VALIDATE_GEOMETRY: determina se a geometria é válida.

3.1.2 LINGUAGEM PL/SQL

A linguagem *Procedural Language*/SQL (PL/SQL) é uma extensão de linguagem procedural da ORACLE Corporation para SQL. A linguagem de acesso a dados padrão para bancos de dados relacionais. A linguagem PL/SQL oferece recursos de engenharia de software modernos, como a encapsulação de dados, o tratamento de exceções, a orientação a objetos, entre outros.

Além de permitir a manipulação de dados, a linguagem também permite que as instruções de consulta da linguagem SQL sejam incluídas em unidades procedurais de código e estruturas em blocos, tornando a linguagem SQL uma linguagem avançada de processamento de transações. Com a mesma pode-se usar as instruções SQL para refinar os dados do ORACLE e as instruções de controle PL/SQL para processar os dados (Oracle, 2000).

Por ser uma linguagem de programação procedural permite criar objetos de esquema tais como: `procedures, functions, triggers, packages e cursores.`

A linguagem PL/SQL é uma linguagem estruturada em blocos, o que significa que os programas podem ser divididos em blocos lógicos. Um bloco PL/SQL consiste em até três seções: declarativa (opcional), executável (necessária) e tratamento de exceções (opcional) e possuem a seguinte estrutura básica (Figura 13).

Figura 13 – Estrutura básica de um bloco PL/SQL

```
[ DECLARE
-  Declarações de variáveis, cursores, exceções definidas pelo usuário]
BEGIN
- Instruções SQL
- Instruções PL/SQL
[ EXCEPTION
- Ações a serem desempenhadas quando ocorrem erros ]
END;
```

A linguagem PL/SQL desempenha um papel central tanto para o *Oracle Server* (através de procedimentos armazenados, funções armazenados, gatilhos de banco de dados e pacotes) quanto para as ferramentas de desenvolvimento Oracle (através de gatilhos de componente do *Oracle Developer*) (Oracle, 2000).

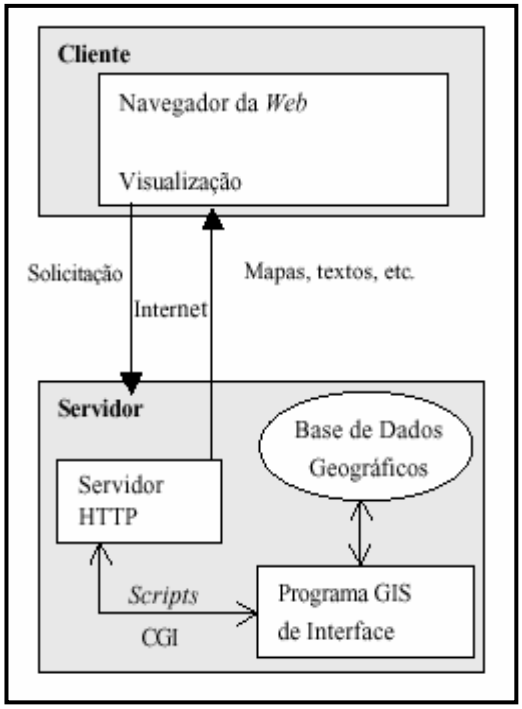
4 APLICAÇÕES WEB

Um dos desafios crescentes para as instituições que lidam com informações geográficas é a publicação de dados através da Internet. Por sua natureza gráfica e bidimensional, o ambiente WWW (*World Wide Web*) oferece uma mídia adequada para a difusão da geoinformação. A médio prazo, espera-se que a disponibilidade “on-line” de grandes bases de dados espaciais e de ferramentas eficientes de navegação torne a geoinformação acessível de forma ampla, sem a necessidade de aquisição de software específico (Câmara, 2000).

As diferentes maneiras utilizadas para publicar dados geográficos na Internet, em geral, recaem no modelo clássico da Internet (cliente/servidor). Essa arquitetura é composta de um cliente com capacidade de exibição de dados geográficos e um programa de servidor de geoprocessamento sendo executado juntamente com o servidor web para responder às solicitações sobre os dados geográficos (Gattass, 2000).

A Figura 14 representa essa arquitetura. Tipicamente o cliente é um navegador (como Netscape ou IExplorer) que realiza solicitações a um servidor web utilizando o protocolo HTTP *(Hypertext Transfer Protoco*l). Quando o cliente realiza a solicitação de um mapa ou outra operação sobre os dados geográficos, o servidor web a repassa para o servidor de geoprocessamento, que atende à solicitação e envia a resposta para o cliente através do servidor web.

Figura 14 – Arquitetura Típica para a Publicação de Dados Geográficos na *Web*



Fonte: Gattass, 2000.

A comunicação entre os dois servidores é feita, em geral, através de interfaces como CGI *(Common Gateway Interfac*e), ISAPI *(Internet Server Application Program Interface* – Microsoft) e NSAPI (API do servidor Netscape) (Gattass, 2000).

Para facilitar a difusão de informação geográfica através da Internet, a solução mais adotada é acoplar um servidor de dados geográficos. As tecnologias disponíveis podem ser enquadradas genericamente em duas grandes classes:

a) servidores de imagens, que, respondendo a pedidos remotos, enviam uma imagem (matriz) de tamanho fixo nos formato GIF ou JPEG. Esta solução permite configurar o servidor para responder a diferentes tipos de consulta, sem requerer que todos os dados a ser transmitidos sejam pre-computados. Entretanto, o usuário consegue visualizar apenas as imagens enviadas; qualquer novo pedido é enviado de volta para o servidor, resultando em mais uma transferência pela Internet. Como exemplo, temos o "*Internet Map Server*", da ESRI.
b) servidores de mapas, que adotaram como solução a transmissão de dados no formato vetorial. Estes servidores encapsulam a informação em formatos gráficos, que podem ser visualizados por meio de programas adicionais (*plug-ins*) acoplados a *browsers* ou por meio de *applets* Java. Esta estratégia permite uma maior

flexibilidade do lado do cliente, que pode realizar operações locais de visualização e consulta sob os dados transferidos. O tempo de acesso inicial para transferência é maior que no caso anterior, mas muitas das operações posteriores serão realizadas localmente, o que resulta usualmente num tempo de resposta médio melhor. Exemplo são os produtos “Geomedia Web Map” da INTERGRAPH, “Map Guide” da AUTODESK e “SpringWeb” do INPE.

Esta arquitetura está baseada no modelo de camadas múltiplas, sendo utilizadas três camadas (Câmara, 2001):

a) um navegador para Internet que serve como um cliente universal. Acoplado ao navegador tem-se um *applet* Java. O *applet* é responsável por funções ligadas ao cliente tais como: apresentação de mapas e objetos geográficos, controle de diálogos e interfaces e controle de geração de expressões em linguagem de consulta;
b) servidor de aplicação: constituída de um servidor para protocolos HTTP com capacidade de executar *servlets* em JAVA. O *servlet* é um aplicativo que permanece em execução no servidor aguardando por solicitações dos clientes e tem a capacidade de atender diversas solicitações simultâneas. O *applet* pode estabelecer uma conexão com qualquer servidor onde exista um *servlet* preparado para receber suas requisições;
c) servidor de dados: composta por um Sistema Gerenciador de Banco de Dados Relacional que armazena todos os dados utilizados pelo sistema. O *servlet* acessa as informações do banco de dados relacional utilizando a API JDBC da linguagem Java que é um conjunto de especificações que define como um programa escrito em Java pode se comunicar e interagir com um banco de dados.

4.1 APPLET

Applets são pequenos programas escritos em linguagem Java que são embutido em páginas web. Pelo fato deles poderem aproveitar da classe de interface gráfica fornecida pelas bibliotecas de classes de Java padrão, os *applets* podem ser usados para implementar interfaces com o usuário muito mais ricas do que é tipicamente possível usando-se apenas

HTML. Quando os usuários acessam essas páginas, os *applets* são descarregados (*downloaded*) para seus computadores e executadas com um *browser* habilitado para Java. *Applets* são tratados pelo *browser* da mesma forma que qualquer outro recurso de mídia, como arquivos de imagens, áudio ou vídeo (Souza, 2000).

Em vez da atividade ocorrer do lado do servidor, como no caso do CGI (*Common Gateway Interface*), ela acontece do lado do cliente. Isso quer dizer que os a*pplets* não são restritos por largura de banda ou velocidade do *modem* quando estão sendo executados. Os usuários podem ver e ouvir efeitos multimídia em páginas web de forma eficaz e oportuna (Hoff, 1997).

O *applet* tem como funções básicas o controle de apresentação do modelo e dos dados geográficos e seus atributos aos usuários. Além disso o *applet* se preocupa em manter um *cache* de dados do lado do cliente de forma a minimizar a transferência de dados de forma desnecessária entre o cliente e o servidor.

4.2 SERVLETS

Um *servlet* é um programa escrito em linguagem Java que é carregado dinamicamente para atender às solicitações de um servidor web, ou seja, é uma extensão acrescentada ao servidor que aumenta a sua funcionalidade. Os servidores web respondem às solicitações dos usuários, geralmente, usando o protocolo HTTP através do envio de documentos escritos em HTML (Gattass, 2000).

Em outras palavras poderia se dizer que os *servlets* estão para os servidores web assim como os *applets* (componente de um navegador gráfico e não uma aplicação) estão para os navegadores, com a diferença que os *servlets* não possuem interfaces gráficas.

Os *servlets* podem ser carregados em diversos servidores, pois a API utilizada para escrevê-los usa apenas o ambiente da Máquina Virtual Java do servidor. O protocolo HTTP é o mais utilizado para a comunicação com o servidor. Os *servlets* podem ser usados para o processamento de formulários (HTML), interagir com bancos de dados, ou de uma maneira

geral atuar como uma camada intermediária em uma arquitetura de três camadas, como apresentado esquematicamente na figura 15 (Gattass, 2000).

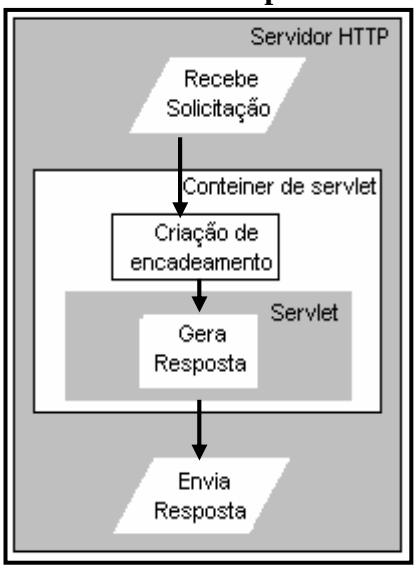
Figura 15 - Arquitetura de execução de um *servle*t.

Fonte: Gattass, 2000.

Os *servlets* de Java têm um modelo de programação similar aos *scripts* de CGI, na medida em que recebem uma solicitação de HTTP de um navegador da web como entrada e espera-se que localizem e/ou construam o conteúdo apropriado para a resposta do servidor (Fields, 2000).

Mas diferentemente dos programas tradicionais CGI que necessitam da criação de um novo processo para tratar de cada nova solicitação, todos os *servlets* associados com um servidor da web rodam dentro de um único processo, que roda em uma *Java Virtual Machine* (JVM). Como ilustrado na Figura 16, ao invés de criar um processo para cada solicitação, a JVM cria um encadeamento de Java para tratar de cada solicitação do *servlet*. Os encadeamentos de Java têm muito menos *overhead* do que os processos completos, e são executados dentro da memória do processador já alocado pela JVM, tornando a execução do *servlet* consideravelmente mais eficiente do que o processamento CGI. Já que a JVM persiste além da vida de uma única solicitação, os *servlets* também podem evitar muitas operações demoradas, como conexão a um banco de dados, ao compartilhá-los entre todas as solicitações.

Figura 16 – Processo de servidor para rodar *servlets*



Fonte : Fields, 2000

Há várias alternativas para a montagem de páginas, e as principais são (Gattass, 2000):

a) CGI é um padrão de interface utilizado entre servidores HTTP e programas para se comunicarem. O servidor executa um programa (chamado *scrip*t) cujo resultado é então transmitido ao cliente. Normalmente, um *script* CGI gera como resultado uma página HTML para ser exibida no navegador. Os programas CGI são executados na máquina onde está localizado o servidor web. A principal desvantagem de se utilizar *scripts* CGI é que para atender a cada requisição do usuário o servidor web precisa criar um novo processo (cada processo produz uma nova conexão com o banco de dados utilizado e o servidor web tem que esperar até que os resultados lhe sejam enviados), o que resulta numa baixa eficiência. Outro ponto negativo é com relação à segurança, uma vez que os arquivos *(script*s) não ficam inteiramente protegidos, pois devem ser armazenados nos sub-diretórios CGI-bin do servidor. Mais uma grande desvantagem está relacionada à baixa taxa de reutilização do código, em função das linguagens usadas para a programação;
b) API proprietárias estão incluídas em muitos servidores web. Exemplos mais difundidos são a NSAPI (API do Servidor da Netscape), na qual os programadores podem criar módulos binários que irão acrescentar e/ou substituir elementos para autenticação, autorização ou geração dinâmica de conteúdos, e ISAPI *(Internet Server Application Program Interface* – Microsoft), que tem como princípio básico criar uma DLL (*Dinamically Linked Libraries*) que é carregada no servidor quando

o HTTP é inicializado e permanece instalada enquanto for necessária. Esta DLL irá gerenciar a conexão com a base de dados sem a necessidade de se criarem novas conexões, como ocorre com o CGI. O grande problema em adotar tais soluções é que elas são proprietárias, perdendo-se a portabilidade. Outro problema é que geralmente estas aplicações são desenvolvidas utilizando-se linguagens de programação como C ou C++, que podem interferir no bom funcionamento do servidor web;

c) *Active Server Pages* (ASP) é a solução da Microsoft para a criação de páginas dinâmicas. Ele fica embutido nas páginas HTML, não sendo pré-compilado. O servidor web processa um arquivo utilizando uma DLL (ASP.DLL) que irá interpretar os comandos ASP embutidos na página HTML. O grande problema desta solução é que ela está ligada a um tipo particular de servidor – o *Internet Information Server* (IIS) – ou a alguma ferramenta que consiga fazer a tradução do ASP antes de enviar a resposta para o cliente.

Segundo Goodwill (1999), as principais razões que indicam o uso de Java *servlets* são as seguintes:

a) eficiência: a inicialização de um *servlet* é feita apenas uma vez, ou seja, após ele ser carregado pelo servidor web, as novas solicitações são apenas chamadas do método de serviço;
b) persistência: os *servlets* podem manter o estado entre solicitações;
c) portabilidade: eles são escritos usando Java e, por este motivo, são portáteis, bastando utilizar o JVM (*Java Virtual Machine*) do servidor, sem fazer uma alteração sequer no código-fonte;
d) robustez: Java possui um método bem definido para o tratamento de erros que minimiza a perda de memória por alocações indevidas *(garbage collecto*r);
e) segurança: os *servlets* por si só herdam toda a segurança que é peculiar a um servidor web, porém os Java *Servlets* também podem contar com o *Java Security Manager*;
f) reutilização: como os *servlets* são escritos em Java, eles possuem as vantagens da utilização de uma linguagem orientada a objetos, como a possibilidade de reutilização.

4.3 JAVA SERVER PAGES

Java Server Pages (JSP) é uma tecnologia baseada em Java que simplifica o processo de desenvolvimento de páginas com conteúdo dinâmico. A própria funcionalidade JSP é tipicamente implementada usando-se a tecnologia *servlet*.

JSP são arquivos texto, normalmente com a extensão .jsp, que substituem as páginas HTML tradicionais. Os arquivos JSP contém HTML tradicional junto com código embutido sendo possível o acesso de dados do código de Java rodando no servidor. Quando a página é solicitada por um usuário e processada pelo servidor de *HyperText Transport Protocol* (HTTP), a parte HTML da página é transmitida. No entanto, as partes de código da página são executadas no momento em que a solicitação é recebida e o conteúdo dinâmico gerado por este código é unido na página antes de ser enviada para o usuário. Isto propicia uma separação dos aspectos de apresentação HTML da página, da lógica de programação contida no código (Fields, 2000).

A JSP oferece diversos benefícios como um sistema para a geração de conteúdo dinâmico. Primeiramente, sendo uma tecnologia baseada em Java, ela se aproveita de todas as vantagens que a linguagem Java fornece em relação a desenvolvimento e acionamento. Como uma linguagem orientada a objetos com forte tipagem, encapsulamento, tratamento de exceções e gerenciamento de memória automática, o uso de Java conduz a uma produtividade aumentada do programador e a código mais robusto. Pelo fato de haver uma API padrão publicada para JSP, e pelo fato do *bytecode* de Java compilado ser portável através de todas as plataformas que aceitam uma *Java Virtual Machine* (JVM), o uso de JSP não o restringe ao uso de uma plataforma de hardware, sistema operacional ou software de servidor específico (Fields, 2000).

Além disso, pelo fato dela permitir completo acesso a plataforma de Java principal, a JSP pode se aproveitar de todas as outras API de Java padrão, incluindo aquelas para acesso a banco de dados compatíveis com muitas plataformas, serviços de diretórios, processamento distribuído e criptografia. Esta habilidade em alavancar uma ampla gama de fontes de dados, recursos de sistema e serviços de rede significa que a JSP é uma solução altamente flexível para a criação de aplicações baseadas na web e repleta de recursos (Fields, 2000).

A própria JSP oferece diversas vantagens como um sistema para geração de conteúdo dinâmico. Dentre estas vantagens, encontram-se a performance aperfeiçoada na CGI e um modelo de programação que enfatiza o *design* de aplicações centradas em componentes. Este modelo de programação permite que os programadores usem a JSP para manter uma rígida separação entre os elementos de apresentação de uma aplicação da web e sua implementação básica. Esta separação, por sua vez, facilita uma divisão do trabalho dentro da equipe de desenvolvimento da web, ao estabelecer uma interface bem definida entre o desenvolvimento da aplicação e o design da aplicação (Fields, 2000).

4.4 JDBC

Diferentemente de outras linguagens de criação de *scripts* na web, a JSP não define seu próprio conjunto de *tags* para acesso a banco de dados. Ao invés de desenvolver um outro mecanismo para acesso a banco de dados, os *designers* de JSP escolheram utilizar a API de banco de dados Java (JDBC). Quando uma aplicação JSP precisa se comunicar com um banco de dados, ela o faz através de uma classe *driver* fornecida pelo fornecedor gravada na API JDBC (Fields, 2000).

JDBC é um padrão de comunicação, definido pela Sun Microsystems e utilizado na linguagem Java, permitindo uma aplicação cliente “conversar” ou “enxergar” uma aplicação servidora, no caso um SGBD.

Com o JDBC é possível:

a) estabelecer uma conexão com um banco de dados;

b) enviar declarações SQL;

c) processar os resultados.

5 DESENVOLVIMENTO DO PROTÓTIPO

Este trabalho visa desenvolver um protótipo de uma aplicação que permita via web consultar informações do tipo *raster*[2] e vetorial[3], sendo que estas informações estão armazenadas em um banco de dados ORACLE 8i. E para especificação deste trabalho foi utilizado a ferramenta *Rational Rose*.

Para resolução deste trabalho pode-se utilizar a arquitetura baseada em tecnologia Java, utilizando-se um *applet* no ambiente do cliente e um *servlet* no ambiente do servidor, e propõe uma forma de interface com o usuário que irá produzir consulta, que serão pré-processadas pelo *applet* e enviadas ao *servlet* se necessário.

As imagens utilizadas neste protótipo, são imagens reais da cidade de Rio do Sul. Estas estão armazenadas em um diretório no servidor.

5.1 ESPECIFICAÇÃO

Para especificação deste trabalho utilizou-se o *Unified Modeling Language* (UML) composto de diversos diagramas, dos quais:

a) diagrama de caso de uso;
b) diagrama de classe e
c) diagrama de sequência.

Através do Modelo Geo-OMT é possível visualizar a modelagem geográfica.

Através do Modelo de Entidade e Relacionamento são apresentadas as tabelas utilizadas neste protótipo.

---

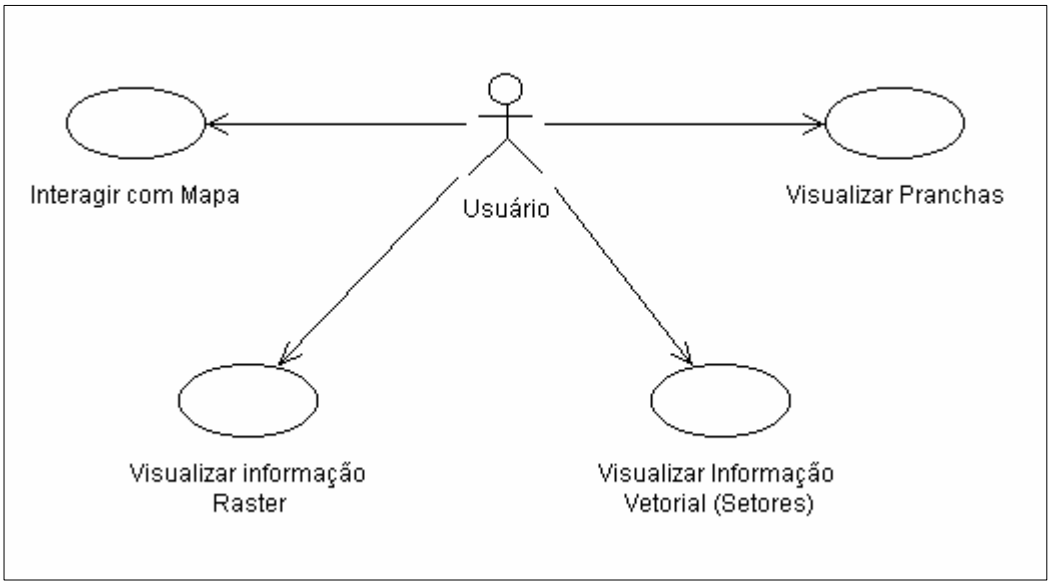
[2] RASTER (ou digital) é definido como informações não simbolizadas por equações matemáticas.

[3] VETORIAL (ou numérico) normalmente é constituído de um arranjo de coordenadas numéricas e mnemônicos que tratados por funções gráficas específicas transformam-se em elementos gráficos pontuais, lineares e areais em sua representação em um dispositivo de saída.

5.1.1 DIAGRAMA DE CASO DE USO

Na figura 17 está representado o diagrama de caso de uso, que demonstra as possíveis interações do usuário com o protótipo.

Figura 17 - Diagrama de casos de uso para o protótipo



No caso de uso “Interagir com Mapa” ocorre as interações que o usuário efetua com relação ao mapa, que pode ser através das operações de *ZoomIn, ZoomOut*, Restaurar Mapa, *Pan* ou ainda visualização das coordenadas.

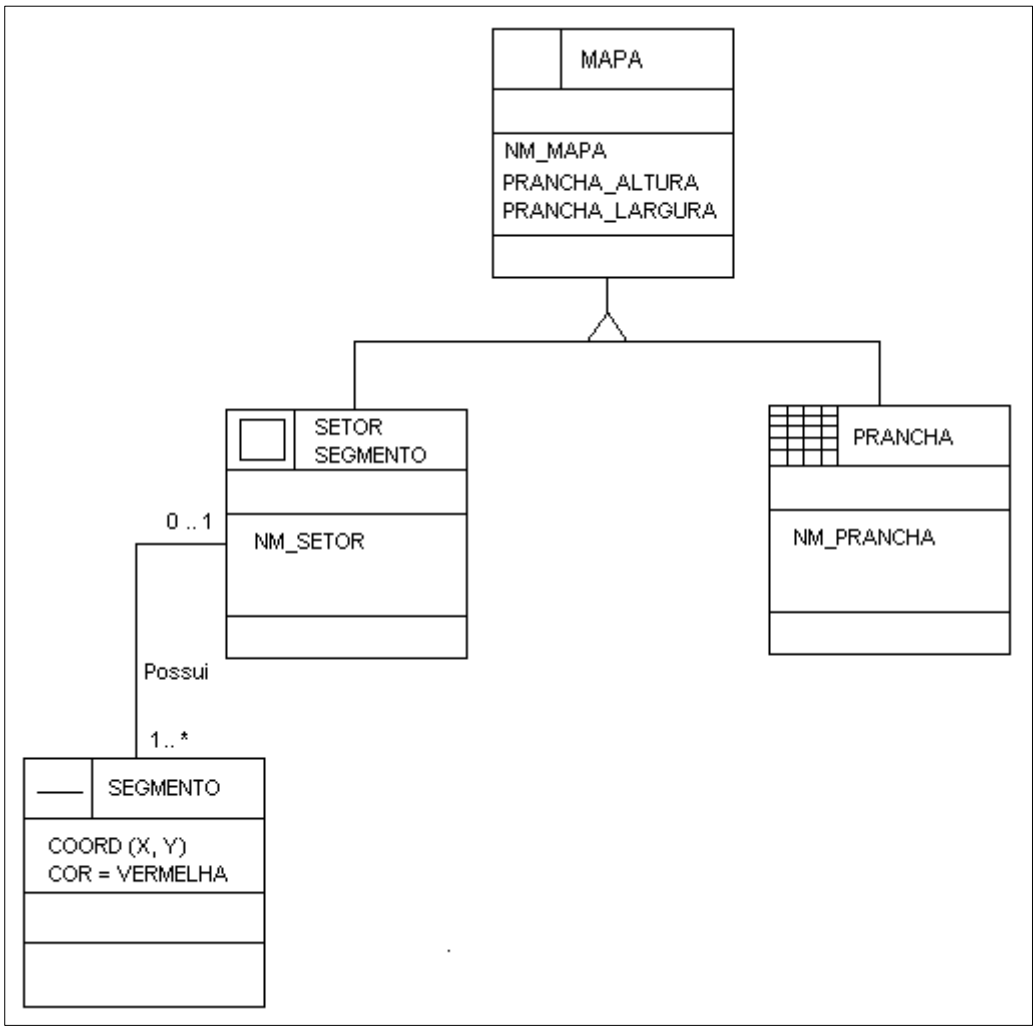
O usuário terá a possibilidade de visualizar as informações *raster* e vetorial. A informação *raster* é representada através das imagens da cidade e as informações vetoriais através dos segmentos que formam os setores, ou ainda, visualizando as descrições dos próprios setores.

E ainda é possível visualizar as grades de pranchas que é cada parte da imagem do mapa da cidade.

5.1.2 MODELO GEO-OMT

Para modelagem de dados geográficos utilizou-se o Geo-OMT, conforme demonstrado na figura 18.

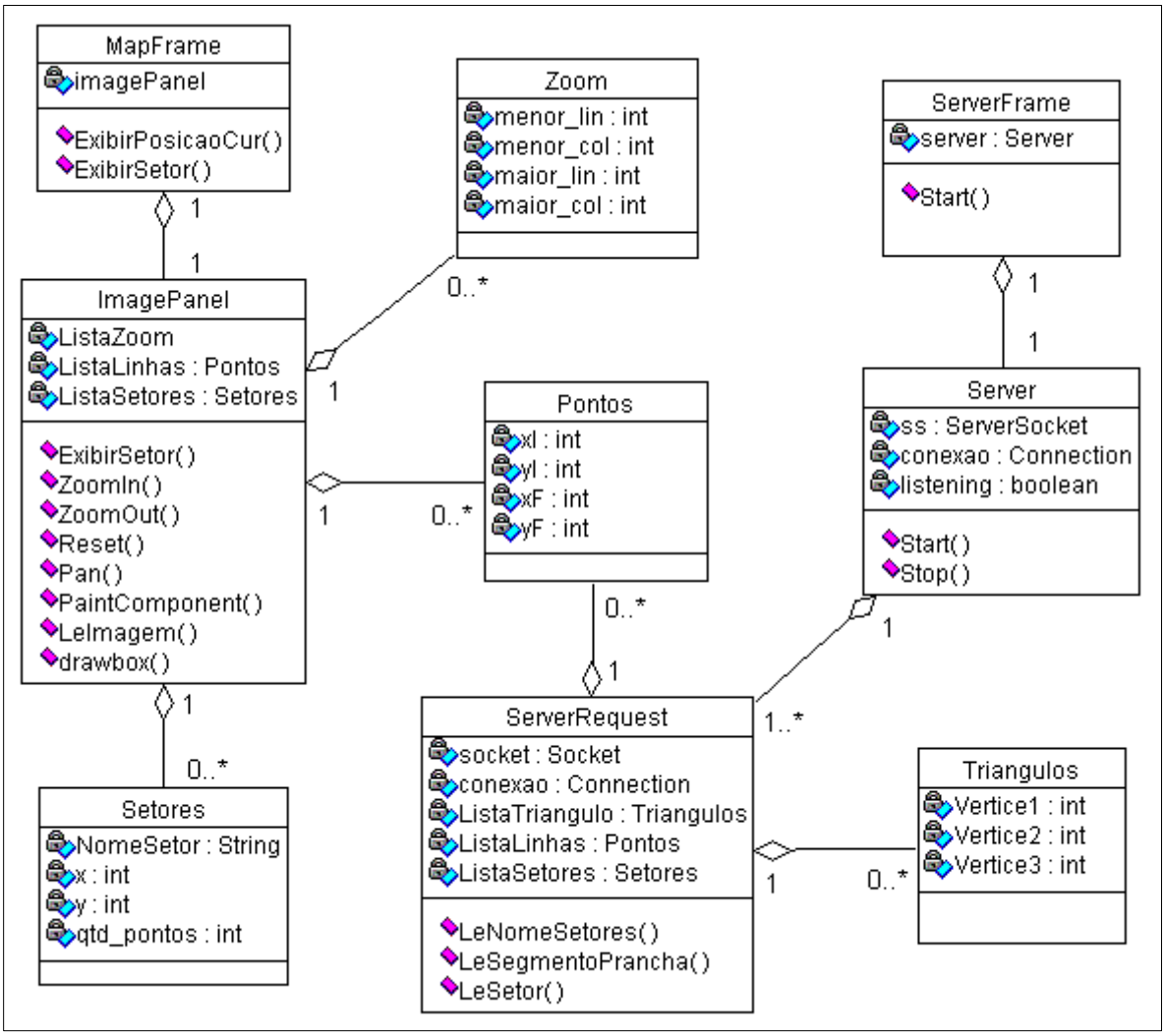
Figura 18 – Modelo Geo-OMT



5.1.3 DIAGRAMA DE CLASSE

Na figura 19 está representado o diagrama de classe. Para a representação do diagrama de Classe foi utilizada a ferramenta *Rational Rose*.

Figura 19 - Diagrama de classe do protótipo



Para a implementação deste trabalho foram criadas as seguintes classes:

a) ServerFrame: classe de interface do software servidor;

b) Server: classe utilizada para ficar aguardando requisições do software cliente;

c) ServerRequest: classe instanciada pela classe Server para cada requisição do software cliente e que possui rotinas que acessam o banco de dados;

d) MapFrame: classe de interface do software cliente;

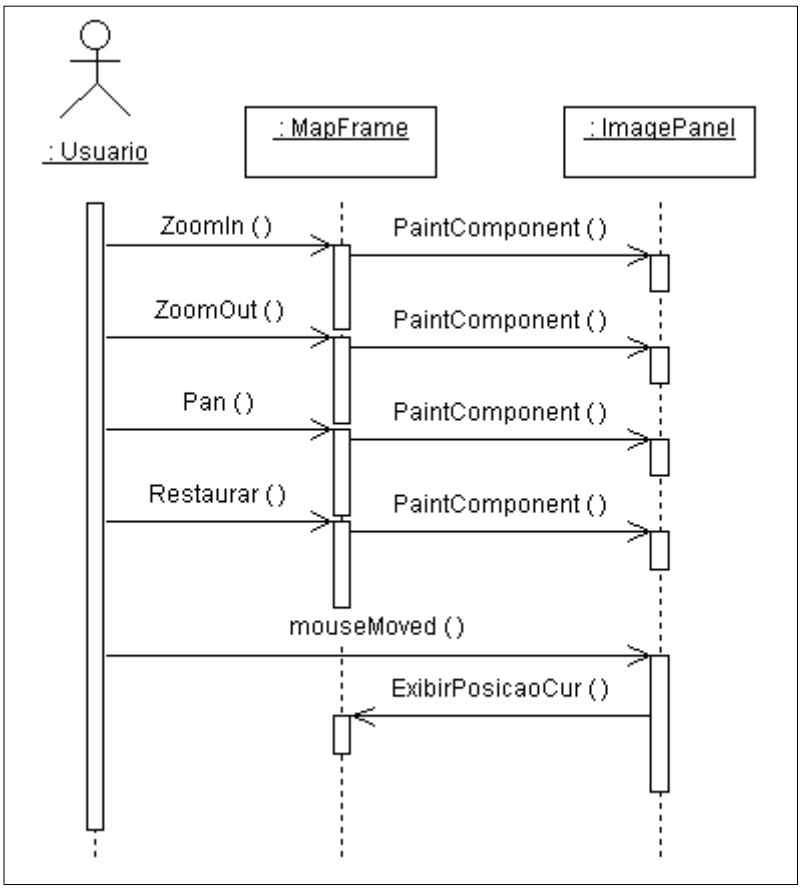
e) ImagePanel: classe responsável pela exibição das informações espaciais (*raster* e vetorial), que são basicamente as imagens do mapa e desenho das linhas que formarão os setores. Esta classe é também responsável por controlar as ações (eventos) do usuário no mapa, como por exemplo, a seleção de áreas para fazer *Zoom* e exibição de coordenadas do mapa;

f) Triângulos, Setores, Pontos e Zoom: classes utilizadas para auxiliar no armazenamento de informações em Vetores.

5.1.4 DIAGRAMA DE SEQUÊNCIA

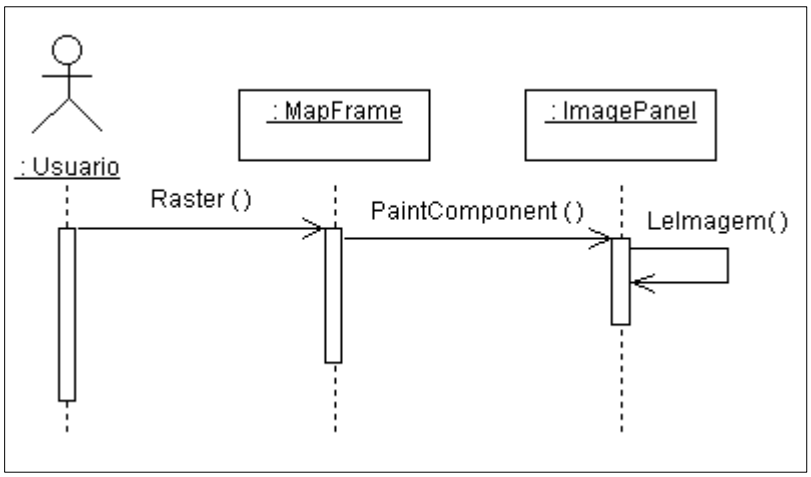
Através do diagrama de sequência é representado os objetos e a troca de “mensagens”, conforme Figura 20 que demonstra o caso de uso “Interagir com Mapa”.

Figura 20 – Diagrama de Sequência - Interagir com Mapa



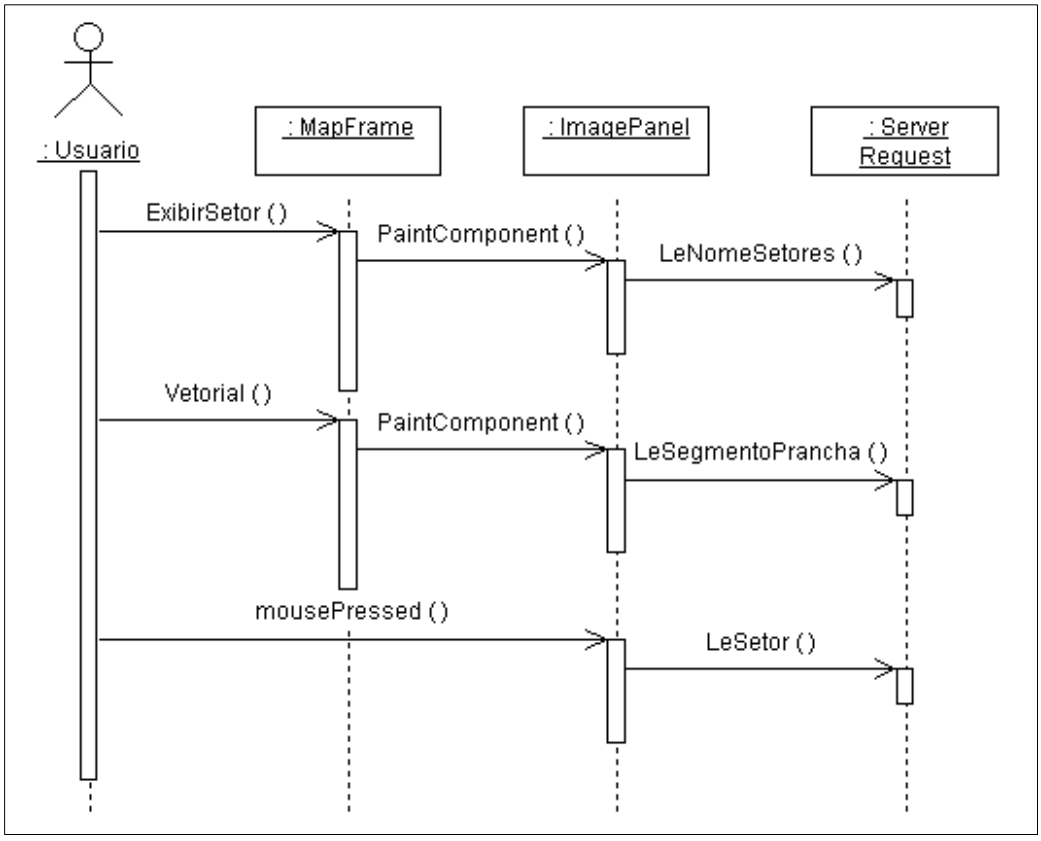
O caso de uso “Visualizar Informação *Raster*” está demonstrado na Figura 21.

Figura 21 – Diagrama de Sequência - Visualizar Informação Raster



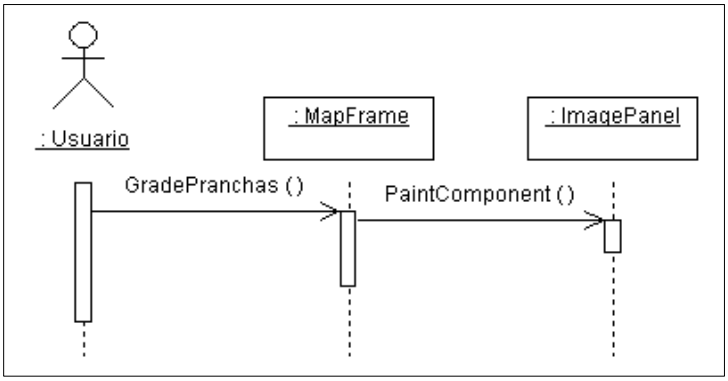
Através da Figura 22 é possível visualizar o caso de uso “Visualizar informação Vetorial (Setores)”.

Figura 22 – Diagrama de Sequência - Visualizar Informação Vetorial



O caso de uso “Visualizar Pranchas” está demonstrado na Figura 23.

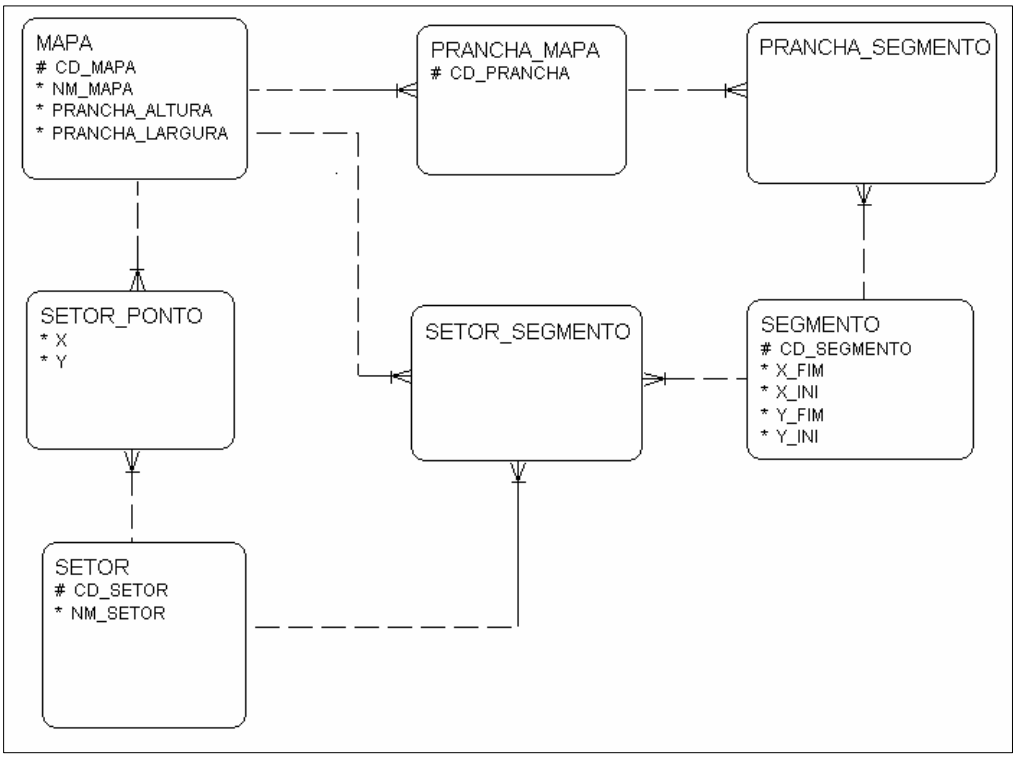
Figura 23 – Diagrama de Sequência - Visualizar Pranchas



5.1.5 MODELO ENTIDADE RELACIONAMENTO

Através da Figura 24 é possível visualizar o Modelo de Entidade e Relacionamento, que possui todas as tabelas utilizadas no protótipo e os relacionamentos das mesmas com suas cardinalidades. Tabelas estas utilizadas para armazenar dados necessários para a visualização das informações vetoriais. Para a representação do Mer, foi utilizado a ferrementa *Case* da *ORACLE*, denominada *ORACLE Designer 6i*.

Figura 24 – Modelo de Entidade e Relacionamento do Protótipo



Com relação as tabelas utilizadas pelo protótipo, pode-se relacionar as necessidades destas, da seguinte forma:

a) Mapa: tabela utilizada para armazenar informações básicas sobre os mapas das cidades;

b) Prancha_Mapa: armazenar as pranchas pertencentes aos mapas;

c) Prancha_Segmento: armazenar os segmentos de reta que serão exibidos para cada prancha;

d) Setor: armazenar informações básicas sobre os setores;

e) Segmento: armazenar as coordenadas x e y (inicial e final) dos segmentos de retas utilizados para formar os setores;

f) Setor_Ponto: armazenar as coordenadas dos vértices que formarão os setores. Estas informações são utilizidas para exibição dos nomes dos setores;

g) Setor_Segmento: armazenar os segmentos de retas que formarão cada setor.

5.2 IMPLEMENTAÇÃO

Para implementação deste trabalho, houve a necessidade de dividir o protótipo em duas partes: o software servidor e o software cliente.

5.2.1 SOFTWARE SERVIDOR

O quadro 2 demonstra uma parte do código da classe *ServerFrame* usado somente como interface com o usuário, que permite informar a porta que será utilizada. Esta classe instanciará a classe *Server*.

Quadro 2 – Classe ServerFrame

```
try {
   // obter o nro da porta, informado na tela para utilizar na
   // comunicacao entre softwawre server e cliente
   Integer n = new Integer(this.jTextFieldPort.getText());
```

```
   // instanciar a classe SERVER que ficará aguardando por
   // requisicoes do software cliente
   this.server = new Server(n.intValue(),this.jTextAreaLog);
   new Thread(this.server).start();
} catch(NumberFormatException nfe) {
   this.jTextAreaLog.append("Porta deve ser um numero !\n");
}
```

O Quadro 3 mostra o método *start* da classe *Server* que tem o objetivo de criar uma conexão com o bando de dados *ORACLE* e ficar "escutando a porta" (a que foi definida na classe *ServerFrame*). Quando isto ocorrer será criado uma *Thread* que irá instanciar a classe *ServerRequest* que possui as rotinas que necessitam buscar informações do banco.

Quadro 3 – Método *start* da classe Server

```
public void start() {
   // variavel utilizada p/ saber se está aguardando
   // requisicoe do software cliente.
   this.listening = true;
   try {
     // criar conexão com o banco de dados oracle. Esta conexao será criada
     // somente uma vez e passada como parametro para cada requisicao
     // do software cliente.
     DriverManager.registerDriver(new
                           oracle.jdbc.driver.OracleDriver());
     Conexao= DriverManager.getConnection
           ("jdbc:oracle:oci8:@banco02", "system", "manager");
     this.ss = new ServerSocket(getPort());
     // laço faz o processamento somente se o server continuar "ouvindo"
     // requisicoes do software cliente.
     while (listening) {
        // ler requisicao do socket
        Socket socket = this.ss.accept();
        if (listening) {
           // apos ter recebido requisicao, instanciar classe
           // ServerRequest que possui rotinas de acesso ao
           // banco de dados
           Thread t = new Thread(new ServerRequest(socket,
                                 conexao, this.log));
           t.start();
        }
```

```
     }
   }
   // fechar socket e conexao com o banco de dados.
   this.ss.close();
   conexao.close();
}
```

A comunicação de informações (objetos) entre o software servidor e o software cliente ocorre da seguinte forma: o software cliente cria um *socket* e através da classe *ObjectOutputStream* é feito a serialização da informação. Através da classe *ObjectInputStream* ocorre o retorno do software servidor, como demonstrado no quadro 4.

Quadro 4 – Comunicação entre software servidor e software cliente

```
// obter o endereco do host para ser utilizado pelo socket
String host = url.getHost();
Socket socket = new Socket(host, 5500);
// criar objeto para comunicação ( transmissão ). Para cada writeObject
// deverá ser dado um readObject, pois estas informações estão no Buffer.
ObjectOutputStream oos = new ObjectOutputStream(new
BufferedOutputStream(socket.getOutputStream()));
// mandar qual rotina chamar
oos.writeObject("Rotina");
// mandar demais parâmetros, se necessário
oos.writeObject("Paramatro01");
// a função flush é utilizada para serialização das informações
oos.flush();
// criar objeto para comunicação ( recepção )
ObjectInputStream ois = new ObjectInputStream(new
BufferedInputStream(socket.getInputStream()));

// Retorno da informação que foram serializadas

Var01 = ois.readObject();
```

Da mesma forma ocorre no software servidor, que está definido na classe *ServerRequest*. Este processo pode ver visto no quadro 5. Neste mesmo quadro pode ser observadado a implementação de uma das rotinas utilizadas na classe *ServerRequest*.

Quadro 5 – Classe *ServerRequest*

```
try {
   // criar objeto para comunicação. Será recebido parâmetros enviados pelo
```

```
   // software cliente.
   ObjectInputStream ois = new ObjectInputStream(new
BufferedInputStream(this.socket.getInputStream()));
   // ler primeiro parametro, que é a rotina solicitada
   Operacao = (String) ois.readObject();
   // testa qual rotina foi recebida
   If (operacao.equalsIgnoreCase("LeSegmentosPrancha")) {
      // criação de variáveis para utilização na seleção de informações
      Statement sql = null;
      ResultSet rs = null;
      // ler outro parametro enviado, que identifica de que prancha será
      // lida os segmentos de linha.
      String prancha = (String) ois.readObject();
      sql = conexao.createStatement();
      // executa select para retornar informações necessárias
      rs = sql.executeQuery
          ("select x_ini, y_ini, x_fim, y_fim " +
           "  from prancha_segmento ps, segmento s " +
           " where s.cd_segmento = ps.cd_segmento " +
           "   and cd_prancha = '" + prancha + "'" +
           "   and cd_mapa = " + mapa);
      // laço para ler registros retornados do select executado
      while (rs.next()) {
          // pegar valores retornados p/ cada coluna do select
          int x_ini = rs.getInt(1);
          int y_ini = rs.getInt(2);
          int x_fim = rs.getInt(3);
          int y_fim = rs.getInt(4);
          // foi criado a classe Pontos para auxiliar no armazenamentos
          // de informações que serão retornadas para o software cliente.
          Pontos tmpLinhas;
          tmpLinhas = new Pontos();
          tmpLinhas.xI = x_ini;
          tmpLinhas.yI = y_ini;
          tmpLinhas.xF = x_fim;
          tmpLinhas.yF = y_fim;
          // armazenar estrutura em um Vector que no final
          // será retornado para o software cliente.
          ListaLinhas.add(tmpLinhas);
```

```
       }
      // criar objeto para comunicao ( transmissao ) e
      // transmitir informações de retorno
      ObjectOutputStream oos = new ObjectOutputStream(new
BufferedOutputStream(this.socket.getOutputStream()));
      oos.writeObject(ListaLinhas);
      oos.flush();
      oos.close();
      rs.close();
      sql.close();
  }
```

5.2.2 SOFTWARE CLIENTE

No software cliente, pode-se visualizar as informações espaciais de diversas formas, as quais se pode citar:

a) grade de pranchas;
b) nível r*aster* (*background*);
c) nível vetorial;
d) descrição dos setores e
e) coordenadas.

A implementação da opção de grades de pranchas, está exibido no quadro 6.

Quadro 6 - Rotina para exibir as Grades das Pranchas

```
// exibir Grade das Pranchas
  if (Grade) {
     // setar cor das linhas a serem desenhadas para a grade
     g.setColor(Color.black);
     // variáveis utilizadas para posicionamento da grade
     int Pos_X, Pos_Y;
     int Cont_lin = 0, Cont_col = 0;
     // laço para percorrer as pranchas exibidas na tela
     for ( int i = menor_lin; i <= maior_lin; i++ ) {
        for ( int j = menor_col; j <= maior_col; j++ ) {
```

```
            // encontrar posicao xy inicial p/ desenhar prancha
            Pos_X = (Cont_col * LarguraPrancha);
            Pos_Y = (Cont_lin * AlturaPrancha);
            // desenhar retângulo ( que representa a grade )
            g.drawRect(Pos_X, Pos_Y,
                       LarguraPrancha, AlturaPrancha);
            Cont_col++;
         }
         Cont_col = 0;
         Cont_lin++;
      }
   }
```

O objetivo da exibição do Nível *Raster* (*background*), é exibir diversas pranchas como imagens que formar a imagem final do mapa da cidade. O quadro 7 mostra como é implementado esta rotina.

Quadro 7 - Desenhar *Background*

```
// Exibir imagens de background
if (Background) {
   int Pos_X, Pos_Y;
   int Cont_lin = 0, Cont_col = 0;
   String NomImagem = null;
   // laço para percorrer as pranchas exibidas na tela
   for ( int i = menor_lin; i <= maior_lin; i++ ) {
      for ( int j = menor_col; j <= maior_col; j++ ) {
         // encontrar posicao xy inicial p/ desenhar imagem
         Pos_X = (Cont_col * LarguraPrancha);
         Pos_Y = (Cont_lin * AlturaPrancha);
         // encontrar nome da imagem que basea-se na posição da prancha.
         // o nome é formado por nros que indicam a linha e coluna.
         NomImagem = "S" + i + j + ".jpg";
         // desenhar imagem na tela conforme altura e largura
         // da prancha. A função LeImagem, faz a leitura da imagem do serv.
         g.drawImage(LeImagem(NomImagem), Pos_X, Pos_Y,
                     LarguraPrancha, AlturaPrancha, this);
         Cont_col++;
      }
```

```
      Cont_col = 0;
      Cont_lin++;
   }
}
```

A informação vetorial é representada através dos segmentos de reta que formam polígonos, e neste trabalho nomeados como setores. Nesta rotina será solicitado ao servidor que retorne a lista dos segmentos que pertencem às pranchas que atualmente estão visíveis, e estes segmentos serão desenhos no mapa. Esta rotina de nível vetorial está representada no quadro 8.

Quadro 8 - Rotina para desenhar informação vetorial

```
// exibir informacao vetorial
if (Vetorial) {
   // setar cor das linhas a serem desenhadas
   g.setColor(Color.red);
   try {
      int deslocamento_X, deslocamento_Y;
      int Cont_lin = 0, Cont_col = 0;
      String NomPrancha = null;
      // laço para percorrer as pranchas que estão exibidas na tela.
      for ( int i = menor_lin; i <= maior_lin; i++ ) {
        for ( int j = menor_col; j <= maior_col; j++ ) {
          // obter nome da prancha para saber de qual pranchas deve
          // ser lida  os segmentos de linha
          NomPrancha = "S" + i + j;
          // obter o endereço do host para utilizar na comunicação
          String host = url2.getHost();
          Socket socket = new Socket(host, 5500);
          // criar objetos para comunicação
          ObjectOutputStream oos = new ObjectOutputStream(new
              BufferedOutputStream(socket.getOutputStream()));
          // chamar rotina no software servidor, para retornar
          // listas de segmentos com suas posicao (x,y)
          // na prancha.
          oos.writeObject("LeSegmentosPrancha");
          oos.writeObject(NomPrancha);
          oos.flush();
```

```
        ObjectInputStream ois = new ObjectInputStream(new
              BufferedInputStream(socket.getInputStream()));
        // retorno de informacao do software servidor
        ListaLinhas = (Vector) ois.readObject();
        Pontos tmpLinhas;
        tmpLinhas = new Pontos();
        int xI, yI, xF, yF;
        // calcular deslocamento para desenhar o segmentos
        // conforme a prancha que está sendo trabalhada
        // no momento.
        deslocamento_X = (Cont_col * LarguraPrancha);
        deslocamento_Y = (Cont_lin * AlturaPrancha);
        Cont_col++;
        for ( int k = 0; k < ListaLinhas.size(); k ++ ) {
          tmpLinhas = (Pontos) ListaLinhas.elementAt(k);
          // calcular NDC, pois para desenhar os segmentos
          // vai depender do Zoom atual.
          double tmpLargura = ((double) LarguraPrancha /
                               Largura_prancha);
          double tmpAltura  = ((double) AlturaPrancha /
                               Altura_prancha);
          xI = (int) (tmpLinhas.xI * tmpLargura);
          yI = (int) (tmpLinhas.yI * tmpAltura);
          xF = (int) (tmpLinhas.xF * tmpLargura);
          yF = (int) (tmpLinhas.yF * tmpAltura);
          // desenhar segmento de linhas.
          g.drawLine(xI + deslocamento_X,
                     yI + deslocamento_Y,
                     xF + deslocamento_X,
                     yF + deslocamento_Y);
        }
        ois.close();
        oos.close();
          socket.close();
      }
      Cont_col = 0;
      Cont_lin++;
    }
```

```
   }
   catch(Exception e) {
     System.out.println("Erro: " + e.getMessage());
   }
}
```

Para o cálculo da posição de onde será colocado esta descrição no mapa, utilizou-se as fórmulas para cada setor: $x = \sum x$ / qtd pontos e $y = \sum y$ / qtd pontos. A implementação está demonstrada no quadro 9.

Quadro 9 - Rotina para exibir os nomes do setores

```
// exibir nome dos setores
if (NomeSetores) {
  try {
    // obter o endereço do host para utilizar na comunicação
    String host = url2.getHost();
    Socket socket = new Socket(host, 5500);
    ObjectOutputStream oos = new ObjectOutputStream(new
           BufferedOutputStream(socket.getOutputStream()));
    // chamar rotina no software servidor, para retornar
    // listas do nome dos segmentos com a soma das
    // posicoes (x,y) para serem exibidos no mapa.
    oos.writeObject("LeNomeSetores");
    oos.flush();
    ObjectInputStream ois = new ObjectInputStream(new
            BufferedInputStream(socket.getInputStream()));
    // receber vetor com a lista dos nomes dos setores.
    ListaSetores = (Vector) ois.readObject();
    Setores tmpSetores;
    tmpSetores = new Setores();
    // fazer leitura do vetor com a lista dos nomes dos setores para
    // desenhar o nome deste no mapa, para poder identificá-los.
    for ( int k = 0; k < ListaSetores.size(); k ++ ) {
      tmpSetores = (Setores) ListaSetores.elementAt(k);
      // calcular NDC, pois para desenhar os segmentos
      // vai depender do Zoom atual.
      double tmpLargura = ((double) LarguraPanel / Larg_mapa);
      double tmpAltura  = ((double) AlturaPanel  / Alt_mapa);
```

```
      // calcular posicao x, y para que descricao do setor
      // fique no centro do setor.
      int x = (int) ((tmpSetores.x / tmpSetores.qtd_pontos) *
                     tmpLargura);
      int y = (int) ((tmpSetores.y / tmpSetores.qtd_pontos) *
                     tmpAltura);
      // desenhar nome do setor.
      g.drawString(tmpSetores.NomeSetor, x, y);
      ois.close();
      oos.close();
      socket.close();
    }
    catch(Exception e) {
      System.out.println("Erro: " + e.getMessage());
    }
  }
}
```

Outra forma de exibir o nome do setor que se deseja, é o usuário pressionar o botão do *mouse* no mapa, que será retornado o nome do setor em que se está posicionado o cursor do *mouse*. Esta rotina é mostrada no quadro 10.

Quadro 10 - Rotina ExibirSetor – parte software cliente

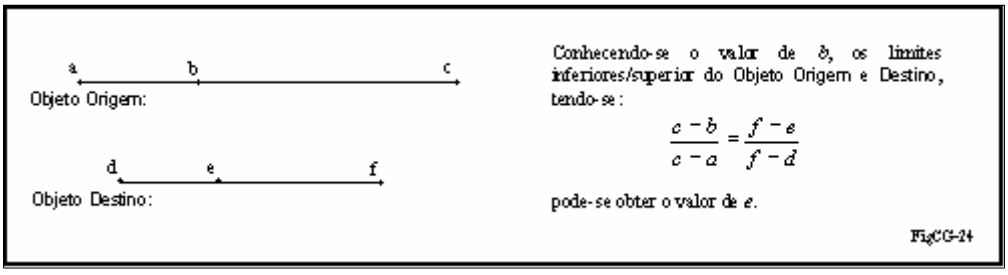
```
// exibir o nome do setor atual.
Void ExibirSetor(int X, int Y) {
   Try {
     // obter o endereço do host para utilizar na comunicação
     String host = url2.getHost();
     Socket socket = new Socket(host, 5500);
     ObjectOutputStream oos = new ObjectOutputStream(new
              BufferedOutputStream(socket.getOutputStream()));
    // chamar rotina no software servidor, para retornar
    // o nome do setor em que esta posicionado com o mouse
    // no mapa.
     oos.writeObject("LeSetor");
     double tmpLargura = ((double) Larg_mapa / LarguraPanel);
     double tmpAltura  = ((double) Alt_mapa / AlturaPanel);
     int x = (int) (X * tmpLargura);
     int y = (int) (Y * tmpAltura);
```

```
      // passar a posicao do mouse como parametro
      oos.writeObject(String.valueOf(x));
      oos.writeObject(String.valueOf(y));
      oos.flush();
      ObjectInputStream ois = new ObjectInputStream(new
                  BufferedInputStream(socket.getInputStream()));
      // receber como retorno o nome do setor
      String Nome_Setor = (String) ois.readObject();
      // exibir o nome do setor na caixa de texto que se
      // encontra acima do mapa
      FrameRoot.ExibirSetor(Nome_Setor);
      ois.close();
      oos.close();
      socket.close();
    }
    catch(Exception e) {
      System.out.println("Erro: " + e.getMessage());
    }
}
```

A implementação da rotina “LeSetor” que se encontra no software servidor pode ser verificado no anexo 1.

Para implementar o procedimento para exibição da coordenada atual no mapa, utilizou-se do algoritmo de Coordenadas de Dispositivos Normalizados (NDC) (Figura 25), como demonstrado no quadro 11.

Figura 25 – Fórmula para cálculo do NDC



Quadro 11 - Rotina de Coordenadas

```
// verificar se ja foi calcular as alturas e larguras das
// pranchas.
if ((LarguraPanel > 0) && (AlturaPanel > 0) &&
    (qtd_colunas  > 0) && (qtd_linhas  > 0)) {
    // fazer cálculo do NDC
    double tmpLargura = ((double) (Largura_Prancha_original *
                         qtd_colunas) / (LarguraPanel));
    double tmpAltura  = ((double) (Altura_Prancha_original *
                         qtd_linhas) / (AlturaPanel));
    // pegar posicao do mouse no mapa
    Point point = new Point(mouseevent.getX(),
                            Mouseevent.getY());
    int pos_x = (int) (point.x * tmpLargura);
    int pos_y = (int) (point.y * tmpAltura);
    // calcular deslocamento, com relacao ao Zoom atual
    int deslocamento_x = (menor_col – col_inicial) *
                         Largura_Prancha_original;
    int deslocamento_y = (menor_lin – lin_inicial) *
                             Altura_Prancha_original;
    // exibir posicao na tela.
    FrameRoot.ExibirPosicaoCur(
           String.valueOf(pos_x + deslocamento_x),
           String.valueOf(pos_y + deslocamento_y));
}
```

O protótipo permite ao usuário interagir através da barra de ferramentas. As 4 funções implementadas, foram:

a) *ZoomIn*;

b) ZoomOut;

c) Restaurar e

d) *Pan*.

No quadro 12 pode-se verificar a rotina de *ZoomIn*.

Quadro 12 - ZoomIn

```
void ZoomIn() {

   // armazenar em array as maiores e menor linhas e
   //colunas para utilizar para ZoomOut
   qtdZoom++;
   Zoom tmpZoom;
   tmpZoom = new Zoom();
   tmpZoom.menor_lin = menor_lin;
   tmpZoom.maior_lin = maior_lin;
   tmpZoom.menor_col = menor_col;
   tmpZoom.maior_col = maior_col;
   ListaZoom[qtdZoom] = tmpZoom;

   // encontrar em qual linhas e coluna foi selecionado o
   // ZoomIn
   int linhaP1  = ((int)GPp1.getY() / AlturaPrancha) + 1;
   int colunaP1 = ((int)GPp1.getX() / LarguraPrancha) + 1;
   int linhaP2  = ((int)GPp2.getY() / AlturaPrancha) + 1;
   int colunaP2 = ((int)GPp2.getX() / LarguraPrancha) + 1;

   // encontrar qual as maiores colunas e linhas e menores
   // colunas e linhas vao ser exibidas
   if (linhaP1 < linhaP2) {
      menor_lin = (menor_lin + linhaP1) - 1;
      maior_lin = menor_lin + (linhaP2 - linhaP1);
   }
   else {
      menor_lin = (menor_lin + linhaP2) - 1;
      maior_lin = menor_lin + (linhaP1 - linhaP2);
   }

   if (colunaP1 < colunaP2) {
      menor_col = (menor_col + colunaP1) - 1;
      maior_col = menor_col + (colunaP2 - colunaP1);
   }
   else {
      menor_col = (menor_col + colunaP2) - 1;
      maior_col = menor_col + (colunaP1 - colunaP2);
   }
   return;
}
```

A cada *ZoomIn* que ocorrer é armazenado as maiores e menores colunas e linhas que estão exibidas no momento, conforme já exibido no quadro 11. Para efetuar o *ZoomOut*, basta retornar o Zoom que foi efetuado por último, ou seja, as últimas informações armazenadas na lista de Zoom, conforme demonstrado a implementação no quadro 13.

Quadro 13 - ZoomOut

```
Void ZoomOut() {
    if (qtdZoom > -1) {
       // pegar as pranchas que estavam sendo exibidas no último Zoom
       menor_lin = ListaZoom[qtdZoom].menor_lin;
```

```
            maior_lin = ListaZoom[qtdZoom].maior_lin;
            menor_col = ListaZoom[qtdZoom].menor_col;
            maior_col = ListaZoom[qtdZoom].maior_col;
            // diminuir para um Zoom anterior
            qtdZoom--;
        }
        return;
    }
```

Para implementar o restaurar, basta exibir as pranchas que foram exibidas no início da aplicação.

Para finalizar, para implementação do *Pan*, basta encontrar quantas pranchas foram deslocadas para cima, para baixo ou para os lados e somar as atuais, conforme mostra o quadro 14.

Quadro 14 - Pan

```
    void Pan() {
       // verificar quantas pranchas foram selecionadas para serem
       // deslocadas.
       int deslocX = ((int)GPp2.getX() - (int)GPp1.getX());
       int deslocY = ((int)GPp2.getY() - (int)GPp1.getY());
       int qtd_colunas = deslocX / LarguraPrancha;
       int qtd_linhas  = deslocY / AlturaPrancha;
       menor_col = menor_col + qtd_colunas;
       maior_col = maior_col + qtd_colunas;
       menor_lin = menor_lin + qtd_linhas;
       maior_lin = maior_lin + qtd_linhas;
       return;
    }
```

5.3 FUNCIONAMENTO DO PROTÓTIPO

Para o funcionamento deste protótipo desenvolvido são necessários os seguintes softwares:

a) Tomcat: para funcionar como um servidor de página web;
b) Java: para instalação da linguagem Java basta a instalação do pacote JDK, que possui o compilador Java e biblioteca de classes necessário para desenvolvimento dos *Applets* e demais rotinas. A versão do JDK utilizado foi o JDK 1.3.1;
c) Browser: para a execução da aplicação no cliente é necessário um navegador de web (*browser*) que seja compatível com Java, como Internet Explorer ou Netscape;
d) Banco de Dados ORACLE 8i: o ORACLE 8i é utilizado para o armazenamento das informações;
e) Java Plug-in version 1.3: para funcionamento dos componentes *Swing* do Java.

Tendo a preocupação com as ferramentas anteriormente mensionadas, segue-se para o protótipo em si, o qual é composto por duas partes:

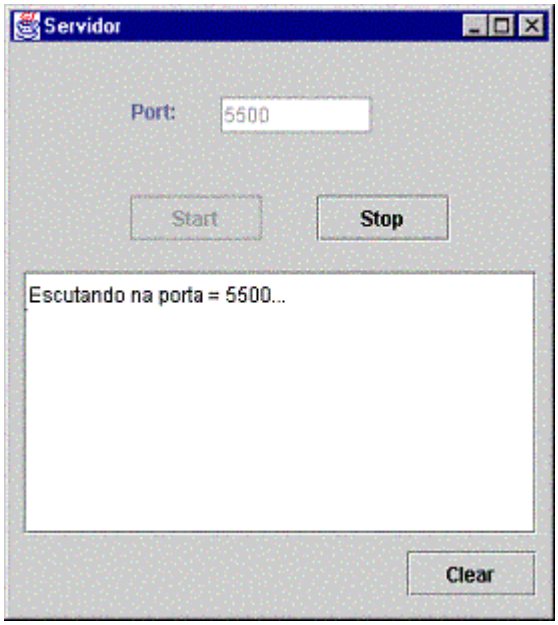
a) Software Servidor: possui a tarefa de ficar “sempre” aguardando requisições dos softwares clientes. O software cliente envia uma requisição ao software servidor de que necessita de uma determinada informação, que geralmente é proveniente do banco de dados, e após é respondido esta mesma requisição;

b) Software Cliente: é um *Applet* que é carregado pelo *browser* do usuário, e através dele é possível se estabelecer a interação do usuário com o protótipo. Este *Applet* possui uma interface gráfica amigável para que o usuário possa facilmente executar suas consulta sobre dados espaciais. Para obter os dados necessários ao funcionamento, comunica-se com o software servidor através de *sockets.*

5.3.1 SOFTWARE SERVIDOR

Na Figura 26 está demonstrado a interface do Software Servidor.

Figura 26 – Imagem do Software Servidor



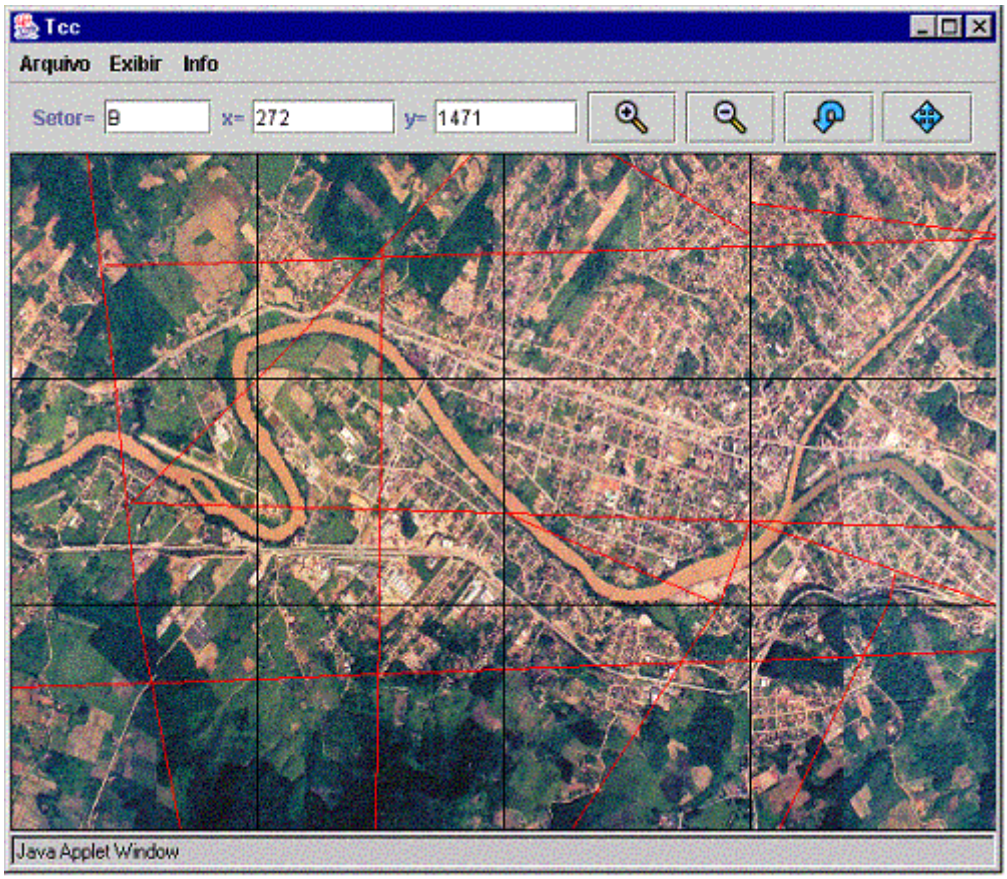
O software servidor deve ser executado antes que o software cliente, pois ele é responsável por ficar “escutadando” as requisições do software cliente.

No software servido há uma caixa de texto que permite informar qual porta ele ficará escutando, que deve ser a mesma utilizada pelo software cliente. Nele ainda há o botão Start que permite iniciar o processo principal dele, já mensionado acima; um botão Stop para parar o processo e por último o botão Clear para limpar a caixa de texto que exibe o log de execução.

5.3.2 SOFTWARE CLIENTE

A figura 27 mostra a interface do software Cliente.

Figura 27 – Imagem do Software Cliente



Com relação às informações geográficas o software cliente permite visualizar informações *Raster* e informações vetoriais. A informação *Raster* é o *background* da aplicação, que é representado com o mapa de uma cidade (no caso, a cidade de Rio do Sul). A informação vetorial pode ser representado através dos segmentos de retas que formam setores, que podem representar por exemplo uma área ou um bairro da cidade.

O usuário terá a possibilidade ficar exibindo e ocultando estas informações pelo menu "Exibir" da aplicação. Este menu possui as seguintes opções: "*Background* (Raster)", "Vetorial", "Grade Pranchas" e "Nome de Setores".

Pode-se resumir a visualização das informações *Raster* e Vetoriais no protótipo como:

a) Pranchas;
b) Nível *Raster* (*background*);
c) Nível Vetorial;

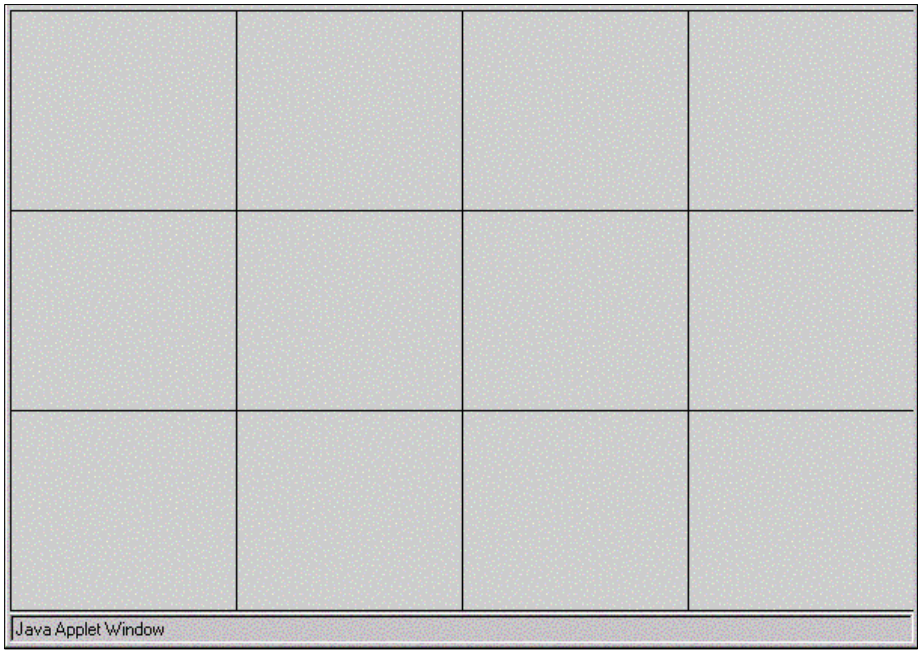
d) Descrição dos Setores e

e) Coordenadas.

A imagem final exibida no protótipo como *background* é constituída de várias imagens menores de uma cidade. E cada imagem desta pode ser definida como uma prancha. As operações que são efetuadas no protótipo pelo usuário, levam em consideração as pranchas que atualmente estão sendo exibidas. Como por exemplo as operações de *ZoomIn*, *Pan*, exibição de setores, entre outras.

Com opção de “Grade Pranchas” do menu exibir selecionada é possível identificar as pranchas que estão sendo exibidas atualmente, conforme demonstrado na Figura 28.

Figura 28 – Pranchas

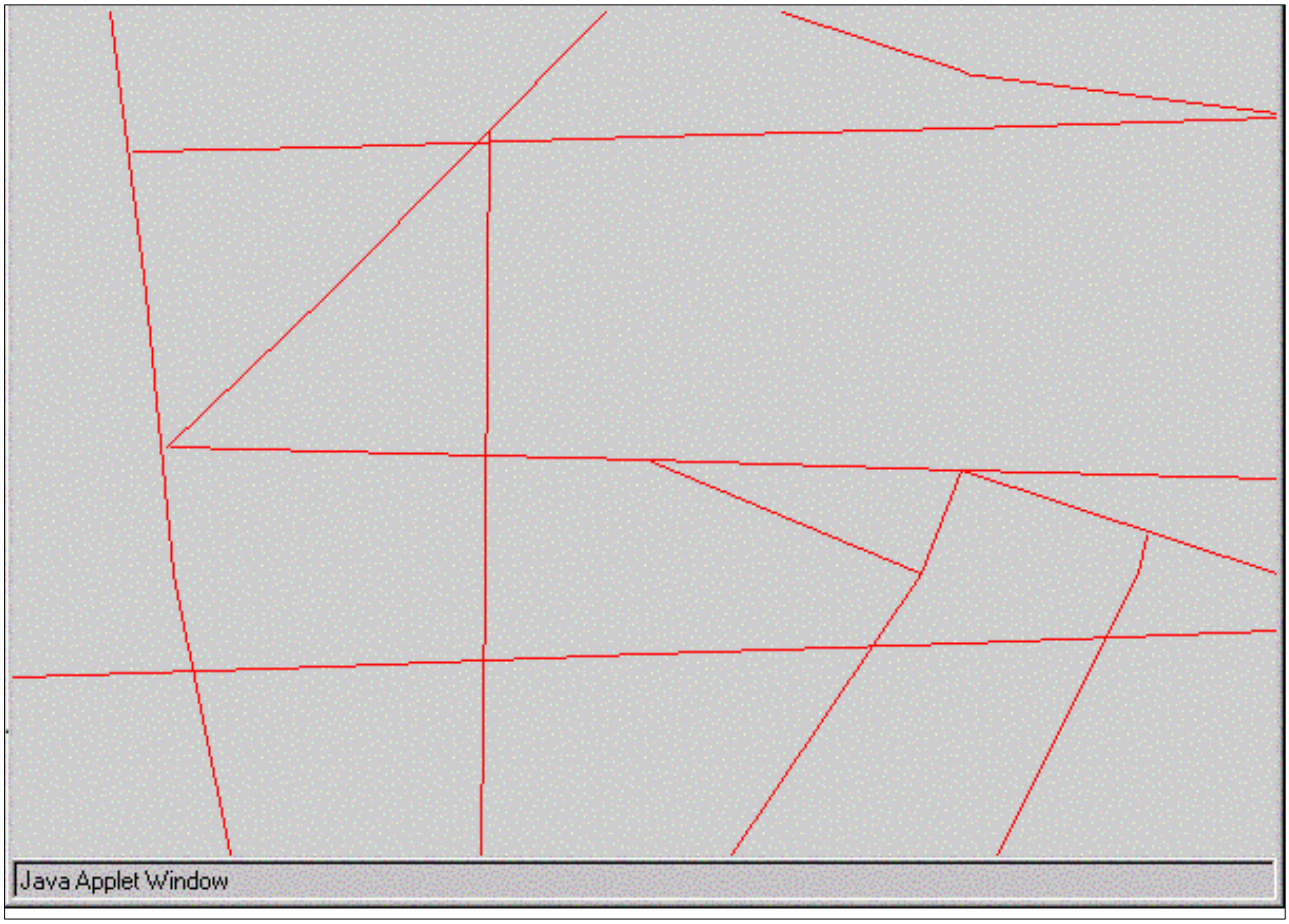


Como descrito anteriormente, o *background* é a exibição das pranchas do mapa, e cada prancha é identificada por uma figura. E unindo-se estas imagens (pranchas) forma-se o mapa, conforme mostra a figura 29.

Figura 29 - Nível *Raster* (*background*)

Através do conjunto de segmentos de retas consegue-se representar a informação vetorial, conforme figura 30.

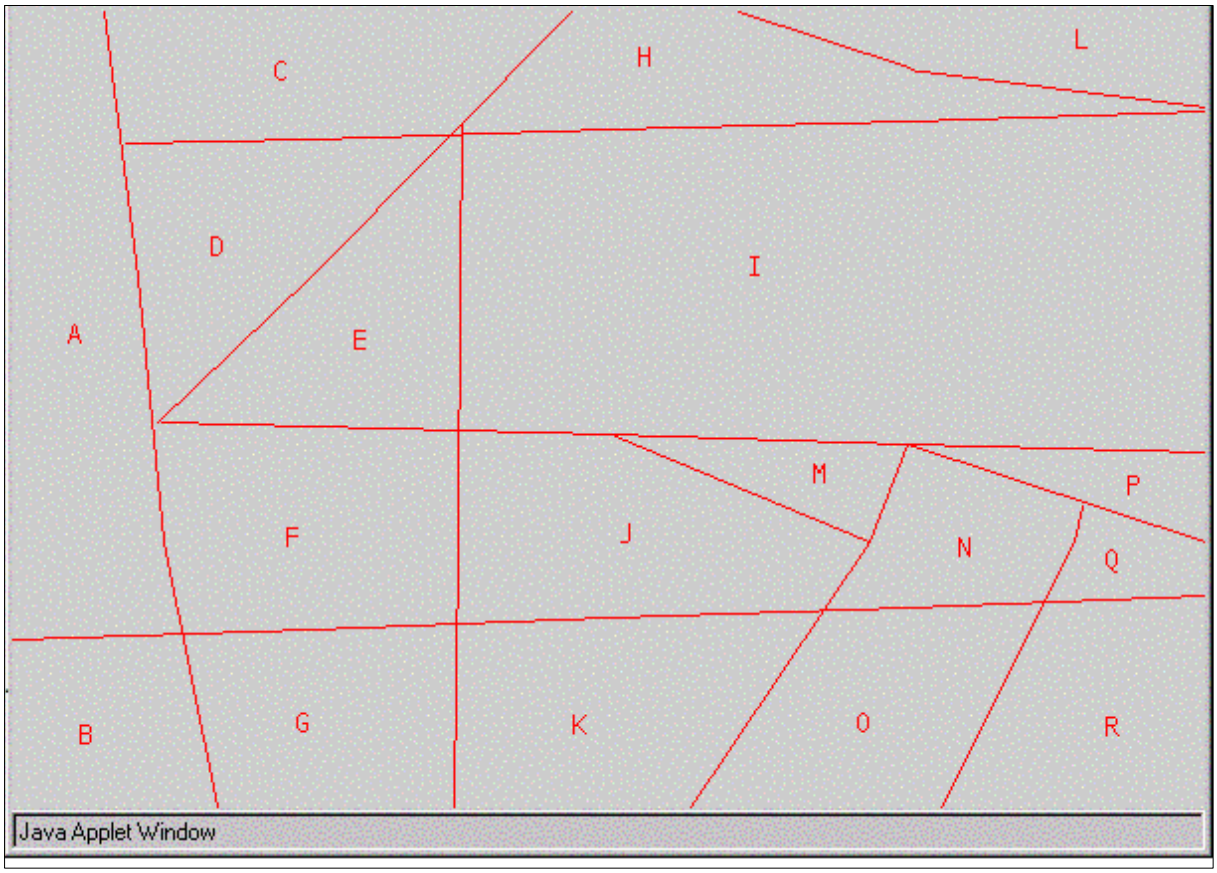
Figura 30 - Nível Vetorial



Os setores, que representam informações vetoriais, podem ser identificados de duas formas neste protótipo.

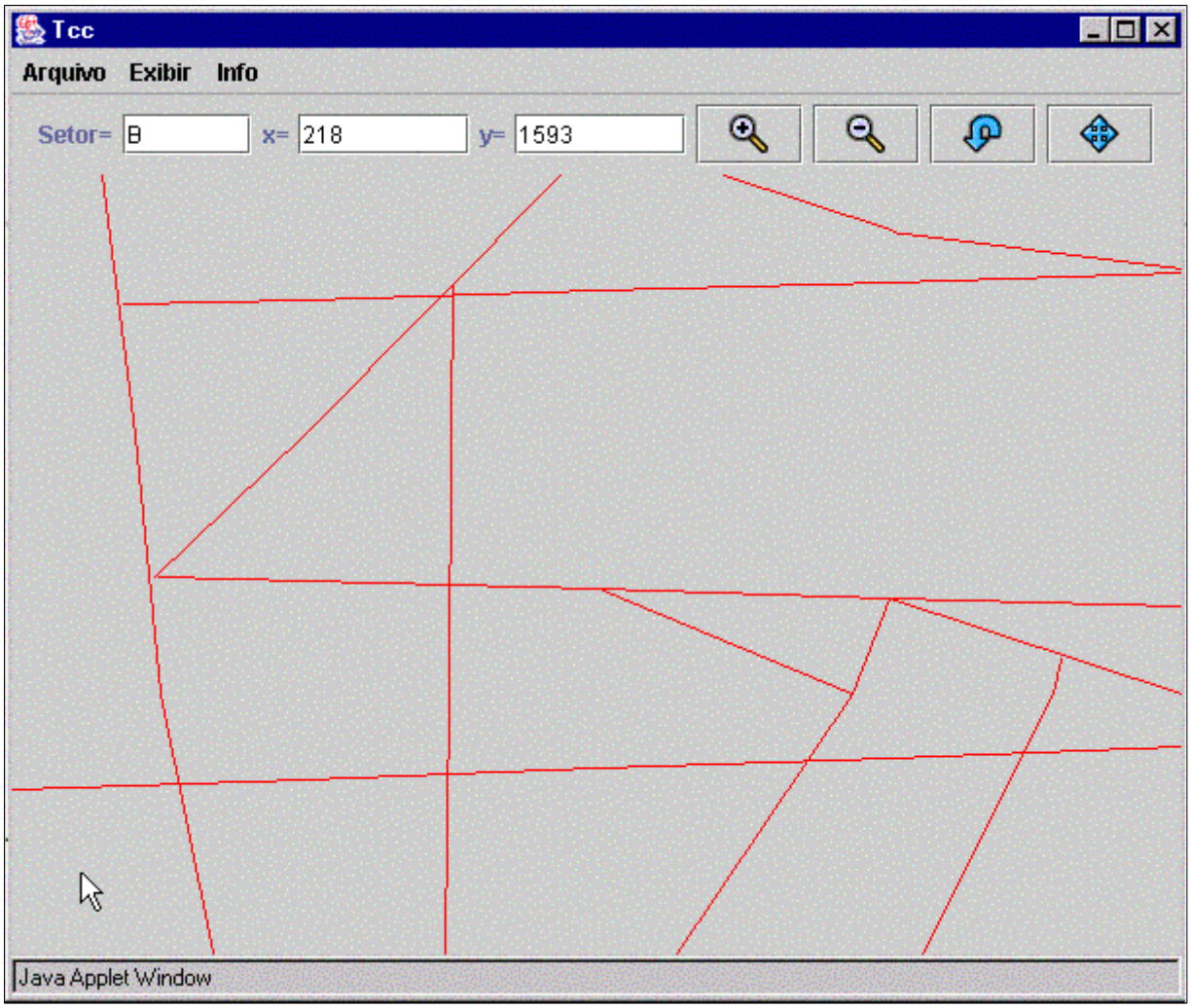
A primeira, seria o usuário selecionar no menu “Exibir” a opção de “Nome de Setores”. Desta forma será feita uma pesquisa na base, o nome cadastrado para cada setor e estes serão exibidos no mapa, conforme demonstra figura 31.

Figura 31 - Descrição dos Setores



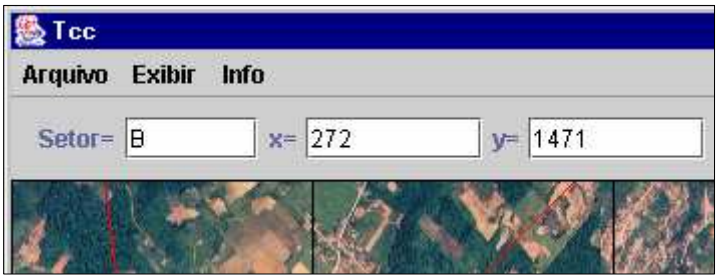
A segunda, é quando o usuário der um "*click*" dentro do mapa, e será feita uma pesquisa para identificar em qual setor pertence o ponto. O nome deste setor será exibido na caixa de texto que se encontra acima do mapa no protótipo com o identificação de "Setor", conforme figura 32.

Figura 32 - Descrição dos Setores



As coordenadas é um dado muito importante quando se trabalha com informações espaciais. No protótipo esta informação é visualidade em duas caixa de texto que se encontram acima do mapa principal, conforme mostra figura 33.

Figura 33 - Coordenadas do mapa



Estes dois valores representam a quantos metros se encontra a posição do ponteiro do *mouse* no mapa em relação ao mapa todo. No protótipo foi definido que cada prancha possuía 800 metros de largura e 600 metros de altura. Caso se aplique um *ZoomIn* neste mapa a partir da segunda prancha (em relação a largura), então a posição se inicia em 800. Desta forma pode-se, por exemplo, identificar a distância entre dois objetos no mapa em unidade de metros.

O protótipo permite ao usuário interagir através da barra de ferramentas. Esta permite 4 operações, como demonstra a figura 34:

Figura 34 – Barra de Ferramentas

a) *ZoomIn*;

a) *ZoomOut*;

b) Restaurar e

c) *Pan*.

Para fazer o *ZoomIn*, o usuário deverá clicar no mapa para selecionar o início da área de seleção. Enquanto que não for liberado o botão do *mouse* será exibido um retângulo da área atualmente selecionada. Após liberado o botão do *mouse* e clicado no botão de *ZoomIn* (Figura 35), será exibido as pranchas que abrangem a área selecionada pelo usuário, conforme demonstrado na figura 36.

Figura 35– Seleção de área para *ZoomIn*

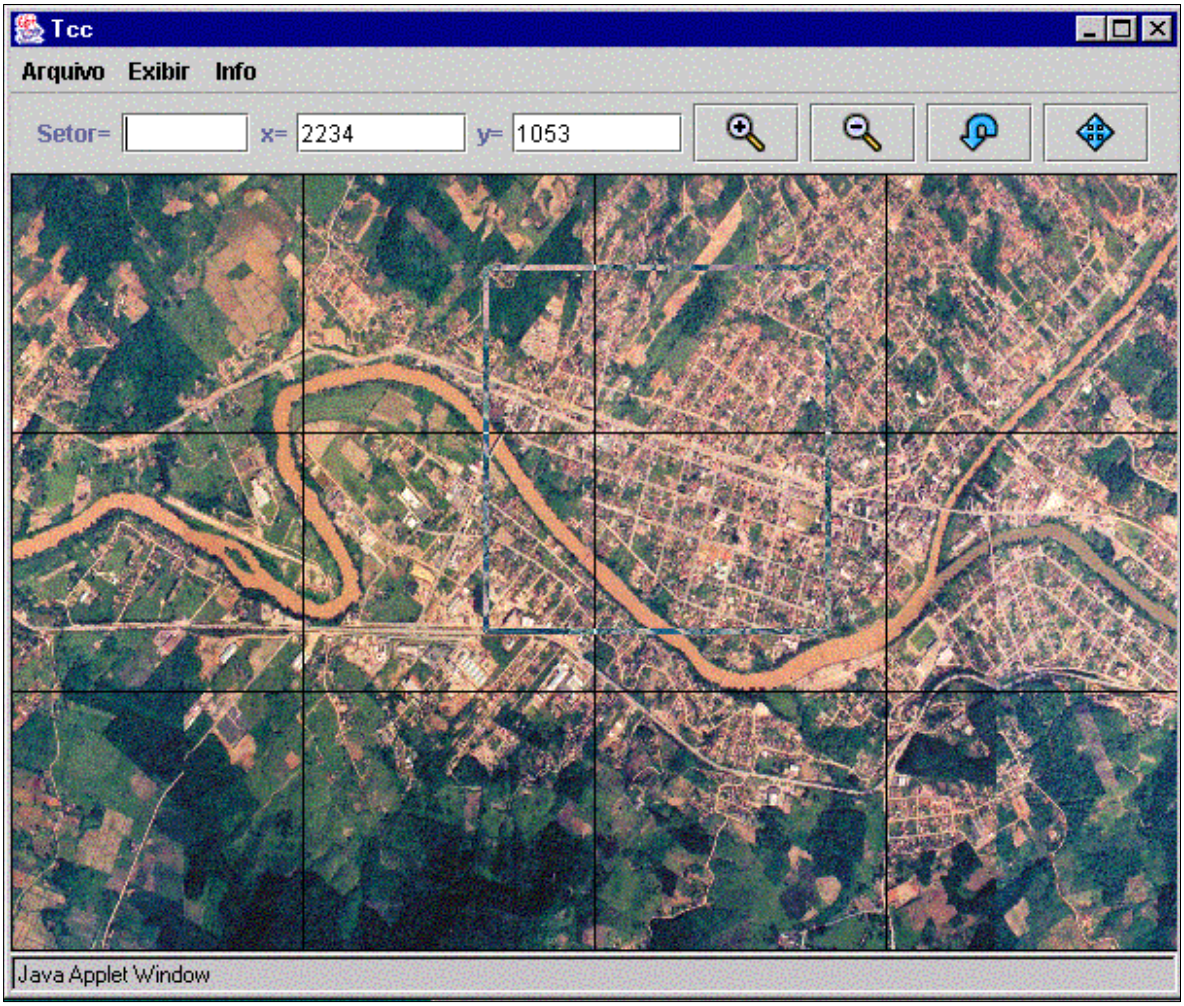
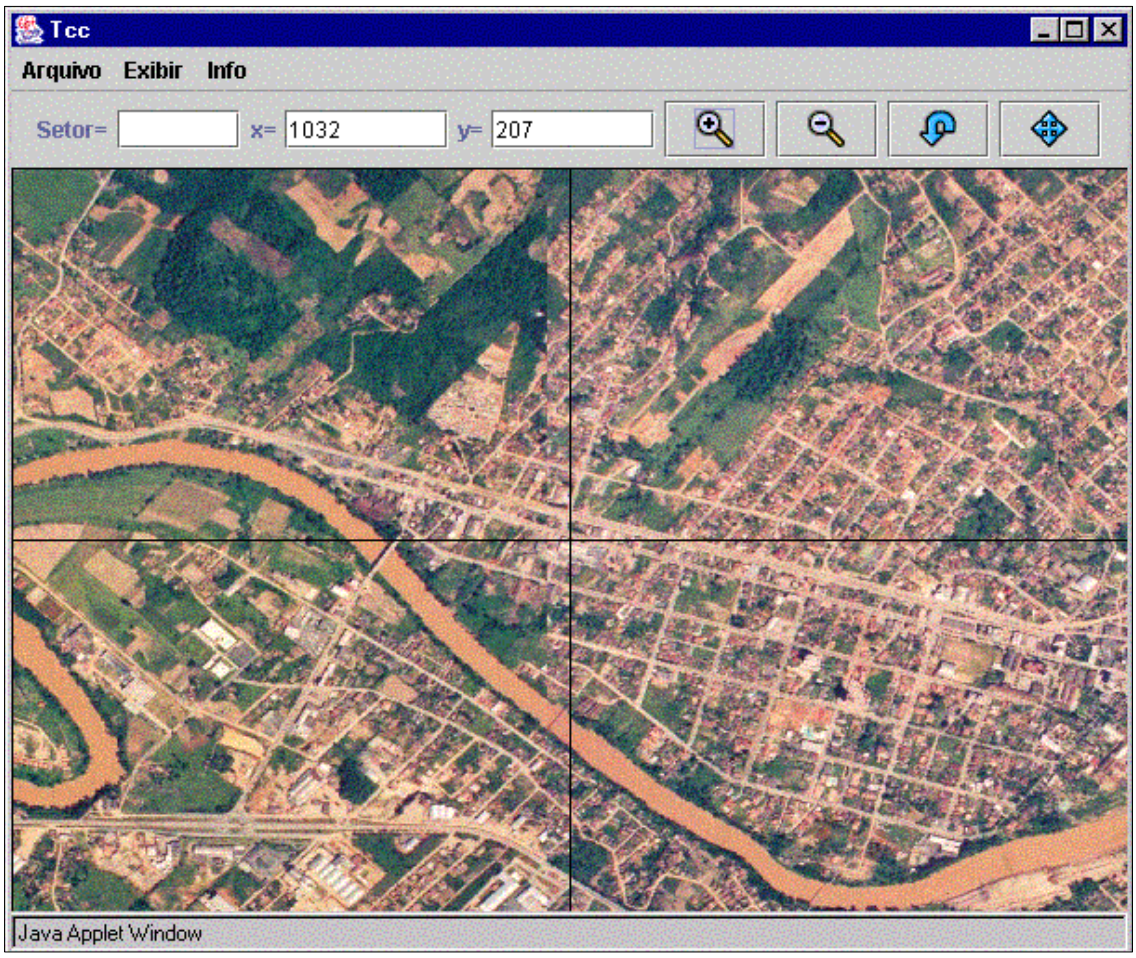


Figura 36 – Área selecionada após *ZoomIn*



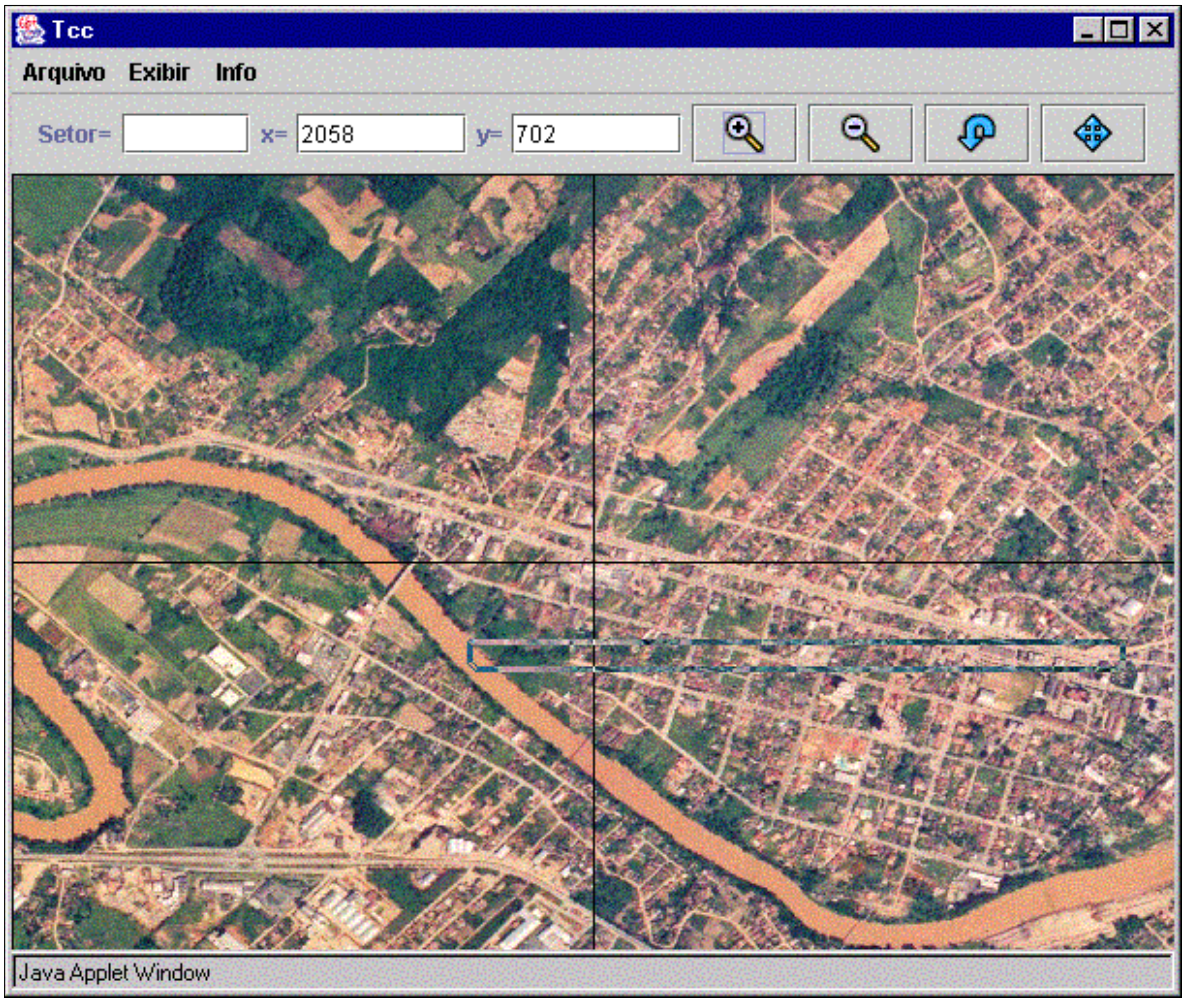
Quando o usuário clicar no botão de *ZoomOut*, será restaurado o último *Zoom* exibido, ou seja, quando for feito a operação de *ZoomIn*, será armazenada em uma lista as pranchas que foram exibidas e quando ocorrer o *ZoomOut* será mostra o último *Zoom* da lista.

Para restaurar para a exibição inicial, basta cliclar no botão de Restaurar.

O botão de *Pan* permite deslocar o mapa para os lados, para cima o para baixo, com relação as pranchas que estão sendo exibidas atualmente. Mas para ocorrer um *Pan*, já deve ter ocorrido um *ZoomIn*, pois em caso contrário não haverá o que deslocar.

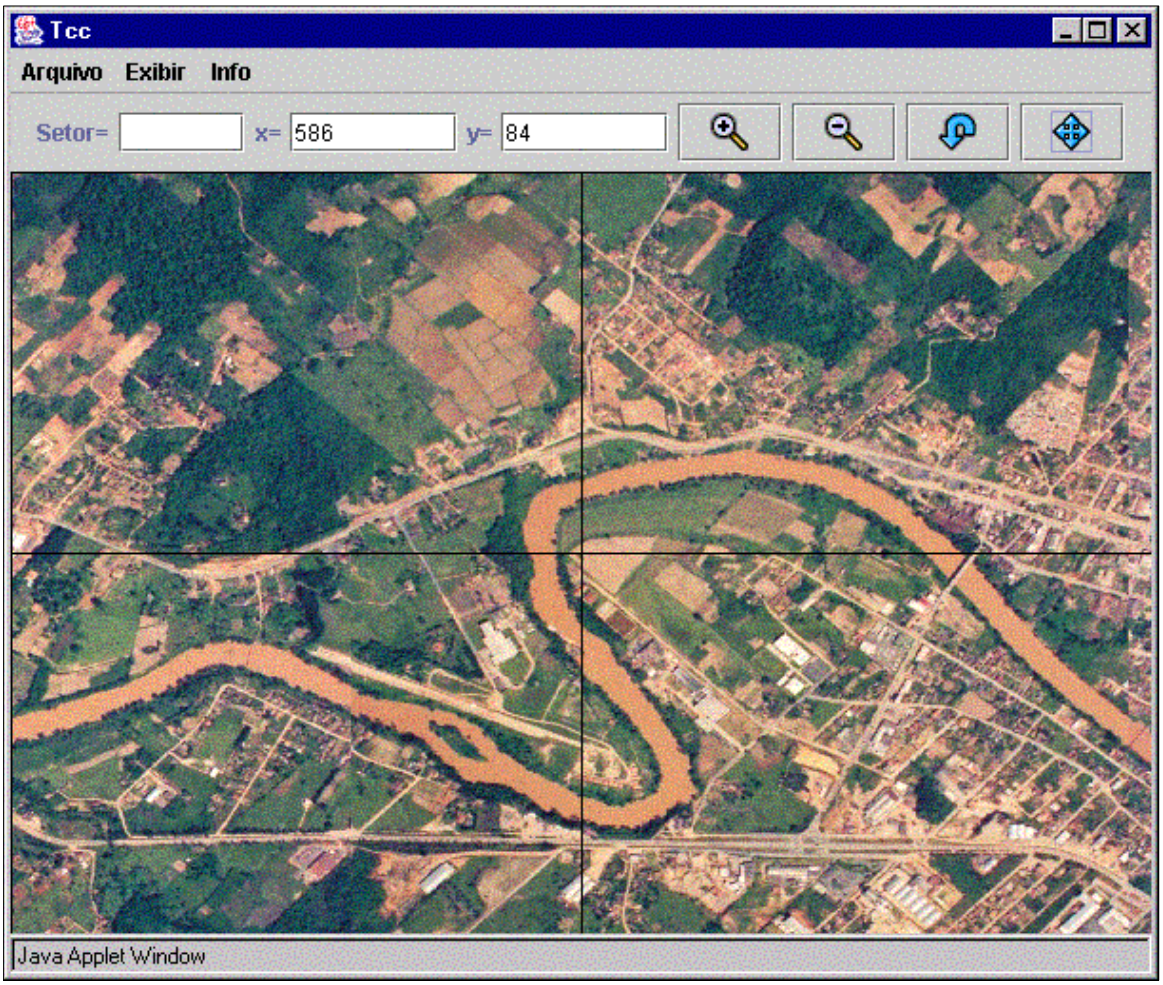
O usuário pode clicar para definir o ponto inicial, e sem liberar o botão do mouse move para alguma direção (esquerda, direita, cima, baixo) indicando para que direção deseja deslocar a visualização atual, conforme demonstrado na figura 37.

Figura 37 – Seleção de área para *Pan*



A figura 38 mostra como ficou a visualização após o *Pan* efetuado pelo usuário

Figura 38 – Área selecionada após *Pan*



6 CONCLUSÕES

O objetivo proposto do trabalho foi atendido com sucesso, pois o protótipo permite que seja trabalhado com informações espaciais via web, acessando banco de dados *ORACLE*. Este permite também efetuar operações como *ZoomIn, ZoomOut, Pan,* entre outras. Funcões estas existentes em ferramentas de geoprocessamento.

As informações espaciais foram apresentadas de duas formas: informações *Raster* e informações vetoriais. A informação *Raster* é o *background* da aplicação, que é representado como o mapa da cidade, e a informação vetorial pode ser representado através dos segmentos de retas que formam setores.

Observou-se que poderiam ser utilizadas diversas camadas (níveis) para representar várias informações vetoriais, além das utilizadas neste trabalho, como por exemplo: rios, estradas, ferroviais, estabelecimentos, vegetação, entre outras.

Observou-se também neste trabalho que quando se pretende trabalhar com imagens, deve-se levar em consideração o tamanho que estas imagens possuem. Pois elas podem interferir na utilização da aplicação pelo usuário, deixando esta muito “pesada”. Isto deve ser levado em consideração principalmente em ferramentas utilizadas via web, pois será feito o *download* das imagens para a máquina cliente, antes destas serem utilizadas.

Para o desenvolvimento do protótipo optou-se pela implementação das rotinas vetoriais de geoprocessamento em Java, por desta forma enriquecer o conteúdo do trabalho, apesar destas rotinas estarem implementadas no banco de dados ORACLE, já neste trabalho conceituado como *ORACLE Spatial*.

As ferramentas utilizadas foram adequadas para a obtenção do resultado, citando como por exemplo a Linguagem Java e o banco de dados ORACLE. A ferramenta utilizada para auxiliar no desenvolvimento das rotinas em Java foi a JDeveloper da ORACLE. Esta ferramenta satisfez as necessidades apesar desta ser uma ferramenta que exige bastante da máquina utilizada para o desenvolvimento.

6.1 EXTENSÕES

Sugere-se como extensão deste traballho, que as imagens dos mapas utilizadas pela aplicacão sejam armazenadas também no banco de dados *ORACLE*, assim como as demais informações utilizadas.

Sugere-se também que o protótipo permita informar endereços, e que seja retornado o menor trajeto entre estes endereços. E para este trajeto seja exibido demais informações, como: a quilometragem para percorrer, o custo de taxi para este trajeto, pontos comerciais e ainda um menor trajeto levando em consideração o sentido das ruas e congestionamentos dependendo do horário.

Seria interessante que a exibição de nomes dos setores fossem permitidos também para qualquer visualização (Zoom) feita pelo usuário.

Sugere-se ainda que seja possível visualizar diversas informações vetoriais além dos setores, que foi a utilizada neste trabalho. Poderiam ser informações como: tubulações de água e esgoto que se encontrar no subterrâneo, ferrovias da cidade, diversos estabelecimentos (farmácias, supermercados, hospitais), entre outros.

REFERÊNCIAS BIBLIOGRÁFICAS

BORGES, Karla. **Modelagem de dados geográficos: uma extensão do modelo OMT para aplicações geográficas**. 1997. 128 f. Dissertação de Mestrado, Fundação João Pinheiro, Belo Horizonte.

CÂMARA, Gilberto et al. **Anatomia de sistemas de informação geográfica :** trabalho para escola de computação. Campinas, Unicamp, 1996.

CÂMARA, Gilberto. **Interoperabilidade em geoprocessamento**: conversão entre modelos conceituais de sistemas de informação geográfica e comparação com o padrão open gis, São José dos Campos, dez. 2000. Disponível em: <www.dpi.inpe.br/gilberto/infogeo/infogeo12.pdf >. Acesso em: 05 set 2001.

CÂMARA, Gilberto. **Arquiteturas Cliente-Servidor para Bibliotecas Geográficas Digitais** , São José dos Campos, mar. [2001?]. Disponível em: <www.dpi.inpe.br/geopro/trabalhos/cliente_servidor.pdf>. Acesso em: 06 jul 2002.

CAMARGO, Marcos Ubirajara de Carvalho e. **Sistemas de informações geográficas como instrumento de gestão e saneamento**. Rio de Janeiro, Abes, 1997.

DAVIS, Clodoveu. **Modelagem Semântica em Geoprocessamento,** São José dos Campos, jun. 2001. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/bdados/cap2-modelagem.pdf>. Acesso em: 02 nov 2001.

FIELDS, Duane. **Desenvolvendo na Web com Java Server Pages**. Rio de Janeiro, Ciência Moderna, 2000.

GATTASS, Marcelo. **Servlets e COM para a Visualização de Dados Geográficos na Web**, Rio de Janeiro, jun. 2000. Disponível em: <www.dpi.inpe.br/nsf-cnpq/servlets.pdf>. Acesso em: 06 jul 2002.

GOODWILL, J. **Developing Java Servlets:** The Authoritative Solution. Sams Publishing, 1999.

HOFF, Arthur Van, **Ligado em Java**. São Paulo, Makron Books, 1997.

JACOB, Alberto Augusto Eichman. **Sistemas de informações geográficas**, Campinas, dez. 1999. Disponível em: <http://www.unicamp.br/nepo/alberto/sig.htm#aplicaões>. Acesso em: 21 jun 2001.

LEAL, Zé Paulo.**Descrição do algoritmo de dijkstra**, Porto, dez. [2001?]. Disponível em : <http://www.ncc.up.pt/~zp/aulas/9900/me/trabalhos/alunos/Algoritmos_do_Caminho_Minimo_em_Grafos/ddijk.html>. Acesso em: 14 jun 2001.

MADEIRA, Henrique. **Tópicos avançados de bases de dados**, [S.l], out .1999. Disponível em: <http://eden.dei.uc.pt/~tabd/plano.html>. Acesso em: 21 jun 2001.

MEDEIROS, José Simeão de. **Bancos de Dados Geográficos e Redes Neurais Artificiais**: Tecnologias de Apoio à Gestão de Território. 1999. 217 f. Tese de Doutorado (Doutorado em Geografia), USP, São Paulo.

MENEGUETE, Arlete. **Sistemas de Informação Geográfica como uma Tecnologia Integradora**: Contexto, Conceitos e Definições, Presidente Prudente, out. 1998. Disponível em: <http://www.prudente.unesp.br/dcartog/arlete/gis/intro_t.htm>. Acesso em: 15 set 2001.

NETTO, Paulo Oswaldo Boaventura. **Teoria e modelos de grafos**. São Paulo, Blücher , 1979.

ORACLE CORPORATION. **Oracle8i Spatial :** Features Overview. Redwood Shores : Oracle, 1998.

ORACLE UNIVERSITY. **Introdução ao oracle: sql e pl/sql**. Jerry Brosnan: Oracle, 1999.

ORACLE UNIVERSITY. **Oracle 8i: desenvolvimento de unidades de programa em pl/sql**. Jerry Brosnan: Oracle, 2000.

SALGADOS, Pedro Emanuel de Castro Faria. **Algoritmos e estruturas de dados II**, [S.l], dez. 2001. Disponível em: <http://tom.fe.up.pt/~ei99006/cadeiras/aed2/mini-site/graphs/text.html#topico1>. Acesso em: 14 jun 2001.

SOUZA, Evandro de. **Protótipo de um sistema am/fm para o acompanhamento das cotas enchentes de blumenau utilizando internet.** 2000. 58 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

THOME, Rogério. **Interoperabilidade em geoprocessamento**: conversão entre modelos conceituais de sistemas de informação geográfica e comparação com o padrão open gis, São José dos Campos, set. 1998. Disponível em: < http://www.dpi.inpe.br/teses/thome/>. Acesso em: 05 set 2001.

ANEXO 1

```
// rotina da classe ServerRequest que retorna nome do setor
// conforme posicao x,y passado como parametro
if (operacao.equalsIgnoreCase("LeSetor"))
{
   String aux = (String) ois.readObject();
   float X = Integer.valueOf(aux).intValue();
   aux = (String) ois.readObject();
   float Y = Integer.valueOf(aux).intValue();
   String Nome_Setor = " ";
   Statement sql = null;
   Statement sql1 = null;
   ResultSet rs = null;
   ResultSet rs1 = null;
   boolean naoTRI = true;
   boolean temPto;

   sql = conexao.createStatement();
   rs = sql.executeQuery (
      "select s.cd_setor, s.nm_setor, count(*) qtd_vertices" +
      "  from setor_segmento ss, setor s " +
      " where ss.cd_setor = s.cd_setor " +
      "   and cd_mapa = " + mapa +
      " group by s.cd_setor, s.nm_setor");

   while (rs.next()) {
      temPto = false;
     // retornar vertices do setor
     sql1 = conexao.createStatement();
     rs1 = sql1.executeQuery ("select x, y " +
                      "from setor_ponto sp " +
                      "where cd_setor = " + rs.getInt(1) +
                      "  and cd_mapa = 1" +
                      " order by sp.cd_sequencia");

     // armazenar vertices em uma lista
     while (rs1.next()) {
         Ponto tmpVertice;
         tmpVertice = new Ponto();
         tmpVertice.x = rs1.getInt(1);
         tmpVertice.y = rs1.getInt(2);
         ListaVertices.add(tmpVertice);
         temPto = true;
     }
     // fechar result set e statement
     rs1.close();
```

```
     sql1.close();

     if (temPto) {
       // Gerar Triangulos dos polígonos ( setores )
       int qtd_vertices = rs.getInt(3);
       Triangulos tmpTriangulos;
       tmpTriangulos = new Triangulos();
       tmpTriangulos.vertice1 = 1;
       tmpTriangulos.vertice2 = 2;
       tmpTriangulos.vertice3 = 3;
       ListaTriangulos.add(tmpTriangulos);
       for (int cont = 3; cont < qtd_vertices; cont++) {
          tmpTriangulos = new Triangulos();
          tmpTriangulos.vertice1 = 1;
          tmpTriangulos.vertice2 = cont;
          tmpTriangulos.vertice3 = cont + 1;
          ListaTriangulos.add(tmpTriangulos);
       }

       Ponto ptoA = new Ponto();
       Ponto ptoB = new Ponto();
       Ponto ptoC = new Ponto();
       Ponto ptoAB = new Ponto();
       Ponto ptoAC = new Ponto();
       Ponto ptoAT = new Ponto();
       Ponto tmpVertice = new Ponto();
       Triangulos tmpTriangulo = new Triangulos();
       float[][] mAUX;
       float[][] mI;
       mAUX = new float[2][2];
       mI   = new float[2][2];
       float ter;
       float S, T;
       for (int i = 0; i < ListaTriangulos.size(); i++)
       { // para cada triangulo
         if (naoTRI) {
           tmpTriangulo = (Triangulos) ListaTriangulos.get(i);
           tmpVertice = (Ponto)
ListaVertices.elementAt(tmpTriangulo.vertice1 - 1);
           ptoA   = tmpVertice;
           tmpVertice = (Ponto)
ListaVertices.elementAt(tmpTriangulo.vertice2 - 1);
           ptoB   = tmpVertice;
           tmpVertice = (Ponto)
ListaVertices.elementAt(tmpTriangulo.vertice3 - 1);
           ptoC   = tmpVertice;

           // GerMatTRI
           ptoAB.x = ptoB.x - ptoA.x;  // SubVector2D(PB, PA)
```

```
            ptoAB.y = ptoB.y - ptoA.y;
            ptoAC.x = ptoC.x - ptoA.x;  // SubVector2D(PC, PA)
            ptoAC.y = ptoC.y - ptoA.y;
            mAUX[0][0] = ptoAB.x;
            mAUX[0][1] = ptoAB.y;
            mAUX[1][0] = ptoAC.x;
            mAUX[1][1] = ptoAC.y;
            ter = ((float) .1) / ((mAUX[0][0] * mAUX[1][1]) -
(mAUX[1][0] * mAUX[0][1]));
            mI[0][0] = ter *   mAUX[1][1];
            mI[0][1] = ter * - mAUX[0][1];
            mI[1][0] = ter * - mAUX[1][0];
            mI[1][1] = ter *   mAUX[0][0];
            // PesPtoTRI
            ptoAT.x = (float) X - ptoA.x; //SubVector2D(PT, PA)
            ptoAT.y = (float) Y - ptoA.y;
            S = (ptoAT.x * mI[0][0]) + (ptoAT.y * mI[1][0]);
           T = (ptoAT.x * mI[0][1]) + (ptoAT.y * mI[1][1]);

           if (!((S < 0) || (T < 0) || ((S + T) > 1))) {
              naoTRI = false;  // dentro do triangulo
              Nome_Setor = rs.getString(2);
           }
           else
              naoTRI = true;  // fora do triangulo
           } // if (naoTRI)
        } // for
      } // if (temPto)
   }

   // enviar dados lidos
   ObjectOutputStream oos = new ObjectOutputStream(new
BufferedOutputStream(this.socket.getOutputStream()));
   oos.writeObject(Nome_Setor);
   oos.flush();
   oos.close();

   // fechar result set e statement
   rs.close();
   sql.close();
}
```