UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM SISTEMA AM/FM PARA O ACOMPANHAMENTO DAS COTAS ENCHENTES DE BLUMENAU UTILIZANDO 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

EVANDRO DE SOUZA

BLUMENAU, DEZEMBRO/2000

2000/2-24

PROTÓTIPO DE UM SISTEMA AM/FM PARA O ACOMPANHAMENTO DAS COTAS ENCHENTES DE BLUMENAU UTILIZANDO INTERNET

EVANDRO DE SOUZA

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. Paulo C. Rodacki Gomes

_____

Prof. Roberto Heinzle

DEDICATÓRIA

“Escale a montanha dos seus ideais e a conquiste.”

Dedico este trabalho aos meus pais e meu irmão que sempre me apoiaram e me deram forças para terminar essa escalada, principalmente nos momentos em que olhava para baixo e sentia vontade de desistir.

AGRADECIMENTOS

Em especial para meu orientador e amigo Dalton Solano dos Reis, que passou tanto tempo comigo e soube me apoiar nas horas mais difíceis.

E também a meus colegas de faculdade e amigos, pelo companheirismo e bons momentos vividos.

Por fim, agradeço a todos aqueles que contribuíram para a conclusão desse trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Este trabalho de conclusão de curso visa realizar um estudo sobre sistemas *AM/FM - Automated Mapping/Facilities Management*, sendo os conceitos adquiridos utilizados para especificar e implementar um protótipo voltado a Internet utilizando *Applets* e *Servlets*, o qual se baseia no Sistema de Cotas Enchentes de Blumenau.

ABSTRACT

This Concluding Course Work presents a study on AM/FM (Automated Mapping/Facilities Management) systems, being the concepts acquired used to specify and implement a prototype looking at Internet using Applets and Servlets, which is based on the System of Cotas Enchentes de Blumenau.

1 INTRODUÇÃO

O Geoprocessamento é o conjunto de técnicas computacionais relacionadas com a coleta, armazenamento e tratamento de informações espaciais ou geo-referenciadas, para serem utilizadas em sistemas específicos a cada aplicação que, de alguma forma, se utiliza do espaço físico geográfico [FAT1999].

Segundo [INT1998] o *AM/FM* (*Automated Mapping/Facilities Management*) ou Mapeamento Automatizado e Gerência de Serviços de Utilidade Pública é uma dessas técnicas do Geoprocessamento, e baseiam-se também em tecnologia *CADD (Computer-Aided Drafting and Design)*. Entretanto, a apresentação gráfica geralmente não é tão precisa e detalhada como em sistemas *CAM (Computer Aided Manufacturing)*; a ênfase de *AM/FM* está centrada no armazenamento, na análise e na emissão de relatórios.

Cada vez mais profissionais das mais diversas áreas vêm se atraindo pelo espaço virtual gerado pela Internet. Então, é de se esperar que se comesse a aplicar conceitos de Geoprocessamento a esse novo ambiente. Como exemplo dessa integração podemos citar o software Spring*Web* que disponibiliza dados do Spring (Sistema de Informação Geográfica) na Internet.

Existem junto ao Instituto de Pesquisas Ambientais aplicações (Sistema Cruzamento e a Carta Enchente) que mostram um uso comum para os sistemas de *AM/FM*. Ambos os sistemas atuam em casos que ocorre elevação nos níveis dos rios, como por exemplo em enchentes, além dos estragos óbvios que as cheias causam há ainda o problema do deslocamento, pois é comum a água cobrir algumas vias tornando-as inviáveis ao tráfego de veículos. Estas aplicações estão se apresentando eficazes quando da necessidade de rápida localização das regiões de emergência, bem como também a indicação do melhor caminho para que se chegue o mais rápido possível a estes locais.

Diante da importância de oferecer informações atualizadas rapidamente a comunidade tornou-se necessário uma nova atualização nos Sistema Cruzamento e Cartas Enchentes de Blumenau. E como a Internet tem se mostrado eficaz na difusão de informações

o sistema poderia ser portado para a *Web*, e com isso ganhar todo o dinamismo que ela oferece.

Contudo, portar um software com características *AM/FM* para um ambiente como a Internet não é trivial, devido aos conceitos envolvidos, exigindo alguma pesquisa extra. Logo desenvolveu-se este trabalho que visa a especificação e implementação de um protótipo de software, de *AM/FM*, que permita o mapeamento da cidade de Blumenau através do Sistema Cruzamento para a Internet.

Este protótipo será especificado através da orientação a objetos seguindo a metodologia da *Unified Modeling Language* (UML) e desenvolvido utilizando-se a tecnologia Java, através do emprego de *Servlets*, que tem se mostrado eficaz no desenvolvimento de aplicações voltadas a Internet.

1.1 MOTIVAÇÃO

A difusão da Internet cria um novo campo para o desenvolvimento de sistemas, campo esse muito promissor mas que precisa ser estudado e explorado para que se crie novos conceitos. Para que se tenha acesso a esse campo é necessário proporcionar uma nova visão aos programadores, uma vez que o desenvolvimento tradicional pode apresentar falhas quando se tenta criar algo voltado a Internet. Esta visão motivou a criação deste trabalho em uma tentativa de entender-se estes novos conceitos e consequentemente permitir acesso a esta área.

1.2 OBJETIVOS

Este Trabalho de Conclusão do Curso tem como objetivo desenvolver um protótipo de software AM/FM para Internet, baseado em estudos realizados no Sistema de Cotas Enchentes de Blumenau utilizando Java *Servlet*.

1.3 ORGANIZAÇÃO DO TEXTO

O segundo capítulo faz um levantamento histórico sobre o sistema atualmente em uso e aplicações correlatas.

O terceiro capítulo trata da revisão bibliográfica dos conceitos envolvidos no trabalho, como Geoprocessamento, *Automated Mapping/Facilities Management* (AM/FM), Internet, modelo Cliente Servidor e grafos, em particular, busca de caminho.

O quarto capítulo descreve a linguagem de programação Java, que é utilizada para desenvolver o protótipo, bem como *Applets* e *Servlets* que são as "especializações" utilizadas para criar aplicações para Internet.

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

No sexto capítulo tem-se as conclusões, limitações e trabalhos futuros.

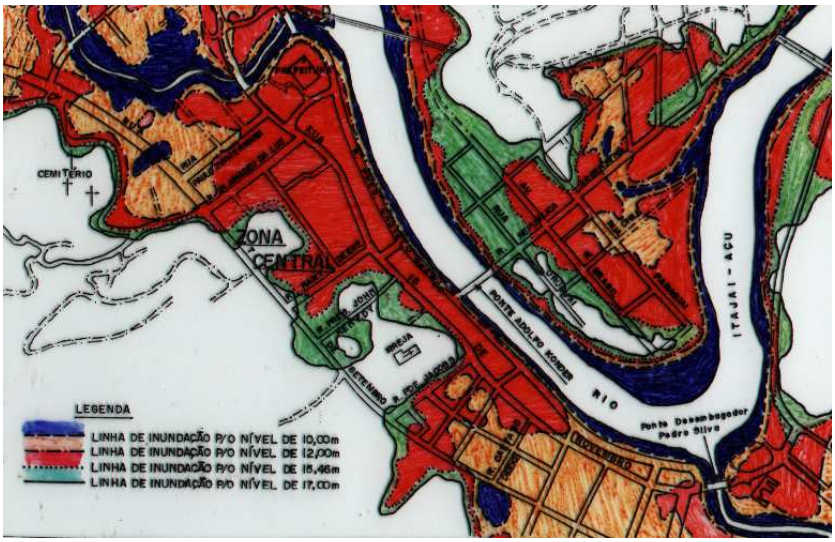
2 HISTÓRICO

2.1 SISTEMA ANTIGO

Visando solucionar alguns dos problemas oriundos da ocorrência de cheias, foram desenvolvidos junto ao Instituto de Pesquisas Ambientais a Carta Enchente e o Sistema Cruzamento.

A Carta Enchente (figura 1), realizada após a enchente de 1984, possibilita identificar as zonas de inundação, mas de uma forma não automatizada, dificultando assim a sua atualização, reprodução, distribuição e consulta. O que poderia ser solucionado através da automatização destas informações em um sistema computacional [KRA1999].

Figura 1 : Carta Enchente de Blumenau.

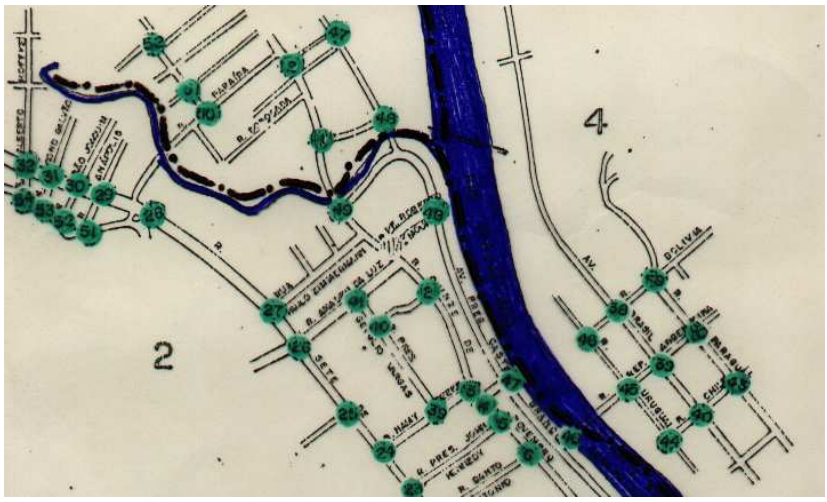


O Sistema Cruzamento (figura 2) foi desenvolvido em 1984 pelo matemático Prof° Cláudio Loesch e sua principal finalidade é orientar a locomoção de pessoas e/ou veículos através da malha viária de um município por ocasião da ocorrência de enchentes. Nestas circunstâncias, as opções de caminhos de acesso entre pontos distintos do município podem

sofrer grandes restrições. Estas, entretanto, são perfeitamente previsíveis, desde que se conheça o acréscimo sofrido pelo nível do rio.

Era composto de 892 cruzamentos, atualizados após a cheia de 1992 para 3.200 cruzamentos. Esses banco de dados abrange toda área do município, independente de sua inundação ou não, e apresenta, para cada cruzamento, além dos nomes das ruas, também a Cotas-Enchente relativa, um ponto de referência próximo, bem como a extensão entre os cruzamentos [KRA1999].

Figura 2 : Alguns cruzamentos cadastrados no Sistema Cruzamentos.



Como um trabalho de conclusão de curso foi construído uma interface gráfica para o sistema cruzamento [REI1992], no qual proporcionou um relacionamento espacial entre as ruas do município (figura 3).

Passados seis anos esse sistema sofreu nova atualização, decorrente de mudanças ocorridas na malha viária do município. E mais, a proposta do novo sistema era operacionalizar essas informações em novo sistema computacional interativo, reformulando a interface no padrão Windows, através de representação gráfica de mapas (figura 4). Esta atualização abrange desde a base de dados até o desenvolvimento de um novo ambiente o qual possa suprir as necessidades dos possíveis usuários [KRA1999].

Figura 3 : Interface Gráfica do Sistema Cruzamento.

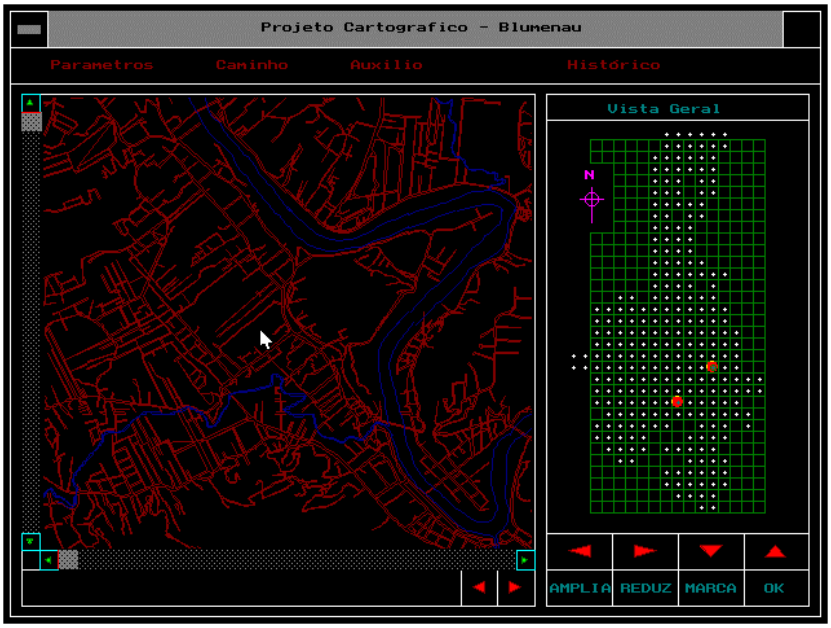
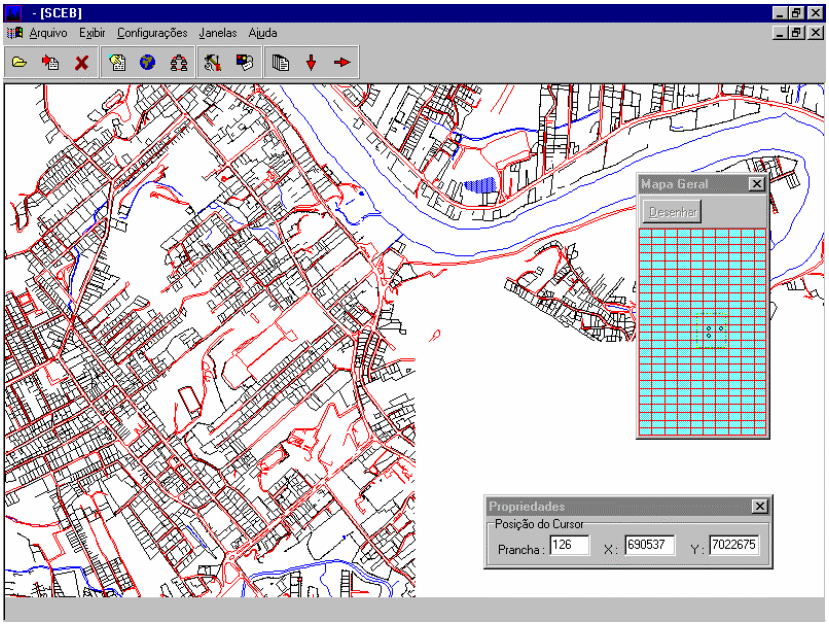


Figura 4 : Sistema Cruzamento atual.



Com o SCEB (Sistema de Cota-Enchente de Blumenau), como foi chamado, a população de Blumenau e interessados em geral poderiam obter informações precisas, e de forma rápida, não só de roteiros para deslocamento na cidade durante as enchentes. Seria possível também consultas e obtenção de mapas de regiões, em qualquer parte do município, com indicações sobre inundações, que podem ser úteis especialmente nas transações imobiliárias. Já para a administração municipal, a utilidade será no planejamento, no que diz respeito à elaboração do Plano Diretor de Ocupação do Uso do Solo, contribuindo ainda com a atuação de órgãos de Defesa Civil durante as cheias.

2.2 SISTEMA CORRELATO

Com o crescimento do número de habitantes e o aumento do desemprego nas cidades, a criminalidade está tornando-se maior, esse efeito tem sido notado não apenas nos grandes centros, mas também nas pequenas e médias cidades. Isso vem ocasionando um crescimento elevado nas ocorrências de segurança pública.

Pensando nisso, o aluno Sandro Alencar Fernandes desenvolveu o protótipo de um ambiente que permite o mapeamento da estrutura viária da cidade de Rio do Sul. Sobre este mapeamento pode-se consultar, de forma gráfica e nítida, informações úteis para a localização e deslocamento de unidades tanto da polícia, quanto do corpo de bombeiros aos locais das ocorrências [FER1999].

[SAF1] Comentário: Talvez isto seja eliminado..... ou siga como extenção

3 CONCEITOS RELEVANTES

Para que seja possível um maior entendimento do trabalho torna-se importante observar alguns conceitos que estão envolvidos.

3.1 GEOPROCESSAMENTO

O termo Geoprocessamento denota a disciplina do conhecimento que utiliza técnicas matemáticas e computacionais para o tratamento da informação geográfica e que vem influenciando de maneira crescente as áreas de Cartografia, Análise de Recursos Naturais, Transportes, Comunicações, Energia, Planejamento Urbano e Regional. As ferramentas computacionais para Geoprocessamento, chamadas de Sistemas de Informação Geográfica (SIG), permitem realizar análises complexas, ao integrar dados de diversas fontes e ao criar bancos de dados geo-referenciados. Tornam ainda possível automatizar a produção de documentos cartográficos [CAM1996].

O Geoprocessamento, de acordo com [INT1998], é tido como "a tecnologia de coleta e tratamento de informações espaciais e de desenvolvimento de sistemas que as utilizam".

Estes sistemas podem ser: *Geographic Information Systems (GIS)* ou Sistemas de Informações Geográficas (SIG), *Land Information Systems (LIS)* ou Sistemas de Informações de Terreno, *Automated Mapping / Facilities Management (AM/FM)* ou Mapeamento Automatizado / Gerenciamento de Serviços de Utilidade Pública, *Computer-Aided Drafting and Design (CADD)* ou Desenho e Rascunho Auxiliado por Computador [DEF1998a] e [DEF1998b].

3.1.1 MAPEAMENTO AUTOMATIZADO / GERÊNCIA DE SERVIÇOS DE UTILIDADE PÚBLICA *(AM/FM)*

AM/FM são sistemas de automação do processo de mapeamento e gerenciamento das informações representadas por itens em um mapa [INT1998].

Automated Mapping (*AM*), ou Mapeamento Automatizado, *Facility Management* (*FM*), isto é, Gerenciamento de Serviços de Utilidade Pública, baseiam-se também em

tecnologia *CADD*. Entretanto, a apresentação gráfica geralmente não é tão precisa e detalhada como em sistemas *CAM*; a ênfase de *AM/FM* está centrada no armazenamento, na análise e na emissão de relatórios. 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 [INT1998].

3.2 INTERNET

A Internet é um conjunto heterogêneos de computadores que se comunicam através de protocolos de rede [BEN1997].

Sob o ponto de vista físico, a Internet é uma conexão entre redes, mas para o usuário ela é um meio que pode ser usado para fornecer um grupo de serviços disponíveis para intercâmbio de informações entre computadores ou indivíduos que fazem parte desta rede.

Os principais serviços são:

a) Correio eletrônico (e-mail): permite que um usuário envie mensagens (texto) a outros usuários na rede;

b) *File Tansfer Protocol* (FTP): permite a transferência eficiente de arquivos entre computadores;

c) *Golpher*: sistema de obtenção de informação orientado por menus, no qual os arquivos disponíveis são indexados, por exemplo, pelo seu tema;

d) *Telnet*: sistema que permite que uma máquina possa ser um terminal de outra máquina na Internet;

e) *World Wibe Web* (WWW): permite disponibilizar através da *Hipertext Markup Language* (HTML) não só texto, mas também recursos multimídia como imagens, sons, vídeos, entre outros, de maneira transparente para o usuário.

3.3 MODELO CLIENTE-SERVIDOR

Uma arquitetura cliente/servidor é uma abordagem da computação que separa os processos em plataformas independentes que interagem, permitindo que os recursos sejam compartilhados enquanto se obtém o máximo de benefícios de cada dispositivo diferente. É basicamente uma forma distribuída ou em rede [BOC1995].

Existem algumas características que identificam a computação cliente-servidor, entre elas estão [BOC1995]:

a) uma arquitetura cliente/servidor consiste em um processo cliente e um processo servidor, que podem ser distinguido um do outro, embora possam interagir totalmente;

b) a parte cliente e a parte servidora podem operar em diferentes plataformas de computador, e geralmente operam, mas isso não é uma regra;

c) tanto a plataforma cliente como a servidora podem ser atualizadas sem que se tenha necessariamente que atualizar outra plataforma;

d) o servidor pode atender a vários clientes simultaneamente, em alguns sistemas cliente/servidor, os clientes podem acessar vários servidores;

e) os sistemas cliente/servidor incluem algum tipo de capacidade de operar em rede;

f) uma parte significativa (em alguns casos, toda ela) da lógica do aplicativo reside no cliente;

g) a ação, em geral, é iniciada no cliente, e não no servidor. No entanto, servidores de banco de dados podem “iniciar a ação” baseados em gatilhos, bem como através de regras de negócio ou procedimento armazenados;

h) uma interface gráfica amigável (GUI – *Graphical User Interface*) geralmente reside no cliente;

i) o servidor de banco de dados deve fornecer proteção e segurança de dados.

A computação cliente/servidor vem tomando espaço da arquitetura tradicional porque ela consegue aproveitar os recursos do sistema de uma forma racional e possivelmente por um custo inferior.

3.4 GRAFOS

O grafo, ou rede, ocorre basicamente em sistema nos quais existe um número finito de estados discretos possíveis e um processo de transição de um estado para outro, com um custo associado à transição entre estados. Essa estrutura é utilizada para modelagem de sistemas de transporte, comunicação, logística, produção, investimentos no mercado financeiro e assim por diante [SWA1990].

Tendo em vista a necessidade de se obter uma representação da malha viária da cidade Blumenau, a qual servirá para traçar caminhos entre dois cruzamentos, foi utilizado uma estrutura de grafo.

3.4.1 PROBLEMA DO MENOR CAMINHO

Dentre as técnicas existentes para a solução de problemas algorítmicos em grafos, a busca ocupa lugar de destaque, pelo grande número de problemas que podem ser resolvidos através da sua utilização. Dentre estes problemas, o problema do menor caminho é um dos problemas que podem ser facilmente modelados através de um grafo.

Suponha que se tenha um mapa da rede rodoviária em que as cidades são representadas por pontos e as estradas por linhas. O mapa pode ser considerado um grafo, em que as distâncias são ponderações das arestas, podendo ser consideradas como custos ou tempo [RAB1992]. A idéia é encontrar o menor caminho entre duas cidades no mapa, representado por quaisquer dois vértices no grafo.

3.4.2 ALGORITIMO DE DIJKSTRA

O algoritmo de Dijkstra, publicado em 1950, se propõem a resolver o problema do menor caminho [RAB1992].

Para apresentar estes algoritmos de busca, é necessário introduzir os conceitos de lista de nós abertos e lista de nós fechados. O conjunto de todos os nós abertos, ainda não expandidos, serão escritos em uma lista ABERTOS. Já os nós gerados e expandidos podem ser armazenados em um conjunto FECHADOS. Além disso:

a) Considere o digrafo $G(V, E)$, uma fonte v_0 e uma função L que associe a cada aresta a um número real não negativo, isto é,

$$\mathbf{L}(v_i,v_j) = \begin{cases} \infty & \text{, se não existe a aresta } (v_i,v_j) \\ 0 & \text{, se } v_i = v_j \\ \text{custo} & \text{, se } v_i \neq v_j \text{ e existe a aresta } (v_i,v_j); \end{cases}$$

b) Constrói-se um conjunto S, que contém os vértices v_i**'s** cujo comprimento mínimo de v_0 a cada v_i, seja conhecido;

c) A cada passo se adiciona ao conjunto S o vértice w pertencente a V-S tal que o comprimento do caminho v_0 a w, seja menor que o correspondente de qualquer outro vértice de V-S;

d) Pode-se garantir que o caminho mínimo de v_0 a qualquer vértice v em S contém somente vértices pertencentes a S.

O desenvolvimento do algoritmo de Dijkstra é feito da seguinte forma (quadro 1):

Quadro 1 : Algoritmo de Dijkstra.

Inicialização: ABERTOS ← (s); FECHADOS ← {}; $\hat{g}(s) \leftarrow 0$.

Algoritmo:

1. Se ABERTOS = (), pare com fracasso. Senão, vá ao passo 2;
2. selecione v de ABERTOS tal que $\hat{g}(v) = \min\{g(w) \mid w \in \text{ABERTOS}\}$, desempatando sempre em favor do nó terminal ∈ T;
3. ABERTOS ← ABERTOS – { v };
4. FECHADOS ← FECHADOS ∪ { v };
5. se v ∈ T, pare com sucesso;
   Gere $\Gamma^+(v)$;
   Para cada m ∈ Γ+(v) faça
   se (m ∉ ABERTOS ∪ FECHADOS) faça {
   $\hat{g}(m) \leftarrow \hat{g}(v) + c((v,m))$;
   PAI(m) ← v;
   ABERTOS ← ABERTOS ∪ { m };
   };
   senão faça {
   temp ← $\hat{g}(v) + c((v,m))$;
   se(temp < $\hat{g}(m)$ faça {
   $\hat{g}(m) \leftarrow$ temp;
   PAI(m) ← v;
   se m ∈ FECHADOS faça {
   ABERTOS ← ABERTOS ∪ { m };
   FECHADOS ← FECHADOS – { m };
   };
   };
   };
6. Volte para 1

4 LINGUAGEM JAVA

Originalmente chamado Oak, a linguagem de programação Java iniciou como parte de um grande projeto para desenvolver um software para consumidores de artigos eletrônicos, mais exatamente para criar uma rede heterogênea de eletrodomésticos. Inicialmente foi usada a linguagem C++ para este projeto, mas a grande variedade de arquiteturas de *hardware* usadas pelos consumidores tornou-se um problema onde a melhor solução seria mudar a linguagem. Então, a equipe da *Sun Microsystems* (*Sun*) desenvolveu uma nova linguagem que chamou de Java [NAR1999].

O Java é uma linguagem projetada para solucionar uma grande variedade de problemas que têm atormentado por vários anos a comunidade da informática; a boa execução na Internet é quase um subproduto da solução desses problemas. Sendo assim, Java é uma linguagem de programação que fornece os alicerces para o desenvolvimento de aplicativos para Internet [HOP1997].

Java não é apenas uma linguagem para programação na Internet, na verdade, é uma alternativa ao C++. É uma linguagem para criar programas seguros, portáveis, robustos, orientados a objetos, de multiprocessos (*multithreaded*) e interativos. Java está estritamente associada à Internet porque o primeiro grande aplicativo, o chamado "*killer-app*" escrito em Java, era um *browser* da *Web* interativo [NAU1997].

4.1 CARACTERÍSTICAS DA LINGUAGEM JAVA

Devido ao fato do Java ter sido desenvolvido com o propósito de ser uma linguagem revolucionária, para atender aos requisitos do mundo real ela teve que incorporar várias características que podem ser resumidos a seguir [NAU1997].

4.1.1 SIMPLES E PODEROSA

Pode-se aprender a programar em Java rapidamente depois de se entender os conceitos básicos da programação orientada a objetos. Com esses conceitos já é possível escrever programas produtivos e satisfatórios, pois os métodos para realizar determinadas

tarefas são claros e em número reduzido. Isso permite que o programador não tenha que se preocupar com as complexidades do funcionamento interno da máquina. Mas esse estilo de simplicidade não tira a habilidade de inovar, pelo contrario, Java permite expressar cada idéia que se tiver de forma clara e limpa.

4.1.2 SEGURA

Um dos principais princípios de projeto do Java é a segurança. Ela nunca teve nenhum recurso inseguro que depois tenha tido que ser coberto para torná-la segura. A maioria das maneiras consideradas óbvias de "invadir" um sistema simplesmente não pode ser descrita como um programa Java. Visto que programas Java não podem chamar funções globais e ter acesso a recursos arbitrários de sistema, há um certo nível de controle que pode ser exercido pelos *runtimes* do Java, e que não podem ser abordados pelos outros sistemas.

4.1.3 ORIENTADA A OBJETOS

Usando uma abordagem limpa, utilizável e pragmática para os objetos, Java conseguiu um equilíbrio entre o modelo purista, "tudo é um objeto", e o modelo de programador prático. A maioria dos outros sistemas orientados a objetos preferiram as hierarquias rígidas, difíceis de gerenciar, ou usam modelos de objetos completamente dinâmicos que deixam de lado grande parte do desempenho e compreensão. Java novamente consegue um equilíbrio, fornecendo um mecanismo de classes simples, com um modelo de interface dinâmico e intuitivo.

4.1.4 ROBUSTA

A maioria dos programas em uso hoje em dia fracassa por um dos dois motivos: gerenciamento de memória ou tratamento de exceções. Java praticamente elimina ambos os problemas com o gerenciamento avançado de memória chamado "coleta de lixo" (*garbage collection*) e com um tratamento de exceções integrado e orientado a objetos. Como Java é uma linguagem muito restrita no que diz respeito aos tipos e declarações, a maioria dos erros comuns pode ser detectada em tempo de compilação. Esse é um exemplo em que Java é menos flexível do que outra linguagens.

4.1.5 INTERATIVA

Java foi criada para atender a requisitos do mundo real, entre eles a criação de programas interativos e em rede. A maioria dos sistemas tem problemas ao lidar com um desses recursos, e muitos com os dois. Java tem vários recursos avançados que permitem escrever programas que fazem muitas coisas ao mesmo tempo, sem perder o controle do que poderia acontecer e quando. O *runtime* do Java vem com a mais elegante solução que já foi criada para a sincronia de vários processos, a de várias linhas de execução (*threads*).

4.1.6 NEUTRA EM RELAÇÃO À ARQUITETURA

Os criadores do Java tomaram várias decisões difíceis na linguagem e no *runtime* do Java, portanto pode-se mesmo "escrever uma vez, executar em qualquer parte, a qualquer tempo e sempre". Você só precisa se preocupar em escrever um bom programa e o Java vai garantir que ele funcione em Macintosh, PC, UNIX, ou em qualquer outra plataforma.

4.1.7 INTERPRETADA E DE ALTO DESEMPENHO

Java realiza a extraordinária façanha da independência de plataforma, compilando em uma representação intermediária chamada *bytecode* Java, a qual pode ser interpretada em qualquer sistema que tenha um *runtime* Java apropriado. Esse *bytecode* foi cuidadosamente projetado para que seja fácil traduzí-la diretamente para o código da máquina nativa e tenha um desempenho alto. Os sistemas do *runtime* do Java que fazem a otimização do código "*just in time*" não perdem nenhum beneficio do código neutro em relação às plataformas.

4.2 JDBC

JDBC é uma API Java para executar comandos SQL. Ela é formada por classes e interfaces escritas na linguagem de programação JAVA. JDBC oferece uma API padrão onde os desenvolvedores podem criar aplicações de banco de dados usando Java. Em outras palavras, com JDBC API não é necessário escrever um programa para acessar cada banco de dados, pode-se escrever um simples programa usando a API JDBC e este programa estará pronto para enviar comandos SQL para os banco [JEP1997].

Com o JDBC pode-se, basicamente, fazer três coisas:

a) estabelecer uma conexão com um banco de dados;

b) enviar declarações SQL;

c) processar os resultados.

4.3 APPLET

Applets são pequenos programas escritos em Java que são embutido em páginas *Web* para produzir os mais diversos efeitos. Quando os usuários acessam essas páginas, as *Applets* são descarregados (*downloaded*) para seus computadores e executadas com um *browser* habilitado para Java. *Applets* são tratadas pelo *browser* da mesma forma que qualquer outro recurso de mídia, como arquivos de imagens, áudio ou vídeo.

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 as *Applets* não são restritas por largura de banda ou velocidade do modem quando estão sendo executadas. Os usuários podem ver e ouvir efeitos multimídia em páginas *Web* de forma eficaz e oportuna [HOF1997].

As *Applets* estão se tornando rapidamente populares. Existem diversas categorias de *Applets* na Internet: animação, jogos, colaboração, compras, aplicativos, acessos Cliente/Servidor, etc.

O termo *Applet* deu origem à marcação `<Applet>` que aparece em um documento, HTML e informa ao *browser* para carregar o código apropriado. Além se usar a *tag* apropriada é necessário informar alguns atributos:

a) *code*: indica a classe que será executada, o nome do arquivo que contém a classe principal da *Applet*;

b) *width* e *height*: são usados para especificar a área que limita a *Applet*, especificando-se, respectivamente, a largura e altura inicial da janela de apresentação da *Applet*, em *pixels*;

c) *codebase*: o URL base da *Applet*. O código da *Applet* é localizado relativamente a este URL. Se esse atributo não é especificado, então é assumido o URL do documento em que a *Applet* está especificada;

d) *alt*: texto alternativo que pode ser exibido em *browser* configurado em modo apenas texto;

e) *name*: o nome simbólico da *Applet*. Este nome pode ser usado por outras *Applets* na mesma página para localizar essa *Applet*;

f) *align*: o alinhamento da *Applet*. Este parâmetro afeta como a *Applet* é posicionada na página;

g) *vspace* e *hspace*: respectivamente, o espaço vertical e espaço horizontal ao redor da *Applet*.

4.4 SERVLETS

Os *Servlets* foram desenvolvidos para trabalhar no modelo de *request*/*response*. Neste modelo, um cliente envia uma mensagem para o servidor (*request*) e o servidor responde (*response*) enviando uma mensagem de volta. As requisições podem chegar na forma dos protocolos HTTP, URL, FTP ou outro customizado ([JAV2000a]).

A API Java *Servlet* contém várias Interfaces Java que fornecem suporte para tratar a ligação entre o servidor remoto e os *Servlets*. A *Servlet* API é uma extensão do padrão JDK, ou seja, ela não é fornecida pela core API, estando abaixo do pacote javax. A API Java *Servlet* contém dois pacotes ([JAV2000b]):

- `javax.Servlet`;
- `javax.Servlet.http`.

Servlets não rodam da mesma forma que *Applets* e aplicações. Os *Servlets* fornecem funcionalidades para estender um servidor *Web*, portanto, são executados no servidor e precisam estar integrados de alguma forma ao mesmo.

A API Java *Servlet* define a interface entre os *Servlets* e o servidor fornece suporte para quatro categorias de serviço ([JAV2000a]).

- Gerenciamento do ciclo de vida do *Servlet*;
- Acesso ao contexto do *Servlet*;
- Classe de utilitários;
- Classe de suporte para protocolos HTTP específicos.

4.4.1 CICLO DE VIDA DO SERVLET

Servlets rodam na mesma plataforma do *Web Server*, como parte do mesmo processo que o servidor *Web*. O *Web Server* é responsável pela inicialização, invocação e destruição de cada instância do *Servlet*.

O *Web Server* se comunica com o *Servlet* através de uma simples interface, `javax.Servlet.Servlet ([JAV2000b])`. Esta interface consiste em três métodos principais:

a) `init()` : o método `init()` será chamado apenas uma vez quando um *Servlet* é carregado pela primeira vez. Isto permite ao *Servlet* fazer algum processo de inicialização como abrir arquivos ou estabelecer conexões com outros servidores;

b) `service()` : o método `service()` é o núcleo de um *Servlet*, é ele que recebe as requisições do cliente, processa e envia a resposta. Cada mensagem de requisição de um cliente resulta em uma chamada ao método `service()`;

c) `destroy()` : o método `destroy()` é chamado antes que o *Servlet* seja finalizado a fim de possibilitar a liberação de recursos que estejam sendo usados, como uma conexão ao banco de dados ou um arquivo aberto.

E dois métodos auxiliares:

- `getServletConfig()`;

- `getServletInfo()`.

Pode-se perceber a semelhança entre esta interface e a interface das *Applets* Java. *Servlets* são para o *Web* Server o que as *Applets* são para o *browser Web*. Uma *Applet* é executada em um *browser Web*, executando ações e respostas através de uma interface específica. Um *Servlet* faz a mesma coisa, só que rodando em um servidor *Web*.

4.4.2 SERVLET CONTEXT

Um *Servlet* nasce e morre nos limites de um processo do servidor. Para compreender o ambiente de operação, um *Servlet* pode recuperar informações sobre este ambiente durante o seu ciclo de vida. A qualquer momento pode-se obter informações sobre o servidor e em cada requisição pode-se obter informações específicas sobre o contexto atual ([JAV2000b]).

Informações de inicialização são passadas ao *Servlet* através do parâmetro `ServletConfig` no método `init()`. Cada servidor *Web* tem seus próprios meios de enviar informações de inicialização ao *Servlet*.

Informações sobre o contexto do servidor estão disponíveis a qualquer momento através do objeto `ServletContext`. Um *Servlet* pode obter este objeto chamando o método `getServletContext()` no objeto `ServletConfig`.

Cada requisição pode conter informações na forma de pares parâmetros nome/valor, como um `ServletInputStream` ou um `BufferedReader`. Esta informação está disponível pelo objeto `ServletRequest` que é passado para o método `service()`.

4.4.3 CLASSES UTILITÁRIAS

A API *Servlet* fornece diversos utilitários. O primeiro é a interface `javax.Servlet.SingleThreadModel`, que forma fácil escrever *Servlets* simples. Se um *Servlet* implementa esta interface, o servidor que o hospeda saberá que não pode fazer uma chamada ao método `service()` do *Servlet* antes que seja terminado o processamento de um *request*. Isto é, o servidor processa todos os serviços com um simples *thread* ([JAV2000a]).

Duas classes de exceção estão incluídas na API. A exceção `javax.Servlet.ServletException` pode ser usada quando ocorrer uma falha geral. Ela notifica o servidor que há um problema. A exceção `javax.Servlet.UnavailableException` indica que o *Servlet* não está disponível. Os *Servlets* podem gerar estas exceções a qualquer tempo.

4.4.4 SUPORTE HTTP

Servlets que usam o protocolo HTTP são muito comuns. O suporte a tratamento do protocolo HTTP é fornecido pelo pacote `javax.Servlet.http`. A classe abstrata `javax.Servlet.http.HttpServlet` fornece uma implementação da interface `java.Servlet.Servlet` ([JAV2000a]).

A forma mais confortável de escrever um *Servlet* HTTP é estendendo `HttpServlet` e adicionando a lógica do sistema. A classe `HttpServlet` fornece uma implementação do método `service()` que despacha as mensagens HTTP para um dos vários métodos especiais.

Estes métodos que correspondem diretamente aos métodos do protocolo HTTP são:

- `doGet()`;
- `doDelete()`;
- `doOptions()`;
- `doPost()`;
- `doTrace()`;
- `doPut()`.

A classe `HttpServlet` é bastante inteligente. Ela não só despacha a requisição HTTP, como detecta qual método está sobreposto em uma subclasse.

Quando se usa as classes de suporte HTTP, geralmente se cria um novo *Servlet* que estende `HttpServlet` e sobrepõe `doGet()` ou `doPost()`, ou possivelmente os dois. Outros métodos podem ser sobrepostos a fim de obter um controle mais fino.

Para os métodos de processamento do HTTP são passados dois parâmetros, um objeto `HttpServletRequest` e um objeto `HttpServletResponse`. A classe `HttpServletRequest` possui vários métodos convenientes para tratar uma requisição, ou pode-se simplesmente ler o texto da requisição.

5 DESENVOLVIMENTO DO SISTEMA

Este trabalho é baseado no Sistema Cruzamento realizado por Danilo Kramel [KRA1999].

5.1 ANÁLISE DO SISTEMA

Para a especificação do sistema proposto neste trabalho será usada a UML que composta por vários diagramas, dos quais podemos citar [FUR1998]:

a) diagrama de classe;

b) diagrama de caso de uso;

c) diagramas de seqüência;

d) diagrama de colaboração;

e) diagrama de estado;

f) diagrama de atividade;

g) diagrama de componente;

h) diagrama de implantação.

Devido ao caráter do trabalho utilizou-se apenas os três primeiros diagramas, com o auxílio da ferramenta case Rational Rose 2000.

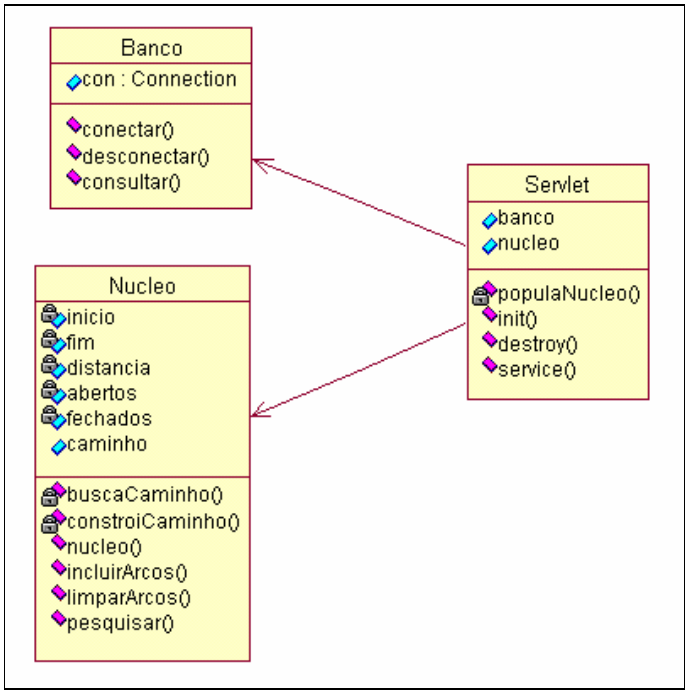
Como o protótipo faz uso de um banco de dados relacional, também utilizou-se um Modelo Entidade Relacionamento (MER) para representar o mesmo.

5.1.1 DIAGRAMA DE CLASSE

Denota a estrutura estática do sistema e as classes representam as entidades que são manipuladas por este sistema. O diagrama é considerado estático pois a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema [FUR1998].

O diagrama de classes do sistema pode ser visto na figura 5.

Figura 5 : Diagrama de classes para o protótipo

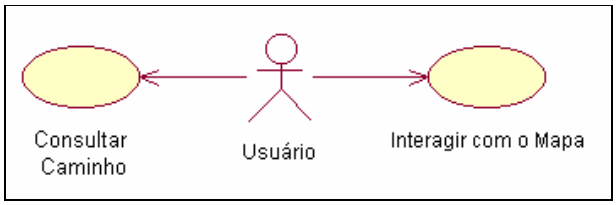


5.1.2 DIAGRAMA DE CASO DE USO

O diagrama de caso de uso descrevem a funcionalidade do sistema percebida por atores externos. Um ator interage com o sistema podendo ser um usuário, dispositivo ou outro sistema [FUR1998].

Para o sistema este diagrama esta representado na figura 6.

Figura 6 : Diagrama de casos de uso para o protótipo

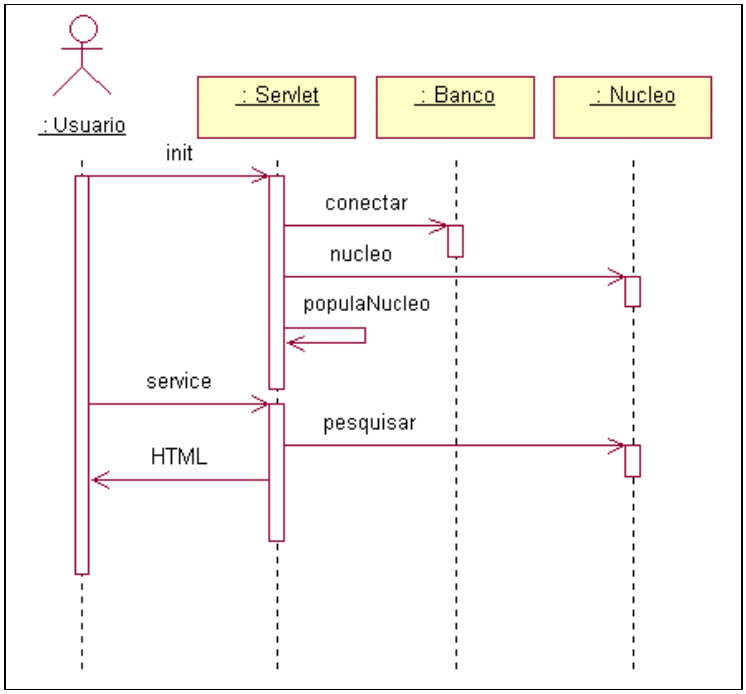


5.1.3 DIAGRAMA DE SEQÜÊNCIA

Apresenta a interação de seqüência de tempo dos objetos que participam na interação. As duas dimensões de um diagrama de seqüência consistem na dimensão vertical (tempo) e na dimensão horizontal (objetos diferentes). O diagrama de seqüência mostra a colaboração dinâmica entre um número de objetos, o aspecto importante desse diagrama é mostrar a seqüência de mensagens enviadas entre objetos [FUR1998].

Na figura 7 pode ser visto um diagrama de seqüência para o protótipo.

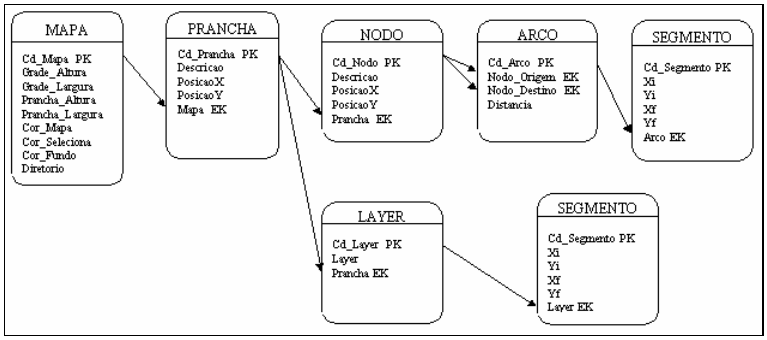
Figura 7 : Diagrama de seqüência do protótipo



5.1.4 MODELO ENTIDADE RELACIONAMENTO

Como este protótipo se utiliza de um banco de dados relacional na figura 8 podemos ver o modelo entidade relacionamento do sistema. Vale ressaltar que o modelo representado é igual ao utilizado pelo sistema antigo, logo, não foi feita qualquer alteração no mesmo.

Figura 8 : MER do sistema antigo



5.2 IMPLEMENTAÇÃO

Neste tópico será demostrado o desenvolvimento do protótipo, desde a instalação do software necessário até implementação e a execução do sistema.

5.2.1 SOFTWARES NECESSÁRIOS

Os softwares necessários para a instalação e funcionamento do sistema são:

a) Java : para instalar a linguagem Java basta instalar o JDK que é composto pelo compilador Java, o interpretador e bibliotecas de classes, sendo o mesmo distribuído gratuitamente em vários sites da Internet. Para a implementação foi utilizado o pacote JDK 1.2.2 que possui o programa de instalação para a plataforma Windows.

b) *Browser* : para executar o software cliente é necessário a utilização de um *browser* compatível com a tecnologia Java, como o Netscape. Para a instalação basta executar o assistente e seguir as instruções.

c) Java Server Web Development Kit (JSWDK) : o pacote JSWDK possui a biblioteca de classes utilizadas para criar os *Servlets*. Basta efetuar a descompactação e acrescentar o diretório do Kit nas variáveis ambientes PATH e CLASSPATH. O pacote utilizado no desenvolvimento foi o JSWDK 2.1.

d) Servidor *Web* : para executar o software servidor é necessário possuir um servidor http compatível com a tecnologia *Servlet*, como o Apache. Também pode ser utilizado o próprio JSWDK, que possui uma implementação de um *Web Server*, através da execução do arquivo *startserver.bat* e para finalizar o arquivo *stopserver.bat*.

5.2.2 FUNCIONAMENTO DO PROTÓTIPO

O protótipo é composto por dois módulos:

a) "Software Cliente" : o que é apresentado no *browser* do usuário, basicamente uma página *Web* comum composta de dois *frames* fixos e um móvel, disponibiliza uma interface gráfica amigável ao usuário. Para obter os dados necessários ao funcionamento, comunica-se com o Software Servidor através de requisições http. Devido ao fato do HTML não trabalhar satisfatoriamente com gráficos vetoriais (mapas) criou-se uma *Applet* com o objetivo de exibi-los.

b) Software Servidor : o Servidor é a parte principal do sistema, ele é composta de três classes, cada uma responsável por determinada parte do processamento. A classe Banco realiza as tarefas relacionadas com a conexão, desconexão e consulta ao banco de dados, a classe Núcleo executa as tarefas de controle do grafo, entre elas a busca do menor caminho e por fim a classe *Servlet*, que implementa a tecnologia *Servlet* responsável pelo atendimento das requisições do cliente. Quando executado pela primeira vez, cria instâncias das classes Núcleo (responsável pelo grafo) e Banco (responsável pela conexão com o banco de dados). Após a inicialização fica aguardando chamadas http e processa-as conforme o solicitado, repassando os resultados ao Cliente.

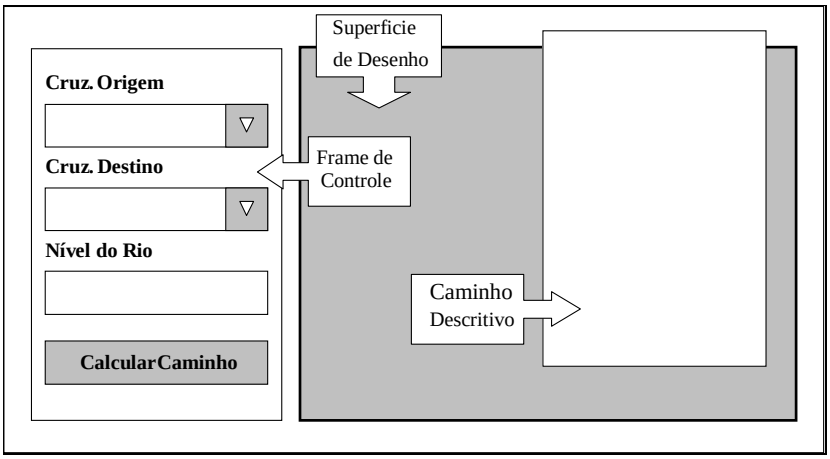
5.2.3 IMPLEMENTAÇÃO DO PROTÓTIPO

A implementação está dividida em duas partes, a implementação do software cliente e do software servidor. Para demonstrar a implementação do protótipo serão demonstrados vários quadros com partes da implementação final.

5.2.3.1 IMPLEMENTAÇÃO DO SOFTWARE CLIENTE

O *layout* inicial da tela do sistema foi idealizado conforme a figura 9, sendo que o mesmo é composto por um *frame* de controle que contém dois campos para a escolha dos cruzamentos inicial e final, um para se digitar o nível do rio e o botão para a execução do processo, um *frame* responsável por mostrar o desenho das ruas (mapa) e por último um *frame* flutuante que mostra o caminho descritivamente (nome das ruas a serem seguidas).

Figura 9 : Layout do "Software Cliente"



Como o Software Servidor é responsável pela formatação em HTML e a única coisa que realmente precisa de uma implementação no "Software Cliente" é a *Applet* de desenho trataremos desta nessa seção.

O Quadro 2 mostra as *packages* que precisam ser carregadas para a criação da *Applet* de desenho. As packages "java.awt.*" e "java.*Applet*.*Applet*" contém as classes com os componentes para o desenvolvimento de interface gráfica, tais como botões, menus e outros, e a package "java.util.*" possui alguns utilitários, como por exemplo, as classes Vector e StringTokenizer.

Quadro 2 : Classes necessárias para a criação do Software Cliente

```
import java.Applet.Applet;
import java.awt.*;
import java.util.*;
```

No Quadro 3 é mostrado o método init() que é executado na inicialização da *Applet*, é utilizado para ler os parâmetros que são passados através do HTML que contém a *Applet*. Esses parâmetros são as coordenadas de desenho do mapa e do caminho.

Quadro 3 : Método de inicialização da Applet

```
public void init(){
    StringTokenizer st;
    // Tratar mapa
    buffer.setLength(0);
    buffer.append(getParameter("MAPA"));
    st = new StringTokenizer(buffer.toString()," ");
    while(st.hasMoreTokens()){
         mapa.addElement(st.nextToken());
    }
    // Tratar caminho
    buffer.setLength(0);
    buffer.append(getParameter("CAMINHO"));
    st = new StringTokenizer(buffer.toString()," ");
    while(st.hasMoreTokens()){
         caminho.addElement(st.nextToken());
    }
}
```

O Quadro 4 mostra o método paint (Graphics g) usado para desenhar o mapa utilizando-se para isso as coordenadas que foram passadas anteriormente.

Quadro 4 : Método que desenha o mapa

```
public void paint(Graphics g){
    int i;
    Integer xi, xf, yi, yf;
    for (i=3; i<mapa.capacity();){
         xi=new Integer(mapa.elementAt(i-3).toString());
         xf=new Integer(mapa.elementAt(i-2).toString());
         yi=new Integer(mapa.elementAt(i-1).toString());
         yf=new Integer(mapa.elementAt(i).toString());
         g.drawLine(xi.intValue(), xf.intValue(),
                        yi.intValue(), f.intValue());
         i = i + 4;
    }
    g.setColor(Color.red);
    for (i=3; i<caminho.capacity();){
         xi=new Integer(caminho.elementAt
                              (i-3).toString());
         xf=new Integer(caminho.elementAt
                              (i-2).toString());
         yi=new Integer(caminho.elementAt
                              (i-1).toString());
         yf=new Integer(caminho.elementAt
                              (i).toString());
         g.drawLine(xi.intValue(), xf.intValue(),
                        yi.intValue(), yf.intValue());
         i = i + 4;
    }
}
```

5.2.3.2 IMPLEMENTAÇÃO DO SOFTWARE SERVIDOR

Como o servidor se divide em três classes bem distintas, vamos tratá-las uma a uma começando pela classe Banco (Quadro 5).

Quadro 5 : Classe Banco

```
import java.sql.*;
class Banco{
    Connection con;
    boolean conectar(String driver, String banco,
                        String usuario, String senha){
         try {
              Class.forName(driver);
              con = DriverManager.getConnection
                        (banco,usuario,senha);}
         catch (Exception e){
              return false;}
         return true;}
    boolean desconectar(){
         try {
              con.close();}
         catch (Exception e){
              return false;}
         return true;}
    ResultSet consultar(String consulta){
         Statement st;
         ResultSet rs;
         try{
              st = con.createStatement();
              rs = st.executeQuery(consulta);}
         catch (Exception e){
              rs = null;}
         return rs;}
}
```

Primeiramente é feito a importação do único pacote necessário para a implementação da classe, “java.sql.*”, que responsável pelas classes que tratam as requisições ao banco de dados.

Logo depois, temos os métodos da classe, começando pelo método conectar(String driver, String banco, String usuário, String senha) que efetua uma conexão ao banco de dados utilizando o driver indicado e usa as strings usuário e senha para "*loggar*" no banco. O método retorna *true* se a conexão foi um sucesso, ou *false*, caso ocorra algum problema.

O método desconectar() fecha a conexão com o banco de dados. Também retorna *true* ou *false* conforme o sucesso da operação.

Por fim, o método consultar executa um comando sql contido na string de entrada e armazena o resultado em um *ResultSet* que e utilizado como retorno do método. Caso ocorra algum problema com a consulta, o retorno assume valor *null*.

No quadro a seguir (Quadro 6), pode-se ver a implementação de alguns métodos da classe Nucleo, mais especificamente o construtor, e os métodos de inclusão e exclusão dos arcos do grafo (caminhos). Observa-se no construtor a instanciação de vários Vectors que serão utilizados para armazenamento de informações referentes ao grafo.

Quadro 6 : Alguns métodos da classe Nucleo

```
public Nucleo(){
    inicio = new Vector();
    fim = new Vector();
    distancia = new Vector();
    abertos = new Vector();
    fechados = new Vector();
    caminho = new Vector();
    paiFechado = new Vector();
    paiAberto = new Vector();
    menor = new Vector();}
public void incluirArcos(String nodoI, String nodoF,
                                       String dist){
    inicio.addElement(nodoI);
    fim.addElement(nodoF);
    distancia.addElement(dist);}
public void limparArcos(){
    inicio.removeAllElements();
    fim.removeAllElements();
    distancia.removeAllElements();}
```

No Quadro 7 pode-se observar a implementação dos dois métodos principais da classe Nucleo. BuscaCaminho recebe os nodos inicial e final e procura um caminho válido, se encontrar um caminho, chama o método constroiCaminho para verificar o tamanho do mesmo. Este processo se repete até que se encontre o menor caminho.

Quadro 7 : Métodos principais da classe Nucleo

```
public float constroiCaminho(String valorInicial,
                             String valorFinal){
    String procura;
    float dist=0;
    int i = fechados.size()-1;
    while(!paiFechado.elementAt(i).equals("#")){
              caminho.addElement(fechados.elementAt(i));
    dist = dist+(Float.valueOf((String)distancia.
                        elementAt(i))).floatValue();
    procura = paiFechado.elementAt(i).toString();
    i = 0;
    while(!fechados.elementAt(i).equals(procura))
         i++;
    }
    caminho.addElement(fechados.elementAt(i));
    dist = dist+(Float.valueOf((String)distancia.
                        elementAt(i))).floatValue();
    return(dist);
}

public boolean buscaCaminho(String valorInicial,
                        String valorFinal){
    caminho.removeAllElements();
    abertos.removeAllElements();
    fechados.removeAllElements();
    paiFechado.removeAllElements();
    paiAberto.removeAllElements();
    String nodo;
```

```
boolean possui = false;
abertos.addElement(valorInicial);
paiFechado.addElement("#");
while(!abertos.isEmpty()){
     nodo = abertos.remove(abertos.size()-1).
                                        toString();
     if(valorFinal.equals(nodo)){
     //construir caminho c/ paiFechado e paiAberto
          paiFechado.addElement(paiAberto.elementAt
                              (paiAberto.size()-1));
          paiAberto.removeElementAt(paiAberto.size()
                                             -1);
          fechados.addElement(nodo);
          return true;
     }
     //buscar todos os filhos de nodo
     for(int i=0; i<inicio.size(); i++)
          if(inicio.elementAt(i).equals(nodo)){
               if(fechados.isEmpty()){
                    paiAberto.addElement
                              (inicio.elementAt(i));
                    abertos.addElement
                              (fim.elementAt(i));
               }
               for(int j=0; j<fechados.size(); j++)
                    if(!(fim.elementAt(i).equals
                           (fechados.elementAt(j))))
                         possui = true;
               for(int k=0; k<abertos.size(); k++)
                    if(!(fim.elementAt(i).equals
                           (abertos.elementAt(k))))
                         possui = true;
               if(possui){
```

```
                              paiAberto.addElement
                                        (inicio.elementAt(i));
                              abertos.addElement
                                        (fim.elementAt(i));
                              possui = false;
                         }
                    }
                    if(!paiAberto.isEmpty())
                         paiFechado.addElement(paiAberto.
                              elementAt(paiAberto.size()-1));
                    fechados.addElement(nodo);
               }
               return false;}
```

A classe *Servlet* utiliza algumas packages (pacotes de classes) incomuns, como se pode ver no Quadro 8, as packages "javax.*Servlet*.*" e "javax.*Servlet*.http.*" possuem as classes para criação dos *Servlets* e tratamento dos eventos que ocorrem nos mesmos. As outras duas já forma vistas anteriormente.

Quadro 8 : Packages utilizadas na classe *Servlet*

```
import java.io.*;
import java.sql.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
```

No quadro abaixo (Quadro 9), esta listado os métodos init() e destroy() do *Servlet* que são responsáveis pela instanciação das classes Banco e Nucleo, bem como a conexão com o banco de dados e construção do grafo através do método populaNucleo(), que carregará para a memória toda a estrutura do grafo em questão.

Quadro 9 : Métodos init e destroy do *Servlet*

```
public void init(ServletConfig ServletConfig)
                              throws ServletException{
    nucleo = new Nucleo();
    boolean ok = true;
    ResultSet tab;
    super.init(ServletConfig);
    try {
         System.out.println("Conectando...");
         banco = new Banco();
         ok = banco.conectar("sun.jdbc.odbc.
              JdbcOdbcDriver","jdbc:odbc:TCC","","");
         if (ok){
              System.out.println("Pronto!");
              populaNucleo();}
         else System.out.println("Negado!");}
    catch (Exception e){
         System.out.println("Problemas...");}}
public void destroy(){
    boolean ok = true;
    try {
         System.out.print("Fechando...");
         ok=banco.desconectar();
         if (ok) System.out.println("Pronto!");
         else System.out.println("Negado!");}
    catch (Exception e){
         System.out.println("Problemas...");}}
```

No próximo quadro (Quadro 10) temos a implementação do método Service do *Servlet* que é responsável por receber e enviar os dados HTTP do cliente. Quando recebe uma requisição, a classe verifica o tipo da operação, e então executa alguma seqüência conforme o tipo. No quadro pode-se ver o "*if*" que será executado caso o tipo da operação seja igual a "getRuas".

Quadro 10 : Método Service

```
public void service (HttpServletRequest
          httpServletRequest, HttpServletResponse
          httpServletResponse)throws ServletException,
          IOException{
     httpServletResponse.setContentType("text/plain");
     StringBuffer buffer = new StringBuffer();
     String query = new String();
     ResultSet rs;
     boolean more;
     try{
          String operacao = httpServletRequest.
                         getParameter("par_operacao");
          ......
          if(operacao.equalsIgnoreCase("getRuas")){
               query = "select descricao from nodo";
               rs = banco.consultar(query);
               more = rs.next ();
               while(more){
                    buffer.append(rs.getString(1)+"#");
                    more = rs.next ();}
               rs.close();
           }else
          ......
     } catch(SQLException sql){}
 httpServletResponse.setContentLength(buffer.length());
        httpServletResponse.getOutputStream().
print(buffer.toString());}
```

5.2.3.3 CRIANDO UM PÁGINA HTML

Para se executar a *Applet* cliente é necessário criar uma página html que será carregada no *browser*. O Quadro 11 mostra o código que compões este arquivo, sendo que a principal parte do mesmo é a tag <APPLET CODE=Tela WIDTH=600

HEIGHT=500></APPLET>, que executa a *Applet* Cliente. As tags PARAM passam para a Applet as coordenadas que serão utilizados para desenhar o mapa e o caminho.

Quadro 11 : Página HTML

```
<HTML>
<HEAD>
<TITLE>TCC-TRABALHO DE CONCLUSÃO DE CURSO</TITLE>
</HEAD>
<BODY>
<H1>SISTEMA CRUZAMENTO</H1>
<HR>
<APPLET CODE = Tela WIDTH=300 HEIGHT=200>
<PARAM NAME=MAPA VALUE="10 10 50 10 ... 10 30 50 30">
<PARAM NAME=CAMINHO VALUE="10 10 ... 50 30">
</APPLET>
<HR>
</BODY>
</HTML>
```

5.3 FUNCIONAMENTO DO PROTÓTIPO

Para iniciar o uso do protótipo é necessário executar algumas configurações como será visto a seguir.

5.3.1 CONFIGURAÇÃO DO SERVIDOR

Devido a quantidade disponível de servidores *Web*, cada um com suas próprias configurações, será explicado apenas como configurar o JSDK que foi utilizado para implementar o protótipo.

No Quadro 12 é possível ver o arquivo de configuração `Servlets.properties`, utilizado para a definição dos *Servlets* usados pelo servidor. Através deste arquivo também é possível passar parâmetros para os *Servlets*.

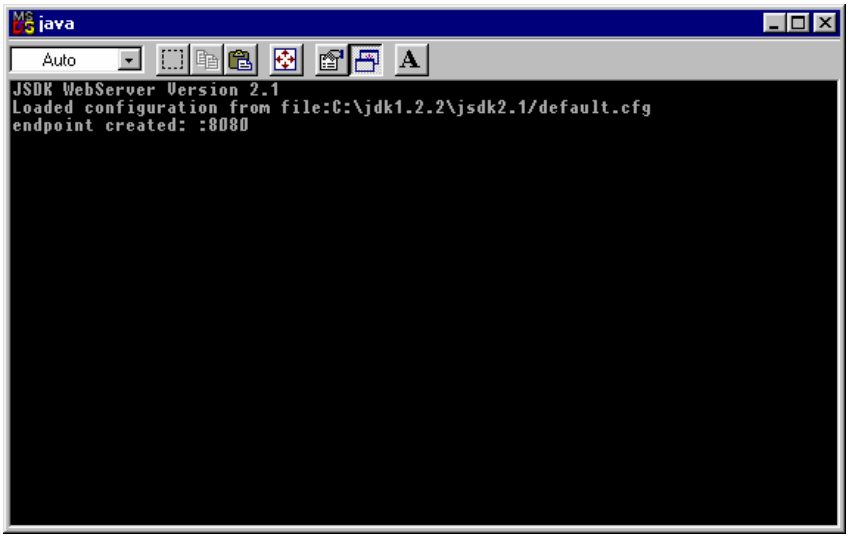
Quadro 12 : Arquivo de configuração *Servlets*.properties

```
# Define Servlets here
# <Servletname>.code=<Servletclass>
# <Servletname>.initparams=<name=value>,<name=value>

sceb.code=Servlet
```

Após a configuração basta executar o arquivo startserver.bat e o servidor estará pronto para aceitar conexões (figura 10).

Figura 10 : Servidor rodando

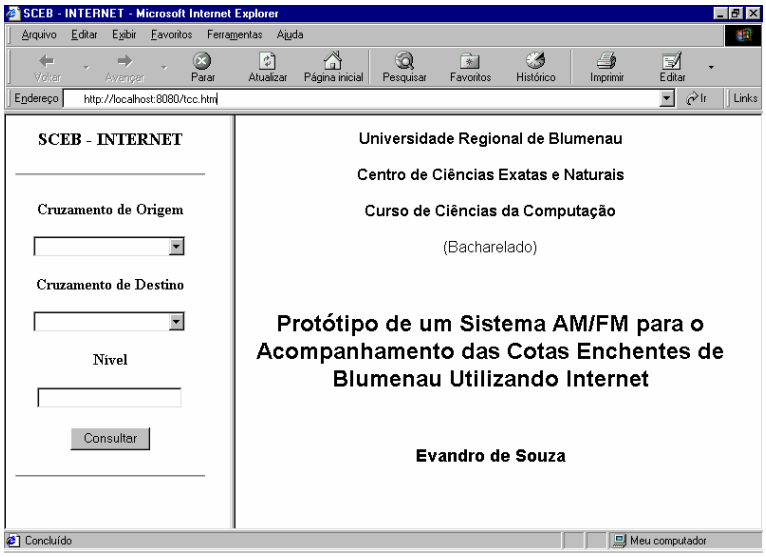


5.3.2 CARREGANDO O PROTÓTIPO

Para iniciar o uso do protótipo é necessário carregá-lo através de um *browser*, para isso basta acessar o endereço http://localhost:8080/Servlet. Convém observar que esse endereço utiliza um servidor *Web* rodando na máquina local.

Pode-se então visualizar a tela inicial do protótipo (figura 11). A interface inicial é composta de dois campos para a escolha dos cruzamentos de origem e destino, um campo para se digitar o nível do rio, um botão para iniciar o processo de busca pelo caminho.

Figura 11 : Tela inicial do protótipo



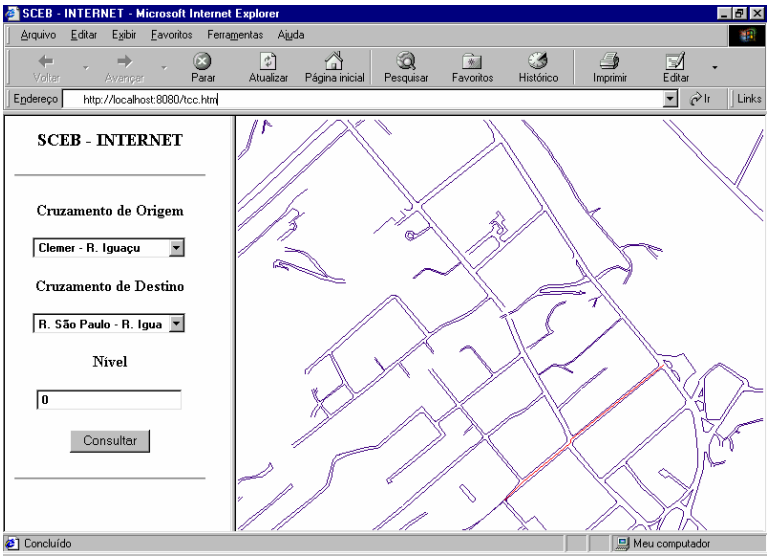
5.3.3 USANDO O PROTÓTIPO

Ao iniciar a utilização do protótipo o usuário tem basicamente apenas a opção de consultar um determinado caminho através do botão `Consultar`.

5.3.3.1 CONSULTAR CAMINHO

Para se consultar um caminho deve-se escolher o cruzamento de origem do primeiro *combo-box* e o cruzamento onde se deseja chegar no segundo *combo-box*. Digita-se o possível nível do rio e clica-se no botão `Consultar`, após algum tempo será desenhado um mapa no *frame* correspondente (figura 12), e também será mostrado um *frame* flutuante com o caminho selecionado descrito, facilitando assim a impressão caso se deseje.

Figura 12 : Tela do protótipo após consulta



Nota-se que existe um traço de outra cor desenhado no mapa, este traço corresponde ao caminho que deverá se seguido para que se chegue ao local desejado sem que se tenha problemas com alagamentos.

5.3.3.2 INTERAÇÃO COM O MAPA

Após o desenho das ruas é possível interagir com o mesmo. Esta interação corresponde a aplicação de *zoom* (figura 13) ou a movimentação do desenho (figura 14).

Figura 13 : Zoom das ruas selecionadas

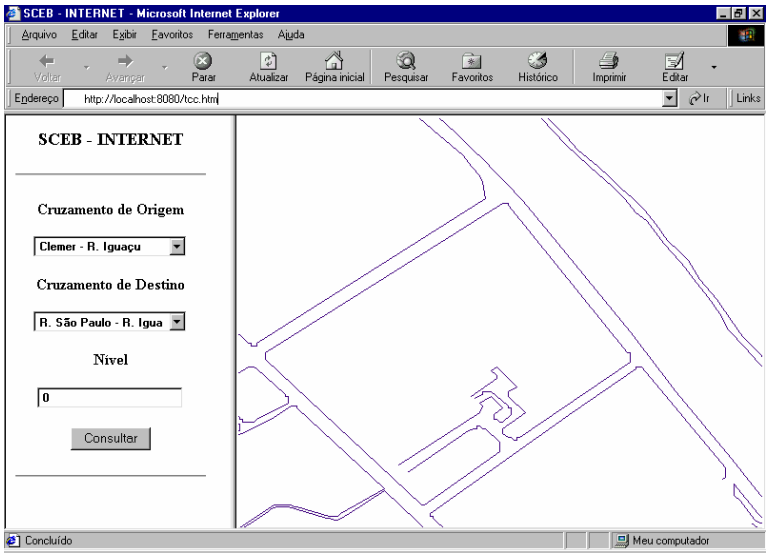
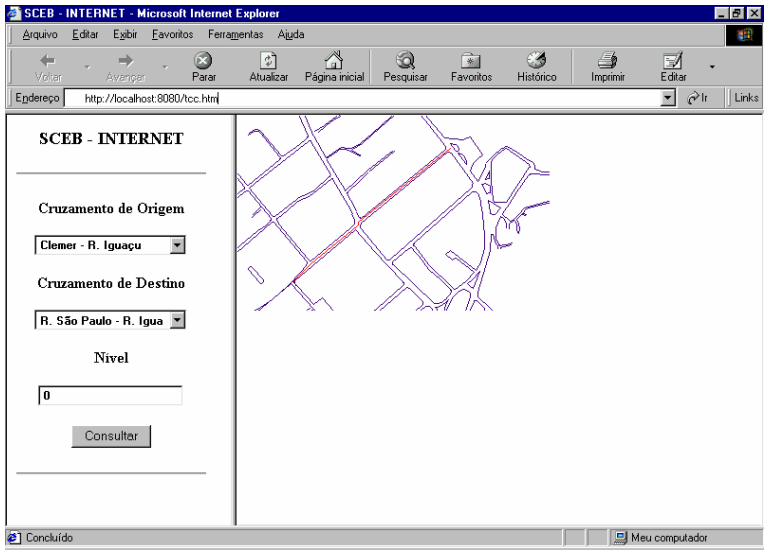


Figura 14 : Desenho das ruas movido para a esquerda e acima



O *zoom* pode se conseguido clicando-se com o botão direito sobre o desenho, com um click sobre a metade superior tem-se a aproximação, e sobre a metade inferior o afastamento.

A movimentação ocorre de maneira similar, para movimentar o desenho para a direita clica-se com o botão esquerdo sobre a metade direita do desenho. Para movimentar o mesmo para cima basta clicar sobre a metade superior e assim sucessivamente.

6 CONCLUSÕES

6.1 CONSIDERAÇÕES FINAIS

Os objetivos deste trabalho foram alcançados, uma vez que se tem implementado um protótipo de sistema AM/FM, baseado no Sistema Cruzamento, disponibilizado via *Web* através de *Servlets*.

Mesmo observando-se a deficiência dos *Servlets* em se lidar com interface gráfica, o que ocasionou pouca interação entre o usuário e o sistema, foi possível verificar que o uso de sistemas AM/FM na Internet é uma alternativa promissora.

6.2 LIMITAÇÕES

Devido as características desse trabalho, notou-se que o mesmo possui algumas limitações, como por exemplo:

- interatividade limitada, o usuário pode realizar apenas tarefas simples, como buscar caminho, zoom e movimentação do mapa;
- banco de dados relacional, o banco de dados utilizado não é orientado a objetos, tornando necessário realizar a "conversão" dos objetos para dados.

6.3 EXTENSÕES

Como extensão deste trabalho pode-se implementar mais opções de interatividade com o usuário, como por exemplo, buscar caminho através da seleção das ruas no mapa e cadastro de novos cruzamentos, ou ainda fazer uma projeção (previsão) das cotas para as próximas horas.

Pode-se ainda utilizar outras técnicas para desenvolvimento na Internet. E como o sistema é orientado a objetos converter o banco de dados relacional para um orientados a objetos.

GLOSSÁRIO

Geographic Information Systems (GIS):	Sistema de Informação Geográfica (SIG) Sistema baseado em computador, que permite ao usuário coletar, manusear e analisar dados georeferenciados. Um SIG pode ser visto como a combinação de hardware, software, dados, metodologias e recursos humanos, que operam de forma harmônica para produzir e analisar informação geográfica.
Land Information Systems (LIS)	Sistema de Informação Territorial Banco de dados que contém a descrição de caraterísticas físicas, administrativas e legais das parcelas de terreno de uma área definida; também pode ser chamado de Land Records System (LRS).
Automated Mapping / Facilities Management (AM/FM)	Mapeamento Automátizado / Gerência de Infra-Estrutura Tipo de sistema que integra funções de mapeamento automatizado e de gerência de equipamentos de infra-estrutura.
Computer-Aided Drafting and Design (CADD)	Projeto e Esboço Assistido por Computador Processo que, além de ter as propriedades de um CAD, permite a modelagem de relações gráficas e análise de relações lógicas.
Computer Aided Manufacturing (*CAM*)	Manufatura Assistida por Computador Uso de computador na automação industrial e na fabricação de produtos.

REFERÊNCIAS BIBLIOGRÁFICAS

[BEN1997] BENETT, Gordon. **INTRANETS**: como implantar com sucesso na sua empresa. Rio de Janeiro : Campus, 1997.

[BOC1995] BOCHENSKI, Barbara. **Implementando sistemas cliente/servidor de qualidade**. São Paulo : Makron Books, 1995.

[CAM1996] CÂMARA, Gilberto et al. **Anatomia de sistemas de informação geográfica** : trabalho para escola de computação. Campinas : Instituto de Computação, UNICAMP, 1996.

[DEF1998a] **Definições simplificadas**. Fator GIS On Line. Endereço eletrônico. http://www.fatorgis.com.br/geoproc/define_simp.htm, 1998.

[DEF1998b] **Definições técnicas**. Fator GIS On Line. Endereço eletrônico. http://www.fatorgis.com.br/geoproc/define_tecn.htm, 1998.

[FAT1999] **FATOR GIS on line**. Fator GIS On Line. Endereço eletrônico: http://www.fatorgis.com.br, 1999.

[FER1999] FERNANDES, Sandro Alencar. **Protótipo de um software de am/fm para o atendimento de ocorrências da cidade de rio do sul**. Trabalho de Conclusão de Cursos (Ciências da Computação), Universidade Regional de Blumenau, 1992.

[FUR1998] FURLAN, José Davi. **Modelagem de objetos através da UML - the unifield modeling language.** Sao Paulo : Makron Books, 1998.

[HOF1997] HOFF, Arthur Van. **Ligado em Java**. São Paulo : Makron Books, 1997.

[HOP1997] HOPSON, K. C. **Desenvolvendo Applets com Java**. Rio de Janeiro : Campus, 1997.

[INT1998] **Introdução ao geoprocessamento**. Courseware em Ciências Cartográficas. Endereço eletrônico: http://www.prudente.unesp.br/dcartog/arlete/hp_arlete/courseware/intgeo.htm, 1998.

[JAV2000a] **Java Servlet API**. Sun Microsystems. Endereço eletrônico: http://java.sun.com/products/Servlet, 2000.

[JAV2000b] **Java developer connection**. Sun Microsystems. Endereço eletrônico: http://developer.java.sun.com/developer, 2000.

[JEP1997] JEPSON, Brian. **Programando banco de bados em Java.** São Paulo : Makron Books, 1997.

[KRA1999] KRAMEL, Danilo. Desenvolvimento do sistema de cotas enchentes de Blumenau (SCEB). **Dynamis**, Blumenau : Universidade Regional de Blumenau, v. 7, n. 28, p.21-25, jul./set. 1999.

[NAU1997] NAUGHTON, Patrick. **Dominando o JAVA**: [guia autorizado da Sun Microsystems]. Sao Paulo : Makron Books, 1997.

[NAR1999] NARAYANAN, S. e LIU, Junhe. **Enterprise Java developer's guide**. New York : MacGraw-Hill, 1999.

[RAB1992] RABUSKE, Márcia Aguiar. **Introdução à teoria dos grafos**. Florianópolis : UFSC, 1992.

[REI1992] REIS, Dalton Solano dos. **Uma interface gráfica para o sistema cruzamento**. Trabalho de Conclusão de Cursos (Ciências da Computação), Universidade Regional de Blumenau, 1992.

[SWA1990] SWAIT JUNIOR, Joffre Dan. **Fundamento computacionais, algoritmos e estrutura de dados**. São Paulo : Makron Books, 1990.