UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

SISTEMA DE CONTROLE AUTOMOTIVO DE VELOCIDADE UTILIZANDO GPS

Autor: _____
Henrique Oscar Duran Lira

Autor: _____
Leonardo Vieira Gullo

Orientador: _____
Prof. Antônio Cláudio Gómez de Sousa, M.Sc.

Examinador: _____
Prof. Sérgio Barbosa Villas-Boas, Ph.D.

Examinador: _____
Prof. Aloysio de Castro P. Pedroza, D.Sc

DEL

Março de 2007

Dedicatória

Dedicamos este trabalho primeiramente aos nossos pais, que sempre nos apoiaram em todas as decisões. Dedicamos ainda às pessoas que de alguma forma contribuíram para que este longo caminho fosse concluído, entre elas nossos grandes amigos da faculdade, irmãos e nossas namoradas.

Agradecimento

Agradecemos primeiramente a Deus que nos deu forças para nunca desistir diante dos grandes desafios impostos pela faculdade.

Agradecemos também aos professores e funcionários do DEL pelo trabalho de ensino realizado, aos runges, ótimos amigos que fizemos na faculdade, e ao povo brasileiro por ter financiado os nossos estudos.

Resumo

A tecnologia está cada vez mais presente em todos os setores industriais e com o setor automobilístico não é diferente. Os carros incorporam as tecnologias mais atuais e isso vem se transformando até em diferencial competitivo para algumas grandes marcas.

Observando esse desenvolvimento dos automóveis e o aumento de acidentes relacionados ao excesso de velocidade principalmente entre os jovens brasileiros, optamos por desenvolver um sistema que monitorasse e controlasse a velocidade dos veículos utilizando a tecnologia de GPS (*Global Positioning System*), utilizando como referência o Código Brasileiro de Trânsito.

Palavras-chaves

GPS, Controle de Velocidade, GPRS, Micro-controlador, Código Brasileiro de Trânsito.

Índice

Índice de Figuras

Índice de Tabelas

Siglas

SisCAV	Sistema de Controle Automotivo de Velocidade
GPS	*Global Positioning System*
CONTRAN	Conselho Nacional de Trânsito
PRN	*Pseudo-Random Code*
UTM	Universal Transversa de Mercator
J2SE	*Java 2 Standard Edition*
J2EE	*Java 2 Enterprise Edition*
JSP	*Java Server Pages*
LCD	*Liquid Crystal Display*
GPRS	*General Packet Radio Services*
TCP	*Transmission Control Protocol*
IP	*Internet Protocol*
USB	*Universal Serial Bus*
RAM	*Random Access Memory*
MVC	*Model/View/Controller*
HTML	*Hypertext Markup Language*
JSP	*Java Server Pages*

1 Introdução

Os acidentes de trânsito estão entre as maiores causas de mortes não-naturais do planeta e por isso vêm sendo tratados como uma questão prioritária de saúde pública. Em 1998 estima-se que morreram mais de 1.170.000 pessoas em acidentes de trânsito no mundo todo.

No Brasil, estimam-se cerca de 750 mil acidentes e 27 mil brasileiros mortos em nossas vias urbanas e estradas anualmente, o que é realmente alarmante. O prejuízo com esses acidentes chega à marca de R$ 105 milhões anuais e as perdas principais são em produção, custos médicos e legais, previdência social, perdas materiais, despesas com seguros, entre outros.

Dentre as maiores causas de acidentes encontram-se o excesso de velocidade e o uso de bebidas alcoólicas e entorpecentes pelos motoristas. Estima-se ainda que cerca de 64% dos acidentes de trânsito sejam causados por falhas humanas, o que é muito preocupante.

Em nosso estado os dados de acidentes relacionados ao excesso de velocidade são ainda mais alarmantes, já que estima-se que no Rio de Janeiro 41% dos acidentes são causados por essa imprudência dos motoristas ao volante. O Rio de Janeiro é o estado com maior freqüência desse tipo de infração sendo seguido por São Paulo (28%) e Brasília (21%).

O principal aliado no combate aos acidentes que têm no excesso de velocidade sua causa principal é a fiscalização eletrônica de velocidade. Desde o início de sua implantação em 1997, o número de mortes relacionadas à violência no trânsito nas localidades que contavam com esse artifício caiu sensivelmente. Deve-se mencionar que o novo Código Brasileiro de Trânsito, em vigor desde 1998, também ajudou para que esses números diminuíssem. O fato é que o povo brasileiro sentiu nitidamente o peso das punições em seu próprio bolso, já que a fiscalização foi bastante intensificada.

Porém, uma recente decisão do CONTRAN, exigindo que sejam colocados avisos sobre o controle eletrônico de velocidade, foi tida como um retrocesso no combate aos acidentes de trânsito e na conscientização dos condutores, já que sabendo da posição dos radares os motoristas podem reduzir sua velocidade apenas nas proximidades dos mesmos, contornando a finalidade principal deste tipo de fiscalização.

Porém, deve-se ressaltar que a fiscalização não deve ser tomada como a única ferramenta no intuito de diminuir o número de acidentes. É necessário um projeto de conscientização dos motoristas para que abusos não sejam cometidos e para que esses números caiam ainda mais em longo prazo.

Foi devido a essa conjuntura atual da regulamentação do trânsito brasileiro que os autores do projeto em questão resolveram desenvolvê-lo. A idéia é apresentar uma alternativa para o controle de velocidade dos automóveis nas rodovias e vias urbanas utilizando o Sistema de Posicionamento Global e o Código Brasileiro de Trânsito como referência.

Dessa forma, foge do contexto do projeto uma discussão ética ou legal sobre a aplicabilidade do mesmo.

A seguir apresentaremos a estrutura do projeto adotando a seguinte divisão: o capítulo 2 apresentará as principais tecnologias envolvidas no projeto; no capítulo 3 abordaremos uma visão geral da solução, apresentando os módulos envolvidos e seus componentes; o capítulo 4 detalhará alguns aspectos importantes no desenvolvimento de cada módulo; o capítulo 5 mostrará alguns recursos adotados para que o projeto fosse desenvolvido de maneira modular e para garantir sua manutenibilidade; no capítulo 6 teremos uma breve descrição dos resultados; no capítulo 7 temos a conclusão apontando como a idéia aqui apresentada pode ser benéfica para a sociedade e por fim a bibliografia. Além dos sete capítulos teremos ainda em

forma de Apêndice a Especificação de Requisitos de Software onde estão detalhadas as etapas e módulos do desenvolvimento dos softwares que integram este projeto e em anexo parte do Código Brasileiro de Trânsito que foi usado como referência (artigos 60, 61 e 62).

2 Fundamentação Teórica

Serão apresentadas agora as principais ferramentas e tecnologias utilizadas durante o desenvolvimento do projeto.

2.1 Global Positioning System (GPS)

O Sistema de GPS (*Global Positioning System)* é um sistema de posicionamento por satélites usado sobretudo para determinar a posição de um receptor que pode estar em órbita ou na superfície terrestre.

Esse sistema foi criado pelo Departamento de Defesa dos Estados Unidos e para sua utilização necessitamos apenas de um receptor que consiga captar o sinal emitido pelos satélites que compõem o chamado Segmento Espacial do Sistema GPS. Além desse segmento podemos identificar ainda outros dois: o Segmento de Controle e o Segmento de Usuário.

O Segmento Espacial é constituído por 24 satélites divididos em 6 órbitas. Esses satélites percorrem a órbita de nosso planeta a cada 12 horas e estão localizados a aproximadamente 22 000 Km de altitude. Vários pontos da Terra podem ser visualizados por 6 a 10 satélites em um dado momento, o que nos fornece redundância já que apenas 4 deles são necessários para uma localização tridimensional (latitude, longitude e altitude). A seguir estão representadas as órbitas desses satélites em torno da Terra:

Figura 1 – Distribuição dos Satélites ao Redor da Terra

O Segmento de Controle nada mais é que o conjunto de estações em terra dispersas ao longo da Zona Equatorial, e que têm, entre outras responsabilidades, a sincronização dos relógios atômicos dos satélites, a monitoração de suas órbitas e a atualização dos dados de almanaque transmitidos pelos mesmos.

O Segmento de Usuário é composto pelos receptores GPS que decodificam as transmissões do sinal de código e fase de vários satélites e calculam sua posição tendo como base a distância até eles.

2.1.1 Transmissão e Cálculo de Posição

Para o cálculo de posição do receptor GPS são utilizados os sinais de rádios enviados exatamente ao mesmo tempo por todos os satélites que constituem o Segmento Espacial. A potência de transmissão é de apenas 50 Watts e o cálculo é feito através do lapso entre a emissão do sinal e sua recepção. Este processo é altamente preciso já que todos os receptores GPS em qualquer lugar no mundo sempre mostrarão a mesma hora, minuto, segundo e até milissegundo, já que cada satélite possui um relógio atômico de altíssima precisão.

Assim, sabendo o tempo que o sinal demora para chegar do satélite até o receptor e a velocidade de propagação do sinal, pode-se determinar a distância entre eles. Como dito anteriormente, de posse da posição de apenas 4 satélites e das distâncias deles até o receptor é possível calcular a coordenada tridimensional deste ponto. Na figura 2 este processo é ilustrado:

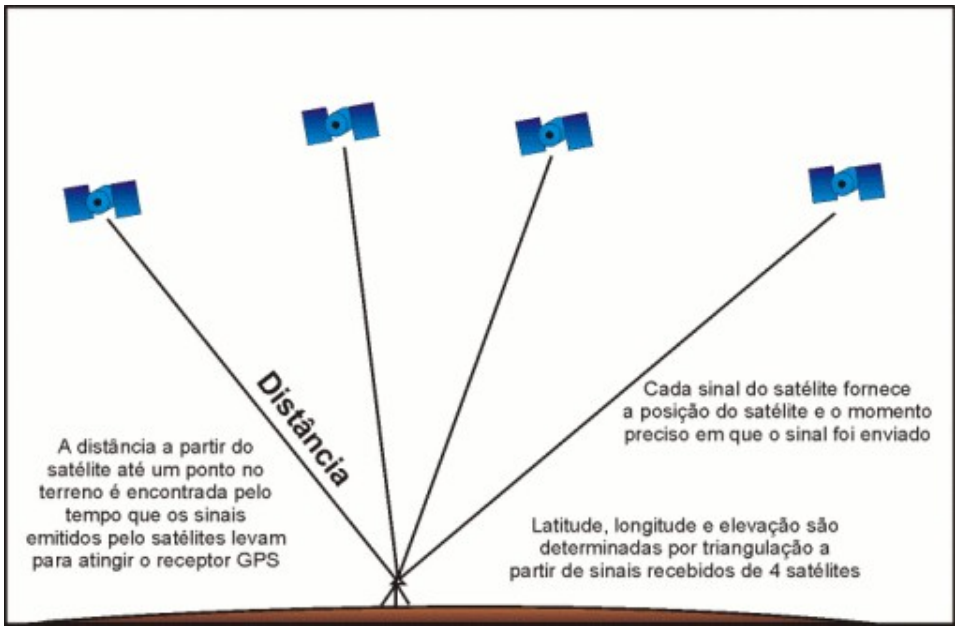


Figura 2 – Ilustração do Cálculo de Posição do Receptor

Uma vez determinada a posição do usuário, o receptor GPS pode calcular outras informações importantes e úteis para navegação, como velocidade, direção, rumo, distância ao destino, hora do nascer e do pôr-do-sol, entre outras coisas.

Em nosso projeto usaremos apenas os dados de latitude, longitude e velocidade do receptor GPS.

2.1.2 Receptores GPS

Existem diversos tipos de receptores GPS, desde os projetados para serem portáteis até os mais precisos e sofisticados, usados geralmente em navios e aviões. Os receptores podem ser verdadeiros computadores sendo capazes de tratar e armazenar grande volume de dados e

até enviar esses dados para outros receptores ou computadores. Alguns modelos possuem mapas bastante detalhados em suas memórias.

Em nosso projeto será usado o GPS Garmin eTreX Legend, que possui antena interna e 12 canais paralelos.

Alguns fatores atmosféricos e fontes de erro podem afetar a precisão de receptores GPS. Os receptores de GPS GARMIN são precisos numa faixa de 10 a 15 metros em média.

2.1.3 Sistemas de Coordenadas

Os sistemas de coordenadas são baseados em padrões de quadriláteros sobrepostos aos mapas que permitem a localização de qualquer ponto da superfície terrestre. O sistema de coordenadas mais utilizado é o Latitude/Longitude. Nele são usadas como referência linhas imaginárias como a Linha do Equador, que é a referência de divisão em Hemisfério Norte (N) e Hemisfério Sul (S), e o Meridiano de Greenwich que divide a Terra em Hemisfério Oeste (W) e Hemisfério Leste (E).

Além dessas duas linhas imaginárias existem ainda os chamados Paralelos de Latitude, que são linhas imaginárias paralelas à do Equador (que por convenção é a linha de 0º de Latitude) e os Meridianos de Longitude, linhas paralelas ao Meridiano de Greenwich (por convenção 0º de Longitude). Também temos, por convenção, que o Pólo Norte está localizado na Latitude 90º Norte e o Sul na Latitude 90º Sul. Assim para localizarmos um ponto qualquer na Terra basta sabermos as informações referentes ao Paralelo e ao Meridiano em que se encontra. Como exemplo podemos citar o último pedido de socorro do navio Titanic, que partiu das coordenadas de latitude 41º e 45' acima do Equador (Hemisfério Norte) e de longitude 050º e 14' a oeste do Meridiano de Greenwich (Hemisfério Oeste). Assim suas coordenadas, no sistema Latitude/Longitude, eram: N 41º 45' W 050º14'.

2.2 J2SE

A plataforma J2SE (*Java 2 Standard Edition)* contém uma série de especificações com funcionalidades distintas. No projeto utilizamos os conceitos de:

- JDBC (*Java Database Connectivity*): usado na conexão ao banco de dados.
- *Servlets*: foi utilizado no desenvolvimento da parte *Web*, já que possui uma API que apresenta de maneira bastante simples os recursos do servidor *Web.*
- JSP (*Java Server Pages*): facilitou o desenvolvimento de conteúdo dinâmico.

2.3 wxWidgets

Em nosso projeto, wxWidgets foi usado por ser uma *framework* que possui elementos gráficos básicos que facilitam a construção de interfaces com o usuário, *threads* e outros utilitários. Ela nos permite que um programa seja executado em diversas plataformas sem que haja modificações no código do mesmo.

Podemos ainda descrever wxWidgets como um utilitário nativo que fornece certa abstração de código, o que a torna mais rápida e com visual mais bem adaptado.

2.4 SQLAPI++

SQLAPI++ é uma biblioteca de C ++ para acessar bases de dados diversas (Oracle, SQL Server, DB2, Sybase, Informix, InterBase, SQLBase, MySQL, PostgreSQL e ODBC). Utiliza APIs nativas da base de dados alvo fazendo com que as aplicações desenvolvidas com essa biblioteca rodem mais rápida e eficientemente. SQLAPI++ fornece também uma interface de baixo nível que permite que os colaboradores alcancem características específicas da base de dados. Encapsulando a API do banco, a SQLAPI++ age como o *middleware* e torna a base de dados portável.

3 Visão Geral do Projeto

Como citado anteriormente, a idéia central do projeto é criar um sistema que controle a velocidade de um dado automóvel, adequando-a, quando necessário, à velocidade máxima permitida na via em que ele estiver trafegando. Para isso utilizamos um receptor GPS instalado em um veículo. O receptor GPS é responsável pela localização do automóvel, fornecendo sua latitude, longitude e sua velocidade atual. Esses dados são enviados a um servidor central através de um modem GPRS (que foi emulado por nós através de um software instalado em um *laptop*). O servidor então trata os dados a fim de localizar o veículo em um mapa das ruas de nossa cidade, que além da localização latitude/longitude de cada rua possui ainda a velocidade máxima permitida nas mesmas, segundo o Código Brasileiro de Trânsito.

Após feita a localização do veículo o servidor envia de volta para o modem GPRS o nome da rua em que o veículo se encontra e a velocidade máxima permitida na via. Essas informações são então mandadas a um micro-controlador, que recebe também diretamente do modem a velocidade atual do veículo, e fica encarregado de fazer o cálculo para tomar a decisão se essa velocidade deve ser diminuída ou não. O micro-controlador foi utilizado já que adicionamos dois sinais de entrada extras, que atuam como sensores de chuva e luminosidade. Esses sensores influenciam o cálculo do ajuste de velocidade já que definem com mais precisão o ambiente em que o veículo se encontra e ajudam a tomar uma decisão mais embasada e apurada da velocidade na qual o veículo deve trafegar com segurança. São disponibilizadas ainda em um *display* de LCD a velocidade atual do veículo e a velocidade máxima permitida. Há a opção ainda de o usuário interagir com o sistema requisitando o nome da rua em que se encontra, o que também é apresentado no *display* de LCD.

O projeto engloba também um *log* das informações do veículo que poderão ser acessadas pela Internet. Para isso, um banco de dados foi criado para armazenar informações como data, hora e localização do veículo. Assim o usuário pode consultar onde seu veículo esteve na data e hora requerida. A seguir encontra-se um esquema da arquitetura básica do projeto.

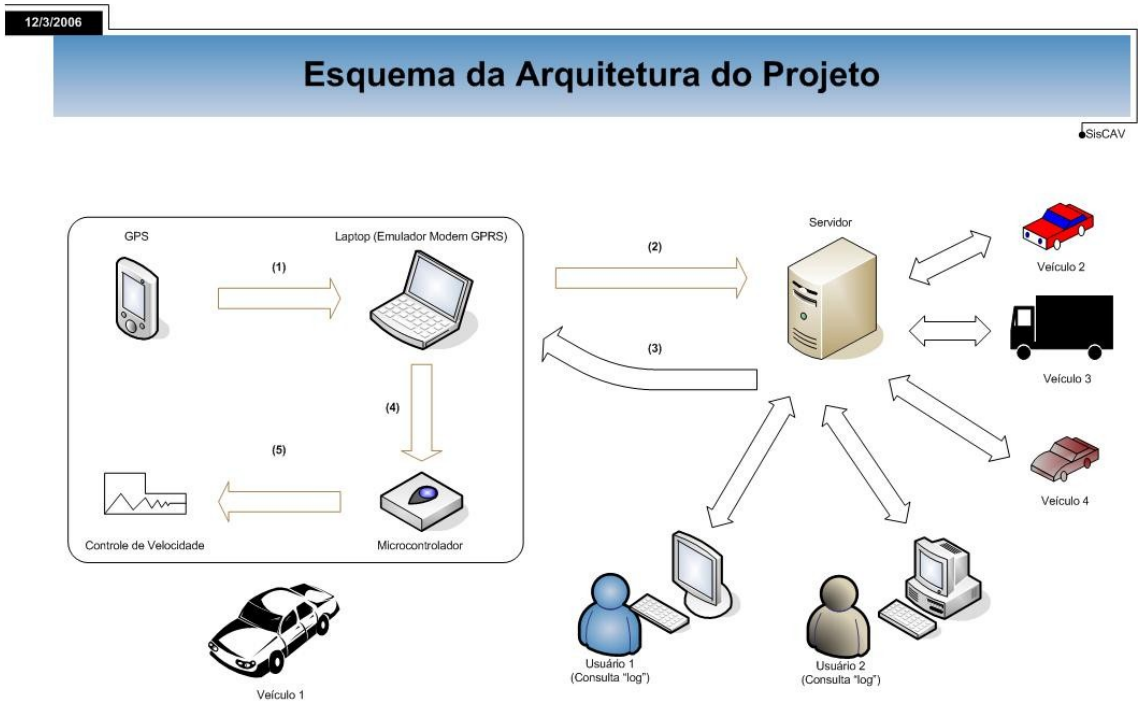


Figura 3 – Esquema da Arquitetura do Projeto

A comunicação entre os componentes, ilustrada acima, será detalhada no próximo capítulo. A figura acima apenas ilustra a arquitetura funcional do sistema, já que na prática por tratar-se de um protótipo para fins demonstrativos, o servidor também funcionará no mesmo laptop que emulará o modem GPRS, porém como a comunicação entre os dois módulos foi desenvolvida via *socket* TCP/IP, os módulos podem estar isolados ou na mesma máquina.

Devemos ressaltar que em nosso projeto não está incluída a implementação do sistema de controle mecânico da velocidade do veículo. Utilizamos uma saída que simulava um sinal que controlava ou não a velocidade do mesmo respeitando as regras supracitadas. Como citado também, o modem GPRS foi emulado através de um software para que o projeto tivesse um custo mais baixo. Assim, muitas vezes iremos nos referir ao Módulo Emulador GPRS como modem.

Para o desenvolvimento, o projeto foi dividido em 4 módulos principais: o Módulo Localizador, que engloba o software instalado no servidor que localiza o veículo no mapa e grava os *logs* no banco de dados; o Módulo Emulador GPRS, que é constituído pelo software que emula o modem GPRS; o Módulo Limitador, que compreende o circuito limitador e a programação do micro-controlador; e por último o Módulo Web responsável pela visualização dos *logs* gravados no banco de dados. Esses módulos serão abordados com detalhes no próximo capítulo.

Antes de iniciarmos o desenvolvimento do projeto foi necessário pesquisar na Internet um mapa que contivesse a latitude e longitude de cada rua de nossa cidade. Após muita pesquisa resolvemos utilizar o mapa encontrado no site http://www.tracksource.org.br/ . Como esse mapa estava no formato *.map, binário, que dificultava sua manipulação, utilizamos um software livre denominado MapDekode que o convertia para o formato*.dbx, do tipo texto. O mapa possuía várias informações, muitas delas desnecessárias para nossa aplicação.

Assim as duas sessões principais que utilizamos foram: [DEF] e [Linien] que são descritas segundo o MapDekode da seguinte maneira:

Section [**DEF**]
Name= String, the name of the **IMG/DBX** file (only comment)
Text= String, the **map name**, used in MapSource
Max Nord= Single, **boundary north** (in decimal degree)
Max Sued= Single, **boundary south** (in decimal degree)
Max Ost = Single, **boundary east** (in decimal degree)
Max West= Single, **boundary west** (in decimal degree)

Section [**Linien**]
one line for the first point of the polygon and one line for each delta point (1 up to 255)
(for details see Section [Punkte])
L00001 String, **line** name, six characters start with "**L**"
you can have double name without problems
492023686 Integer, **north/ south** coordinate absolutely decimal = 41.24094894° N
293377929 Integer, **east/ west** coordinate absolutely decimal = 24.59065394° E
20 Integer, **line type** decimal, Range from 1 to 63
2 Integer, **maximum zone** decimal, you can see this line in zone 0,1 and 2
000493 String, label **offset**, only for MapDekode; If you create a DBX by hand use "FFFFFF"
V 5.2.c 21/34 13.10.03
Railroad String, **label** of the line (for special characters see A11 Special Codes for labels in MapDekode)
For lines type 32 to 37 (0x20 to 0x25 = land / depth contour) this is the altitude in feet
D00001 String, **delta** point, six characters start with "**D**" from 1 to 255
-64615 Integer, **north/ south** coordinate relative decimal
South = -64615 / (2^30/90) = 0.005415966°S
Range depends on the used zone factor (maximum map size): e.g.: ZF = 17 is +/- 1.4° size
So range is +/- 16702650 = +/- 1.4°
22513 Integer, **east/ west** coordinate relative decimal
East = 22513 / (2^30/90) = 0.002138474°E
Range depends on the used zone factor (maximum map size): e.g.: ZF = 17 is +/- 1.4° size
So range is +/- 16702650 = +/- 1.4°

Como podemos perceber a sessão [DEF] delimita os pontos máximos do mapa, que denominamos MaxNorte, MaxSul, MaxLeste, MaxOeste. Já a sessão [Linien] nos diz o nome de cada rua e os pontos (expressos em forma de latitude e longitude) que compõem essa via. Esses pontos foram chamados por nós de *waypoints* e serão mais bem explicados quando abordarmos o Módulo Localizador.

Além dessas informações, necessitávamos ainda estipular uma velocidade máxima para circulação nas mesmas. Isso foi feito levando em consideração o Código Brasileiro de Trânsito que estipula nos Artigos 60, 61 e 62 (ver Anexo A) as velocidades permitidas em cada tipo de via. Assim, esses limites foram colocados manualmente para cada *waypoint*. Como cada via pode possuir vários *waypoints*, uma mesma via pode possuir vários limites de velocidade distintos, o que reflete a realidade, já que em muitas rodovias, ou até mesmo ruas, os limites de velocidade podem mudar em certo ponto.

Como exemplo, uma parte do mapa está ilustrada a seguir. Nela podemos observar os pontos máximos do mapa, dado pelo cabeçalho [DEF], e as localizações de alguns *waypoints* (já nas respectivas vias), a partir do cabeçalho [Linien], junto com a velocidade máxima permitida nos mesmos.

"[DEF]"
"Name=","C:\Tracksource\Municipal\33304550.img"
"Text=","RJ-Rio de Janeiro"

"Max Nord=",-22.74973
"Max Sued=",-23.14592
"Max Ost=",-43.10984
"Max West=",-43.80352

"Country=","Brasil"
"State=","RJ"
"City=","Rio de Janeiro"

"[Linien]"
"L00001",-274313471,-516033535,100,4,1,"00B516","E DA GAVEA"
"D00001",-512,1792,100
"D00002",4864,15104,100
"D00003",768,11520,100
"D00004",0,3584,100
"D00005",-768,3072,100
"L00002",-274427135,-516294655,100,2,2,"00B521","ELEVADO DA JOATINGA"
"D00001",-3584,-3584,100
"D00002",-4864,-5888,100
"L00003",-274555135,-516577279,80,4,1,"00B535","AV GILBERTO AMADO"
"D00001",1792,1536,80
"D00002",0,0,80
"L00004",-274537471,-516706047,100,4,1,"0044C1","PC PROF JOSE BERNARDINO"
"D00001",13056,2816,100
"D00002",1536,2304,100
"D00003",-2048,10496,100
"D00004",-2048,1792,100
"D00005",-13568,-3072,100
"D00006",-1280,-2304,100
"D00007",2048,-9984,100
"L00005",-274523135,-516699391,60,5,0,"00B547","R MANOEL BRASILIENSE"
"D00001",20480,6656,60
"D00002",13056,4352,60
"D00003",2304,0,60
"L00006",-274525695,-516689663,60,5,0,"00B55C","R ALDO BONADEI"
"D00001",20736,6400,60
"D00002",16896,5120,60
"L00007",-274486783,-516698367,60,5,0,"00B56B","R PEDRO LAGO"
"D00001",-13312,-4096,60
"D00002",-14592,-4608,60
"L00008",-274486015,-516708351,60,5,0,"00B578","R G ALBUQUERQUE"
"D00001",-17664,-5632,60
"L00009",-274489343,-516647679,80,4,1,"00B588","AV ERICO VERISSIMO"

"D00001",-1792,1024,80
"D00002",-1536,0,80
"D00003",-2048,-512,80
"D00004",-2048,-256,80
"D00005",-3584,768,80
"D00006",-26368,-42240,80
"L00010",-274500351,-516646655,80,4,1,"00B59B","AC AV MIN IVAN LINS"
"D00001",-1536,1792,80
"D00002",-1024,2048,80
"D00003",-512,2304,80
"D00004",512,2304,80
"D00005",256,2048,80
"D00006",-256,1792,80
"D00007",-768,1280,80
"D00008",-1280,1536,80
"L00011",-274504959,-516631551,80,4,1,"00B588","AV ERICO VERISSIMO"
"D00001",-1280,1280,80
"D00002",0,0,80
"L00012",-274486783,-516695551,80,4,1,"00B5AF","AV ARMANDO LOMBARDI"
"D00001",1536,-3328,80
"D00002",1536,-1024,80
"D00003",1280,0,80
"D00004",768,768,80
"D00005",768,1024,80
"L00013",-274478335,-516731903,80,5,0,"00B5C3","AV NUTA JAMES"
"D00001",19200,-11520,80
"L00014",-274477311,-516739839,80,5,0,"00B5C3","AV NUTA JAMES"
"D00001",2304,2816,80
"D00002",1536,768,80
"D00003",2304,0,80
"L00015",-274519551,-516677375,60,5,0,"00B5D1","R PEDRO BOLATO"
"D00001",12288,3584,60
"D00002",18688,5632,60
"L00016",-274510847,-516663551,60,5,0,"00B5E0","R PAULO MAZZUCCHELLI"
"D00001",21760,5632,60
"L00017",-274500095,-516702463,60,5,0,"00B5F5","R HENRIQUE DE MOURA COSTA"
"D00001",-2560,9728,60
"D00002",-2304,9472,60
"D00003",-2304,9472,60

Feita a definição do mapa, veremos agora o desenvolvimento de cada módulo.

4 Detalhamento do Projeto

Os módulos apresentados e a comunicação entre eles serão agora abordados e alguns trechos dos códigos implementados serão discutidos.

A comunicação física entre os módulos foi feita através da porta serial RS232. Em nosso caso, como utilizávamos um laptop sem esse tipo de porta, foi necessário adquirir um cabo que convertia uma porta USB em RS232. Como o padrão RS232 é um padrão de comunicação full-duplex, podíamos transmitir e receber dados simultaneamente sem problemas de colisão. O GPS deveria mandar dados para o laptop onde estavam instalados o Módulo Emulador GPRS, o Módulo Localizador e o Módulo Web. Da mesma forma, o computador deveria enviar dados para o Módulo Limitador. Assim, o seguinte esquema de conexões foi traçado:

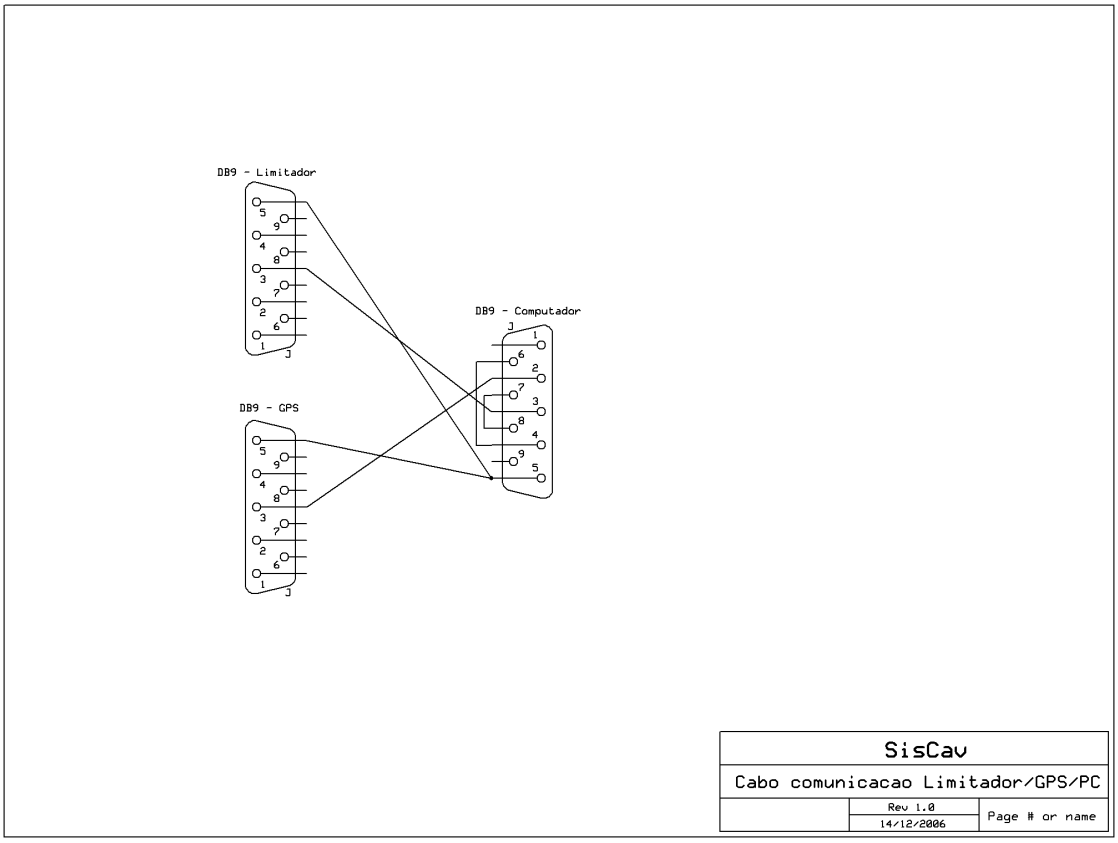


Figura 4 – Esquema Físico de Comunicação entre os Módulos

O cabo liga o pino de recepção Rx do laptop ao pino de transmissão Tx do GPS e o pino de transmissão Tx do laptop ao pino de recepção Rx do limitador. O pino Rx do GPS e Tx do limitador ficam desligados já que não são usados. Todos os pinos GND (terra) também são interligados.

Após vermos a conexão física entre os módulos abordaremos cada um individualmente.

4.1 Módulo Emulador GPRS

O Módulo Emulador GPRS deve ser o primeiro a ser detalhado já que é nele que começa o ciclo de informações.

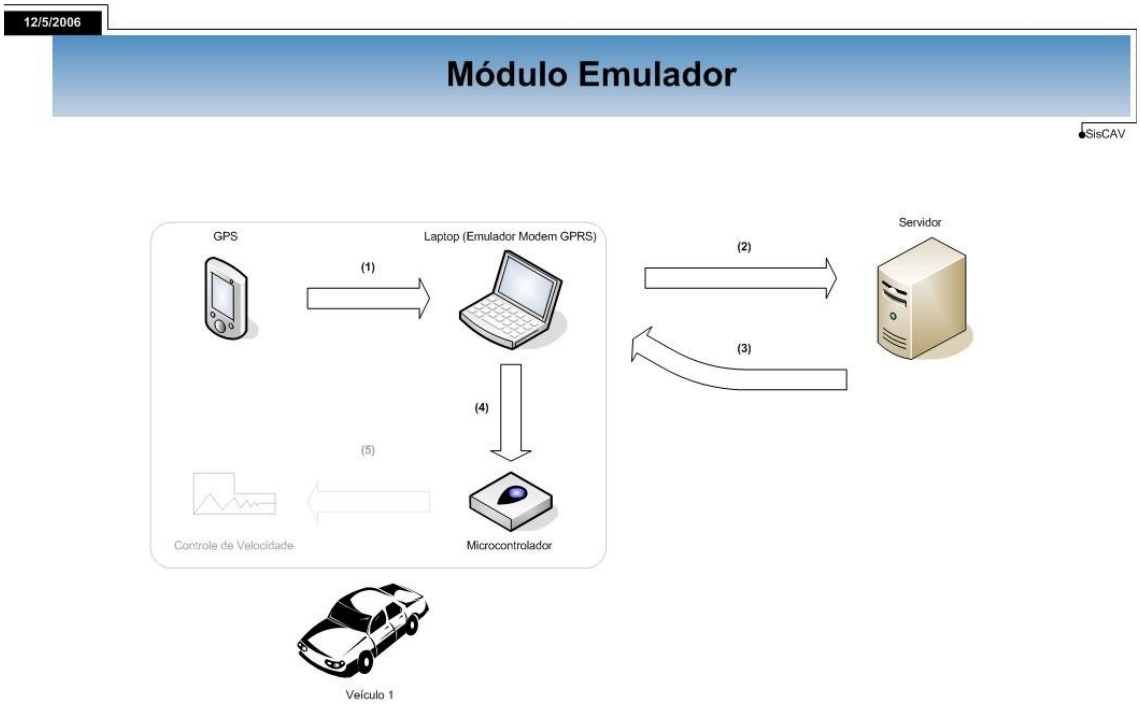


Figura 5 – Ilustração da Comunicação do Modo Emulador GPRS com Outros Módulos

Quando o modem é inicializado ele se conecta ao servidor e envia o ID do equipamento instalado no veículo. Esse ID será usado como chave identificadora, ou seja, cada veículo possui um ID associado a ele. A tela inicial do Módulo Emulador GPRS é ilustrada na figura 6 a seguir.

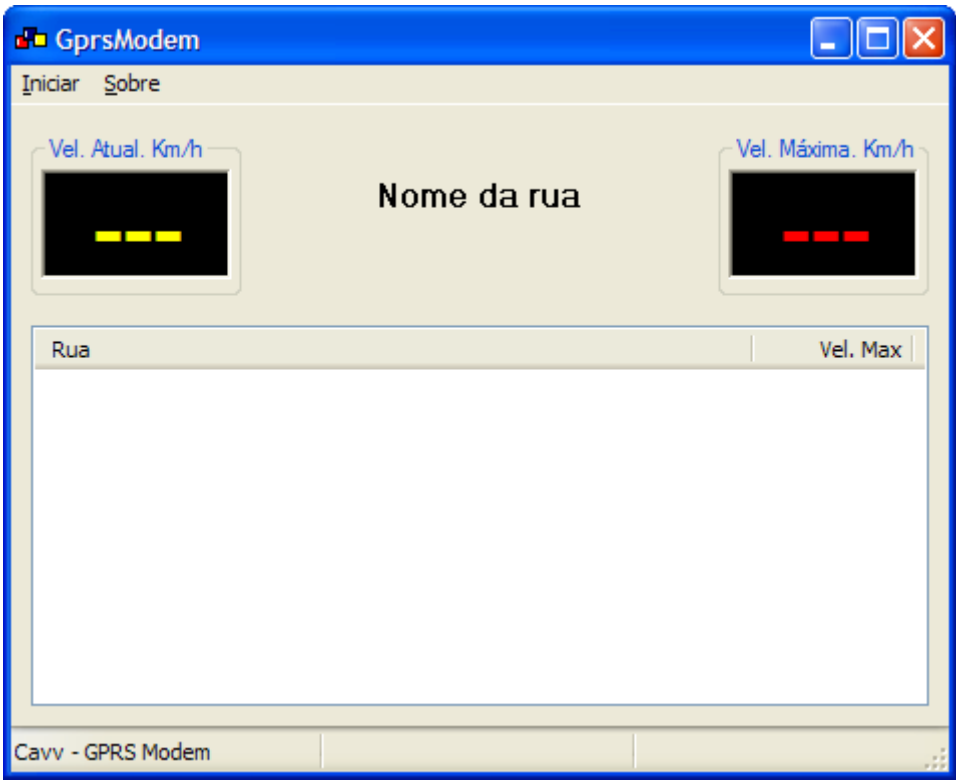


Figura 6 – Tela do Emulador GPRS

Feito isso o modem passa a receber do receptor GPS os dados de latitude, longitude e velocidade do veículo. Esse fluxo é indicado na figura 5 por (1). O GPS manda esses dados a cada um segundo, mas para diminuir o tráfego de dados foi estabelecido um parâmetro na configuração do modem que indica a cada quantas mensagens recebidas do GPS o modem deve enviar uma mensagem ao servidor. Esse artifício não prejudica a localização do veículo já que ela não varia rápida e bruscamente. O mesmo não acontece com a velocidade, que pode ter uma variação grande num curto intervalo de tempo. Dessa forma, a cada mensagem enviada pelo GPS ao Módulo Emulador GPRS, a velocidade do veículo é enviada ao micro-controlador para que o mesmo controle-a com maior precisão utilizando nesse controle a informação de velocidade máxima mais recentemente guardada.

O protocolo utilizado nessa comunicação é o padrão NMEA 2.0. Esse protocolo possui várias mensagens e o GPS as envia a cada um segundo. Entretanto, a única mensagem usada é a "$GPRMC" que possui o seguinte padrão:

Tabela 1 - Formato de Dados da Mensagem $GPRMC

Name	Example	Units	Description
Message ID	$GPRMC		RMC protocol header
UTC position	161229.487		hhmmss.sss
Status	A		A=data valid or V data not valid
Latitude	3723.2475		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12158.3416		dddmm.mmmm
E/W	W		E=east or W=west
Speed Over Ground	0.13	knots	
Course Over Ground	309.62	degrees	True
Date	120598		ddmmyy
Magnetic Variation		degrees	E=east or W=west
Checksum	*10		
<CR><LF>			End of message termination

Por essa tabela observamos que usando apenas essa mensagem conseguimos pegar todos os dados que nos são necessários. Porém, a latitude e longitude passadas pelo GPS (no formato "graus, minutos e fração de minutos") estão em uma unidade diferente daquela usada em nosso mapa (que estão no formato decimal usando um inteiro de 4 Bytes). Assim, fazem-se necessárias duas conversões: a primeira para passar a latitude e a longitude do formato "graus, minutos e fração de minutos" para "graus e fração de graus". Para isso basta dividirmos os minutos da primeira forma por 60. Dessa maneira obtemos a latitude na forma dd,dddddd e a latitude na forma ddd,dddddd. Feito isso devemos passar agora ambos para o formato decimal usando a fórmula $X * (\frac{2^{30}}{90})$, onde X recebe o valor da latitude ou da longitude que se deseja converter.

Além da conversão da latitude e longitude há ainda a conversão da velocidade, já que o GPS a informa em nós, e no mapa ela está disposta em Km/h. Assim, basta multiplicarmos a velocidade em nós por 1,852 para obtermos a velocidade já em Km/h. Parte do código que trata os dados provenientes do GPS é exemplificado a seguir:

```
#include <math.h>

#include "wx/wx.h"
#include "cavGpsNMEAMsgTranslate.h"
#include "cavAuxFunctions.h"
#include "cavErrorLog.h"
#include "cavConst.h"


enGpsTranlateCode CGpsNMEAMsgTranslate::Execute(const char* gpsMsg,

      StGpsPositionMsg &gpsPosition)
{
      try
      {

            gpsPosition.code= eGpsPosition;
            wxString result;

            //lê o tipo de mensagem
```

```
            if(!StringNTok(gpsMsg, result, ',', 0))
                  return eTransGpsError;

            //só estamos interessados na mensagem "$GPRMC"
            if(result != "$GPRMC")
                  return eTransNotRelevantGpsMsg;


            //lê se a mensagem é válida
            if(!StringNTok(gpsMsg, result, ',', 2))
                  return eTransGpsError;

            //se diferente de "A" a mensagem é inválida
            if(result != "A")
                  return eTransGpsError;


            //lê a latitude -> ddmm,mmmmm
            if(!StringNTok(gpsMsg, result, ',', 3))
                  return eTransGpsError;

            //transforma a latitude de ddmm,mmmmm para decimal
            double degree;
            double minute= modf(atof(result.c_str())/100, &degree);
            gpsPosition.latitude=       ((degree+((minute/60)*100))       *
degreeToDecimal);


            //lê se é Norte ou Sul
            if(!StringNTok(gpsMsg, result, ',', 4))
                  return eTransGpsError;

            //se for Sul a latitude é negativa
            if(result == "S")
                  gpsPosition.latitude*=-1;


            //lê a longitude -> ddmm,mmmmm
            if(!StringNTok(gpsMsg, result, ',', 5))
                  return eTransGpsError;

            //transforma a longitude de ddmm,mmmmm para decimal
            minute= modf(atof(result.c_str())/100, &degree);
            gpsPosition.longitude=       ((degree+((minute/60)*100))       *
degreeToDecimal);


            //lê se é Leste ou Oeste
            if(!StringNTok(gpsMsg, result, ',', 6))
                  return eTransGpsError;

            //se for Oeste a longitude é negativa
            if(result == "W")
                  gpsPosition.longitude*=-1;


            //lê a velocidade do veículo em nós
            if(!StringNTok(gpsMsg, result, ',', 7))
                  return eTransGpsError;

            //transforma a velocidade de nós para km/h
```

```
            gpsPosition.carSpeed= (atof(result)*knotsToKmh);


            return eTransGpsPosSpeed;

      }
      catch(...)
      {
            CAV_ERROR_LOG("Erro Desconhecido!")
      }

      return eTransGpsError;

}
```

Agora, esses dados estão prontos para serem enviados ao servidor. Esse fluxo está indicado na figura 7 por (2).

Feita a localização, o servidor envia de volta o nome da rua em que o veículo está e a velocidade máxima permitida (3). Em seguida o emulador GPRS envia esses dados ao Módulo Limitador para que o mesmo proceda com o controle da velocidade do automóvel.

O Módulo Emulador GPRS foi desenvolvido em C++ usando a biblioteca wxWidget.

4.2 Módulo Localizador

Esse módulo é constituído pelo software que localiza o veículo no mapa e grava suas informações, gerando um *log*, no banco de dados. Como demonstrado abaixo ele se comunica com o Módulo Emulador do modem GPRS enviando e recebendo dados, e com o banco de dados.

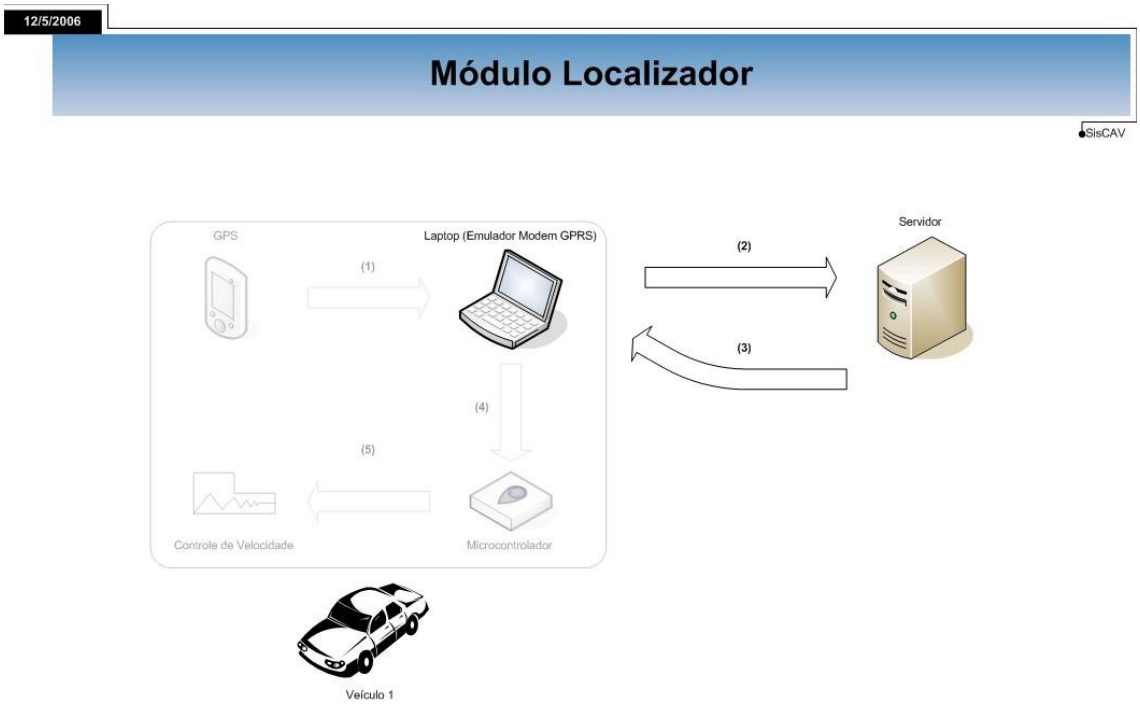


Figura 7 – Interação do Módulo Localizador com Módulo Emulador

Para cada veículo que se conecta o servidor cria uma *thread* específica para servir este cliente. Como dito anteriormente, ao se estabelecer a conexão o servidor recebe o ID do equipamento. Em seguida esta *thread* fica aguardando os dados de latitude, longitude e velocidade do veículo que vem do emulador GPRS. Com esses dados e dispondo do mapa já exemplificado aqui, o programa calcula a rua mais próxima do ponto em questão. Para isso é feito um cálculo de geometria analítica plana bastante comum, o cálculo da distância de um ponto a uma reta. O cálculo é feito da seguinte forma:

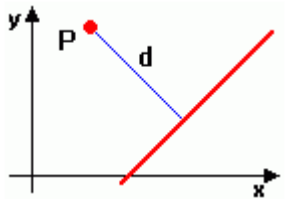


Figura 8 – Ilustração do Cálculo da Distância de Ponto a Reta

Sendo **P** um ponto (X_o , Y_o), e uma reta **r** definida por ax+by+c=0 , temos que a distância do ponto **P** a essa reta é dada por:

$$d(P,r) = \frac{|aX_o + bY_o + c|}{\sqrt{a^2 + b^2}}$$

Em nosso caso, X_o e Y_o representam respectivamente a latitude e a longitude do veículo que é mandada pelo GPS e chega através do emulador GPRS. Em seguida devemos formar uma equação de reta r usando os *waypoints* presentes no mapa. Isso deveria ser feito para cada *waypoint*, porém devemos ter cuidado para que a situação ilustrada a seguir não aconteça:

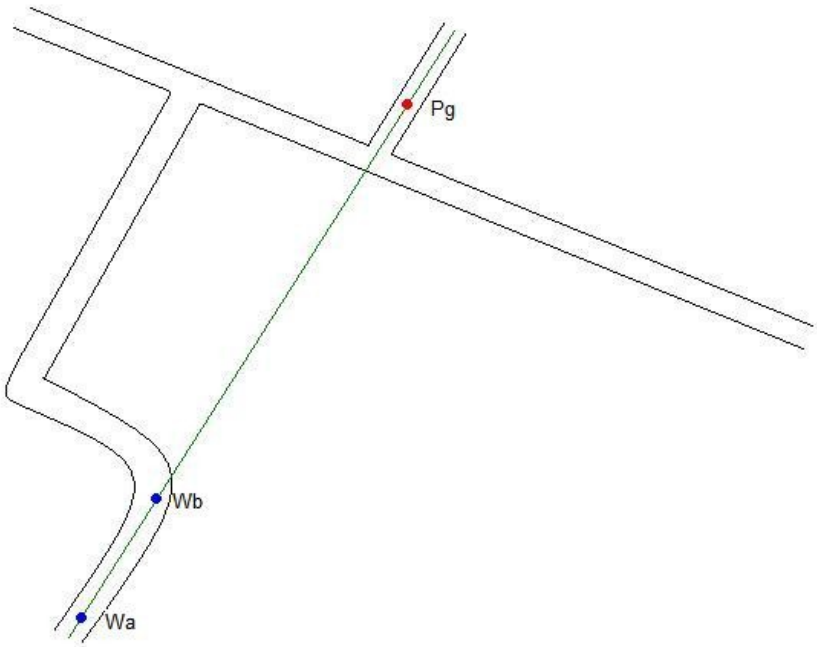


Figura 9 – Exemplo de um Possível Erro no Cálculo da Distância

O que vemos acima é que o ponto Pg (localização do automóvel) pertence à reta dos *waypoints* Wa e Wb, porém não está situada na mesma via dos mesmos. Assim sua distância à reta formada por esses pontos será zero, o que nos levaria a um erro. Por esse motivo, agimos da seguinte forma: consideramos que cada dois *waypoints* formam um segmento de reta. Esses mesmos *waypoints* são usados para formarmos um retângulo no qual um *waypoint* é o vértice superior esquerdo e o outro é o vértice inferior direito. Esse retângulo é aumentado por um offset predefinido, para que então seja a feita a verificação se a localização enviada pelo GPS está dentro ou não desse retângulo. Caso a localização fornecida pelo GPS não se encontre dentro do retângulo então os dois *waypoints* são descartados, passando para os seguintes. Caso esteja, o cálculo da distância do ponto à reta formada pelos *waypoints* é feito e se essa distância for menor do que um valor mínimo atual então o nome da rua, distância calculada e os dois *waypoints* são guardados em uma lista.

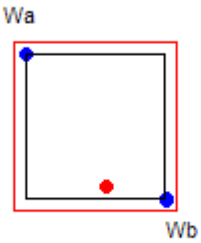


Figura 10 – Ilustração do retângulo formado pelos *waypoints*

Após verificar todo o mapa, a lista é ordenada por ordem de distância para que então seja definido em qual rua o veículo se encontra. Se não houver uma rua localizada anteriormente, a rua da localização será a que possuir a menor distância, e o nome da rua é guardado para a próxima localização. Caso já exista uma rua localizada previamente, verifica-se se a lista contém essa rua e, se isso acontecer, faz-se um cálculo de distância dessa rua para a que tem a menor distância na lista. Se esse cálculo retornar um valor menor do que a maior diferença, a rua localizada será a mesma que a rua localizada anteriormente. Caso a rua anterior não conste na lista, a rua escolhida será a de menor valor na lista.

Após a rua da localização do veículo ter sido escolhida ainda é feita uma decisão de qual *waypoint* o veículo está mais próximo. Isso deve ser feito já que uma mesma rua pode ter *waypoints* com velocidades máximas diferentes, então necessitamos saber em qual valor a velocidade deve ser limitada. Para isso basta calcularmos a distância da localização que o GPS forneceu para os dois *waypoints* do segmento de reta escolhido. O que tiver a menor distância é o *waypoint* a ser considerado. Parte do código que faz a localização do veículo está exemplificado a seguir:

```
#include <math.h>

#include "cavWayLocalizer.h"

using namespace std;

CWayLocalizer::CWayLocalizer():m_offset(5000),m_maxDistance(5000),
m_maxWayDifference(2000),m_maxFoundWay(3)
{
	m_foundWay.reserve(5);
	m_mapManager= NULL;
}

float CWayLocalizer::Localize(	const StGpsPosition &gpsPosition,
```

```
                                                StMapInfo &mapInfo,
                                                StWayInfo &wayInfo,
                                                StWayPoint &wayPoint)
{
      if(!m_mapManager)
      {
            m_wayMap= NULL;
            if(!(m_mapManager= CMapManager::GetInstance()))
                  return -1;
      }
      if(!m_wayMap)
      {
            m_lastWayOk= false;
            if(!m_mapManager->GetMap(gpsPosition, &m_wayMap))
                  return -1;
      }
      else if(!m_mapManager->CheckMap(gpsPosition, m_wayMap->GetMapInfo()))
      {
            m_lastWayOk= false;
            if(!m_mapManager->GetMap(gpsPosition, &m_wayMap))
                  return -1;
      }


      WayVector *wayVector= m_wayMap->GetWayVector();
      WayVector::iterator itWay;
      WayPointsVector::iterator itPoint;
      double a, b, c, distance;
      mapInfo= m_wayMap->GetMapInfo();

      m_foundWay.erase(m_foundWay.begin(), m_foundWay.end());

      for(itWay= wayVector->begin(); itWay != wayVector->end(); itWay++)
      {
            for( itPoint= itWay->points->begin();
                  (itPoint+1) != itWay->points->end();
                  itPoint++)
            {

                  ////////////////////////////////////////////////////////
///////
                  //verifica se a posição do GPS esta compreendida dentro
                  //do retângulo definido pelos dois waypoints
                  //se não estiver passa para o próximo waypoint

                  if( itPoint->latitude < (itPoint+1)->latitude )
                  {
                        if(   (gpsPosition.latitude+m_offset < itPoint-
>latitude) ||
                              (gpsPosition.latitude-m_offset > (itPoint+1)-
>latitude))
                                    continue;
                  }
                  else if((gpsPosition.latitude-m_offset > itPoint-
>latitude) ||
                              (gpsPosition.latitude+m_offset < (itPoint+1)-
>latitude))
                                    continue;

                  if( itPoint->longitude < (itPoint+1)->longitude )
                  {
```

```
                        if(   (gpsPosition.longitude+m_offset < itPoint-
>longitude) ||
                              (gpsPosition.longitude-m_offset > (itPoint+1)-
>longitude))
                                    continue;
                  }
                  else if((gpsPosition.longitude-m_offset > itPoint-
>longitude) ||
                              (gpsPosition.longitude+m_offset < (itPoint+1)-
>longitude))
                                    continue;

                  ////////////////////////////////////////////////////////
///////
                  //faz a distância do ponto do GPS até a reta que passa
pelos
                  //dois waypoints

                  a= (itPoint+1)->latitude - itPoint->latitude;
                  b= itPoint->longitude - (itPoint+1)->longitude;
                  c= ((double)(itPoint+1)->longitude * itPoint->latitude)-
                        ((double)itPoint->longitude *(itPoint+1)-
>latitude);


                  distance= fabs((double)((a*gpsPosition.longitude)+
(b*gpsPosition.latitude)+c)/
                                    (double)pow((a*a)+(b*b),0.5));


                  if(distance <= m_maxDistance)
                  {
                        StFoundWay newFoundWay;

                        newFoundWay.distance= distance;
                        newFoundWay.itWay= itWay;
                        newFoundWay.itPoint= itPoint;
                        newFoundWay.itNextPoint= itPoint+1;
                        m_foundWay.push_back(newFoundWay);
                  }
            }
      }

      if(!m_foundWay.empty())
      {
            sort(m_foundWay.begin(), m_foundWay.end());

            StFoundWay *newWay= &m_foundWay[0];

            ////////////////////////////////////////////////////
            //decide em qual rua o veículo está baseado na rua em que
            //o veículo estava na última localização
            if(m_lastWayOk)
            {
                  for(int i= 0; (i < m_maxFoundWay) && (i <
m_foundWay.size()); i++)
                  {
                        if(m_foundWay[i].itWay->info.wayId ==
m_lastWay.itWay->info.wayId)
                        {
```

```
                              if(abs(m_foundWay[i].distance-
m_lastWay.distance)<=m_maxWayDifference)
                                    newWay= &m_foundWay[i];
                        }
                  }
            }
            ////////////////////////////////////////////////////

            wayInfo= newWay->itWay->info;

            ////////////////////////////////////////////////////
            //calcula de qual wayPoint o veículo está mais próximo
            int difLatitude= newWay->itPoint->latitude-
gpsPosition.latitude;
            int difLongitude= newWay->itPoint->longitude-
gpsPosition.longitude;
            double dist1= pow((difLatitude*difLatitude)+
(difLongitude*difLongitude),0.5);

            difLatitude= newWay->itNextPoint->latitude-
gpsPosition.latitude;
            difLongitude= newWay->itNextPoint->longitude-
gpsPosition.longitude;
            double dist2= pow((difLatitude*difLatitude)+
(difLongitude*difLongitude),0.5);

            if(dist1 < dist2)
                  wayPoint= *(newWay->itPoint);
            else
                  wayPoint= *(newWay->itNextPoint);
            ////////////////////////////////////////////////////

            m_lastWayOk= true;
            m_lastWay= *newWay;
            return newWay->distance;
      }

      m_lastWayOk= false;
      return -1;
}

bool CWayLocalizer::StFoundWay::operator <(const StFoundWay &foundWay)
{
      return (distance < foundWay.distance);
}

bool CWayLocalizer::StFoundWay::operator >(const StFoundWay &foundWay)
{
      return (distance > foundWay.distance);
}
```

Após o veículo ser localizado no mapa como descrito acima, o Módulo Localizador envia para o Módulo Emulador GPRS o nome da rua em que o veículo se encontra e a velocidade máxima permitida.

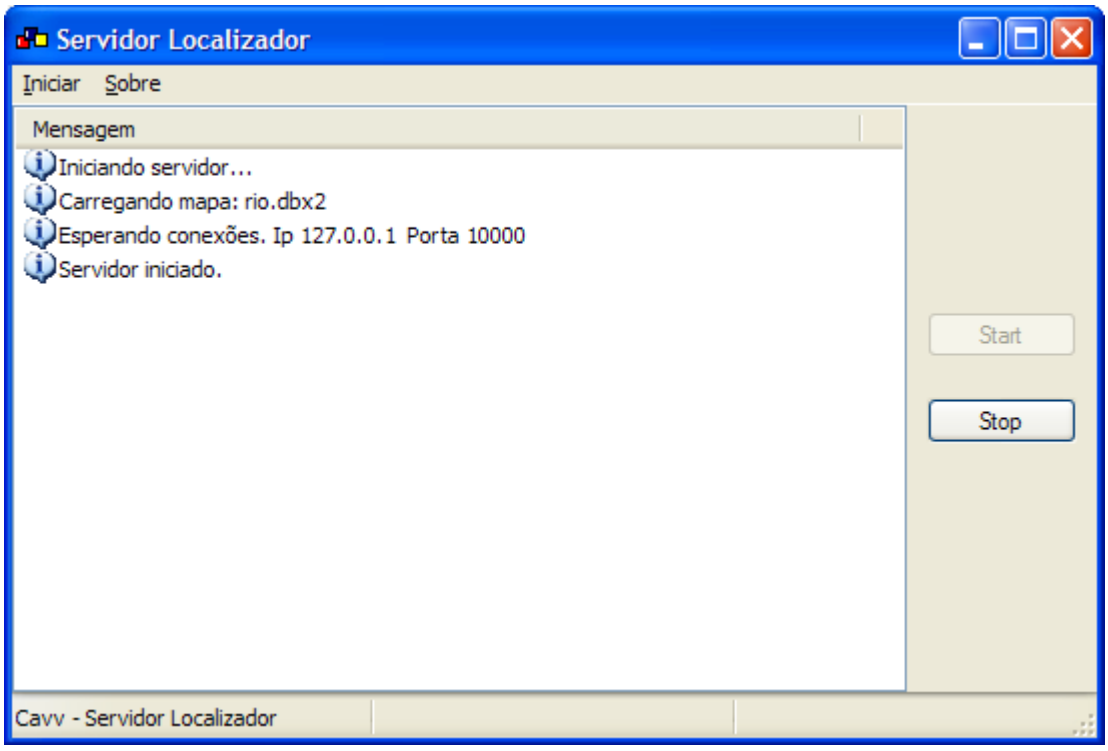
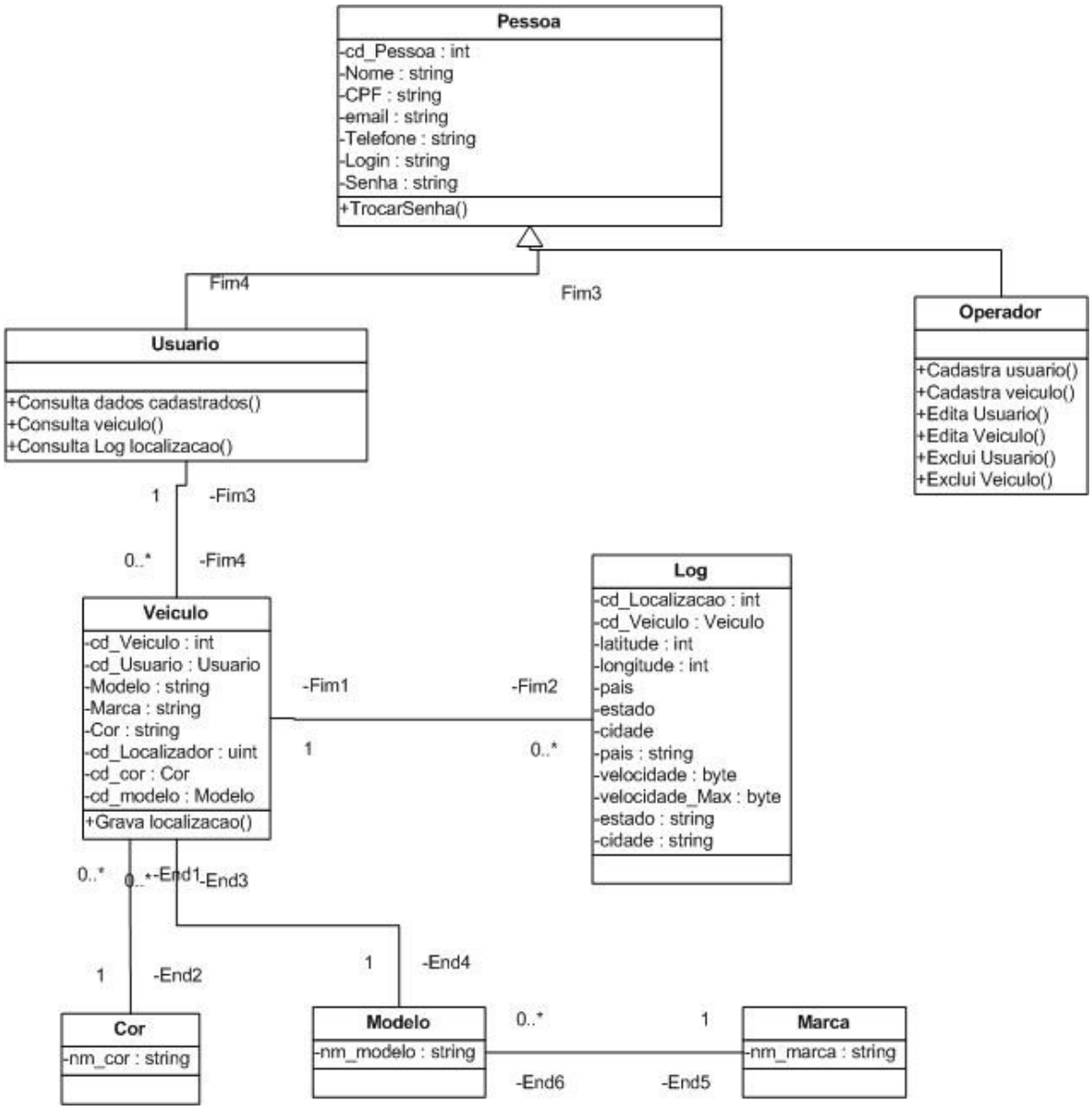


Figura 11 – Tela do Módulo Localizador

É do Módulo Localizador ainda a tarefa de gravar os *logs* dos veículos. Para isso ele utiliza o ID recebido na inicialização do modem e busca na tabela de veículos o veículo possuidor deste código. Em seguida o programa grava na tabela *log* os seguintes parâmetros: latitude e longitude enviadas pelo GPS, país, estado, cidade, nome da rua, velocidade do veículo, velocidade máxima permitida na via, além de data e hora. Para não sobrecarregar o banco há um parâmetro na configuração do servidor que define após quantos *logs* recebidos deve-se gravar no banco. O modelo lógico do banco de dados está representado a seguir:



4.3 Módulo Limitador

O módulo limitador tem a função de controlar a velocidade do automóvel. Como a velocidade pode mudar rapidamente, o micro-controlador envolvido nesse módulo recebe continuamente a velocidade do automóvel para limitá-la ou não, utilizando para tal a velocidade máxima da última via onde foi feita a localização do veículo.

O micro-controlador utilizado foi o AT89S52 da ATMEL que é um controlador CMOS de alta performance com 8Kbytes de memória Flash programável acoplada e 256 bytes de memória RAM. Esse dispositivo é fabricado usando a tecnologia de memória não-volátil de alta densidade da ATMEL e é compatível com a pinagem do padrão 80C51. A memória Flash no chip permite que a memória seja reprogramada dentro do próprio sistema ou por um programador convencional de memória não-volátil. Combinando uma CPU de 8 bits com uma memória Flash programável acoplada em um chip monolítico, a ATMEL AT89S52 é um poderoso micro-controlador que permite uma solução altamente flexível e com baixo custo. O código foi desenvolvido em C usando o compilador KeilC.

Além do micro-controlador, que é a peça principal do Módulo Limitador, foram utilizados para o circuito um chip MAX 232 da Texas, um visor de LCD HD44780U da Hitachi e um regulador de tensão de 5V KIA7805AP da KEC.

Como dito anteriormente, o intuito de usarmos um micro-controlador foi para permitir adicionar ao projeto dois sensores, um de luminosidade e um de chuva, que influenciam na limitação da velocidade. Há ainda a possibilidade de o usuário requisitar o nome da via em que está circulando ao pressionar um botão do micro-controlador. Ao fazê-lo o nome da rua aparece no visor de LCD caso haja uma via localizada. Caso contrário uma mensagem de erro é exibida. As velocidades do veículo e máxima da via são mostradas por *default* no *display* de LCD.

O esquema do circuito e seus componentes estão demonstrados abaixo:

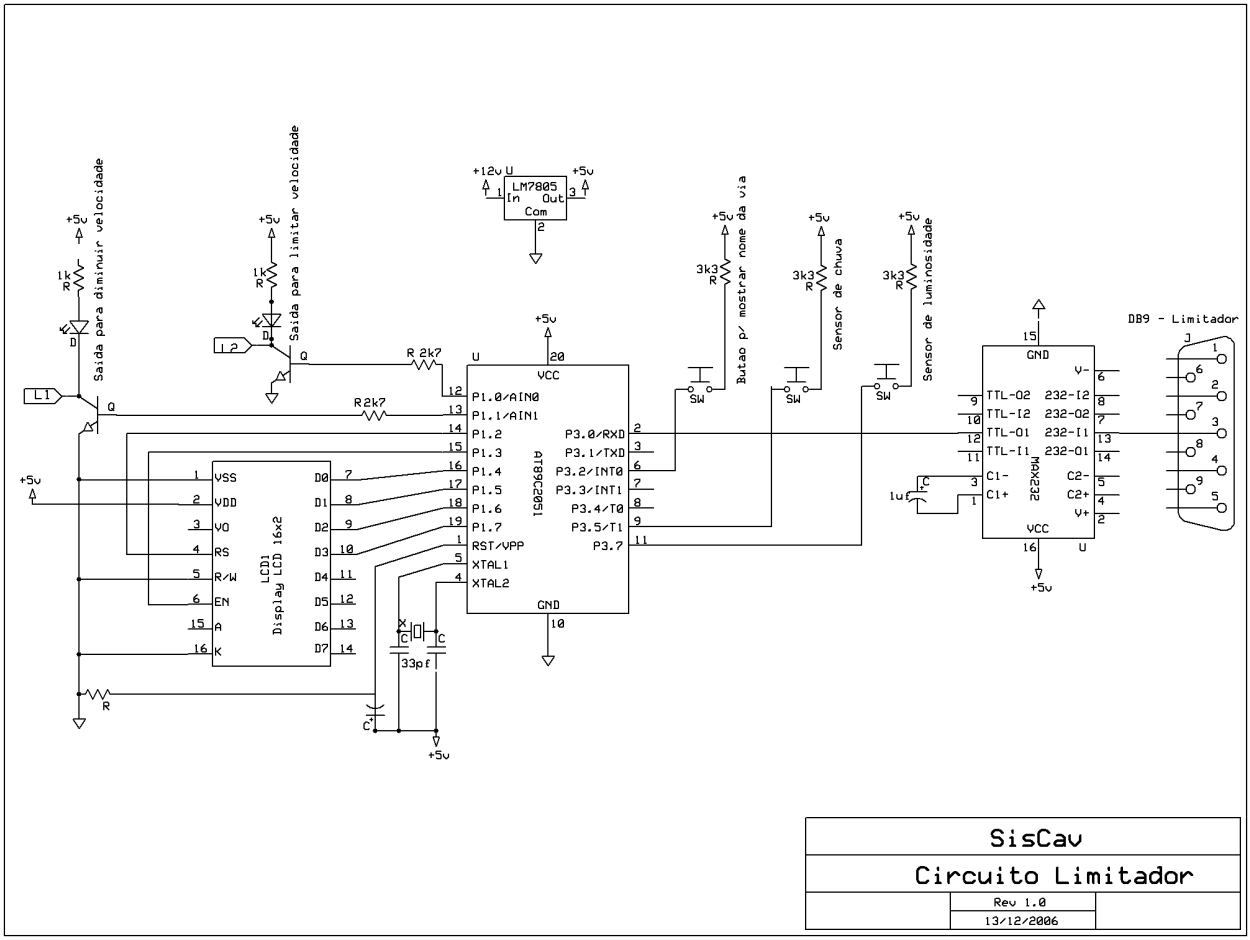


Figura 12 – Circuito do Módulo Limitador

Nele podemos observar as 4 entradas do micro-controlador que são usadas: os 2 sensores, o botão pelo qual o usuário requisita que o nome da rua seja mostrada no visor de LCD e a outra por onde entram os dados vindos do Módulo Localizador transmitidos pelo Módulo Emulador GPRS. Com esses dados cabe ao micro-controlador a decisão de limitar ou não a velocidade do veículo. Um trecho do código do micro-controlador é mostrado a seguir:

```
/
*************************************************************************
***/
/*
*/
```

```
/*   Speed Limiter
                          */
/*
*/
/
*************************************************************************
***/

#include <reg52.h>                        /* special function registers 8052
*/

#include  <stdio.h>                                /*  standard  I/O  .h-file
*/
#include  <ctype.h>                                 /*  character  functions
*/
#include <string.h>                           /* string and memory functions
*/

#include "cavSpeedCtrl.h"


extern interrupt_init ();              /* external function:  init serial
UART */
extern unsigned char messageBuffer[LIMITER_NEW_WAY_SIZE+1];


/*
Posição de cada variável no vetor messageBuffer
      messageBuffer[0]= msgType;
      messageBuffer[1]= maxSpeed;
      messageBuffer[2...]= wayName;
*/


/*indica qual a velocidade do veículo no momento*/
extern unsigned char carSpeed;

/*indica a velocidade máxima após verificar os sensores que podem diminuí-
la*/
unsigned char speedLimit;

/*pino que indica se está de noite. ATIVO em ZERO*/
sbit night=             P0^0;

/*pino que indica se está chovendo. ATIVO em ZERO*/
sbit rain=              P0^1;

/*pino que limita a velocidade. ATIVO em ZERO*/
sbit ctrlSpeed=   P2^0;

/*pino que diminui a velocidade. ATIVO em ZERO*/
sbit reduceSpeed=       P2^1;

/*comandos do LCD*/
sbit lcdEnable=         P3^7;
sbit lcdCmd=            P3^6;


extern bit speedError;              /*indica   se   houve   algum   erro   na
velocidade. Ocorre quando ocorre erro no GPS*/
```

```
extern bit locError;                /*indica se houve algum erro de
localização*/
extern bit receiveStart;            /*indica que começou a recepção de uma
mensagem*/
extern bit receiveComplete;         /*indica que uma mensagem foi recebida*/
extern bit showWayName;             /*indica que o botão de mostrar a
mensagem foi pressionado*/
extern bit showSpeed;               /*indica para mostrar a velocidade*/
extern bit waitShowSpeed;           /*indica se pode mostrar a velocidade,
isto é não esta sendo apresentado o nome da rua */



void SpeedCheck();
void PrintSpeed();
void PrintLCD(const char *buffer);
char *IntToStr(unsigned char speed);
void ClearLCD();
void JumpLCDLine();
void SleepLCD(unsigned char time);
void SendLCDCmd();


void main (void)
{
      memset(messageBuffer, 0, sizeof(messageBuffer));
      memcpy(&messageBuffer[2], INVALID_WAY, sizeof(INVALID_WAY));

      speedError= 1;
      locError= 1;

      /*Inicialização do LCD*/
      /*seleciona o número de linhas do cursor e o formato dos dados(8
bits)*/
      lcdCmd= 0;
      P1= 0x39;
    SendLCDCmd();

      /*seleciona para incrementar a posição do cursor a cada escrita*/
      P1= 0x06;
    SendLCDCmd();

      /*liga o display*/
      P1= 0x0C;
    SendLCDCmd();
      /*Fim inicialização do LCD*/

      /*configura e inicia as interrupções*/
      interrupt_init();

      ClearLCD();
      PrintLCD("Sistema Ativo.");

      while (1)
      {
            if(receiveComplete)
            {
                  receiveComplete= 0;
                  showSpeed= 1;
                  SpeedCheck();
            }
```

```
            if(showWayName)
            {
                  showWayName= 0;
                  waitShowSpeed= 1;
                  ClearLCD();

                  if(!locError)
                        PrintLCD(&messageBuffer[2]);
                  else
                        PrintLCD("Via não         Localizada.");

                  TR0= 1; //habilita o timer1
            }
            if(showSpeed && !waitShowSpeed)
            {
                  showSpeed= 0;
                  PrintSpeed();
            }
      }
}

void PrintLCD(const char *buffer)
{
      unsigned char index= 0;
      lcdCmd= 1;
      while(*buffer)
      {
            P1= *buffer;
            buffer++;
            SendLCDCmd();
            if(++index == 16)
                  JumpLCDLine();
      }
}

void PrintSpeed()
{
      ClearLCD();
      PrintLCD("Vel Max: ");
      if(!locError)
            PrintLCD(IntToStr(speedLimit));
      else
            PrintLCD("---");
      JumpLCDLine();
      PrintLCD("Vel Carro: ");
      if(!speedError)
            PrintLCD(IntToStr(carSpeed));
      else
            PrintLCD("---");
}

void SpeedCheck()
{
      speedLimit= messageBuffer[1];

      /* checa se houve algum erro*/
      if(speedError)
      {
            /* libera a velocidade já que houve um erro*/
            ctrlSpeed= 1;
            reduceSpeed= 1;
```

```
            return;
      }

      if(!(night | rain)) /*está chovendo e à noite*/
            speedLimit*= 0.75;
      else if(!(night & rain)) /*ou esta chovendo ou à noite*/
            speedLimit*= 0.85;

      if(carSpeed <= speedLimit) /*velocidade do carro <= a máxima*/
      {
            ctrlSpeed= 1;                 /*o carro esta livre para aumentar
sua velocidade*/
            reduceSpeed= 1;
            return;
      }
      if(carSpeed <= speedLimit*1.1) /*velocidade do carro <= a máxima com
tolerância*/
            reduceSpeed= 1;   /*indica   que   o   carro   pode   manter   sua
velocidade*/
      else
            reduceSpeed=  0;  /*indica  que  o  carro  deve  diminuir  sua
velocidade*/

      ctrlSpeed= 0;
}

char *IntToStr(unsigned char speed)
{
      static char strSpeed[4];
      strSpeed[0]= ((speed/100)%10)+'0';
      strSpeed[1]= ((speed/10)%10)+'0';
      strSpeed[2]= (speed%10)+'0';
      strSpeed[3]= 0;

      return strSpeed;
}

void ClearLCD()
{
      unsigned char i;//, j;

      lcdCmd= 0;
      P1= 0x01;
      lcdEnable= 1;

      //for(j= 0; j < 25; j++)
            for(i= 0; i < 255; i++)
                  SleepLCD(255);

      lcdEnable= 0;
      SleepLCD(255);

      /*posiciona o cursor na primeira posição da primeira linha*/
      P1= 0x02;
    SendLCDCmd();
      lcdCmd= 1;
}

/*posiciona o cursor na primeira posição da segunda linha*/
void JumpLCDLine()
{
```

```
      lcdCmd= 0;
      P1= 0xC0;
    SendLCDCmd();
      lcdCmd= 1;
}
void SleepLCD(unsigned char time)
{
      while(--time)
      {}
}

void SendLCDCmd()
{
      lcdEnable= 1;
      SleepLCD(100);
      lcdEnable= 0;
}
```

O código que faz a configuração inicial do micro-controlador, definindo o modo de comunicação serial por interrupção e as outras interrupções que serão usadas, está a seguir.

```
    /
    ************************************************************************
    *******/
    /*       serial_init: initialize serial interface
    */
    /
    ************************************************************************
    *******/
    void interrupt_init (void)
    {
      memset(serialBuffer, 0, sizeof(serialBuffer));
      receiveStart= 0;
      receiveComplete= 0;

      SCON  = 0x50;                    /* mode 1: 8-bit UART, enable
    receiver           */
      TMOD  = 0x21;                    /* timer 1 mode 2: 8-Bit reload,
                */
                                                  /*   timer 0 mode 1:
    16 bit count              */
      TH1   = BAUDRATE_4800;              /* baudrate para clock 11.0592
                    */
      TCON  = 0x45;                              /* timer 1 run
                                  */
      IE      = 0x13;                            /* habilita interrupção da
    serial, timer0 e*/
                                                  /*extern0, desabilita
    interrupção do timer1*/
      IP      = 0x10;                            /* coloca a maior
    prioridade para a interrupção da serial*/

      EA      = 1;                                     /* habilita as
    interrupções                  */
    }
```

4.4 Módulo Web

O Módulo Web é o módulo responsável pela visualização dos *logs* gerados pelo Módulo Localizador, além de possuir outras funcionalidades que visam à garantia de seu funcionamento.

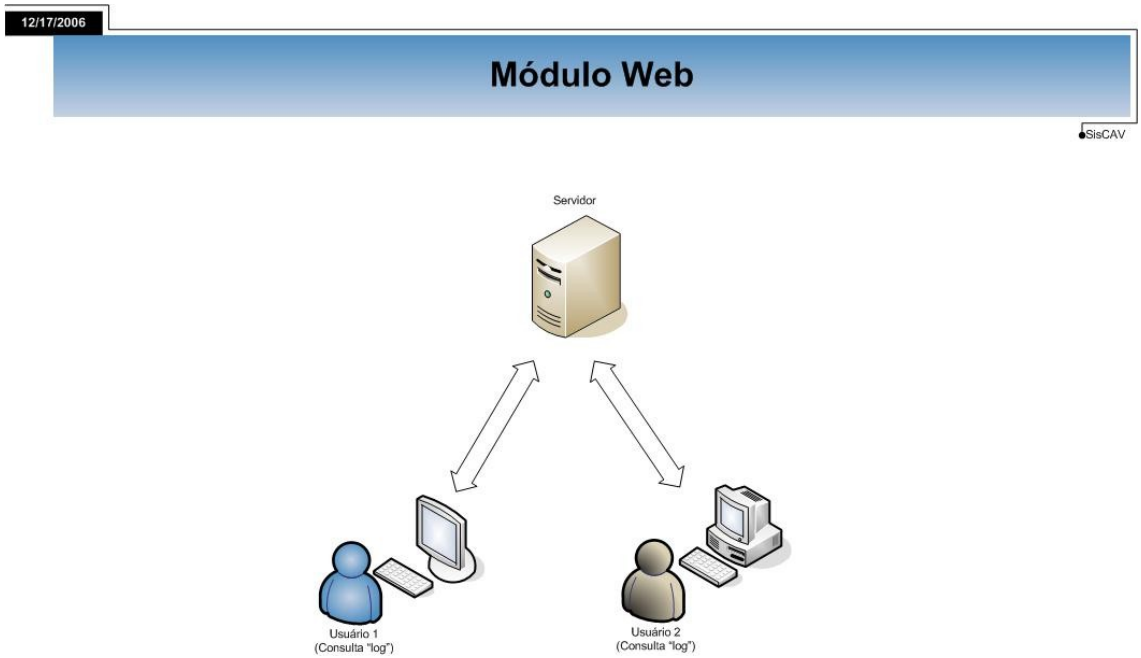


Figura 13 – Ilustração do Módulo Web

O Módulo Web utiliza as seguintes tecnologias associadas:

- Plataforma: Java. Utilização do modelo MVC como padrão de projeto
  - Modelo: Java Bean.
  - Vista: HTML e JSP's.
  - Controle: *Servlets*
- Banco de Dados: Postgre 8.1
- Servidor de Aplicação: Apache Tomcat 5.5.20

Após efetuar o *login* com sucesso, o usuário está apto a visualizar as funcionalidades do sistema. A implementação do *login* de acesso ao sistema foi desenvolvida verificando primeiramente a existência do usuário que possui o *username* informado e depois verificado se a senha que o usuário informou é igual à senha cadastrada. Dependendo da categoria do usuário as seguintes funcionalidades poderão ser acessadas:

- Da parte do Operador:
  - Cadastrar Usuário – O cadastro do usuário é a primeira etapa a ser cumprida. Aqui são cadastrados seus dados pessoais, para que em seguida seu(s) veículo(s) sejam também cadastrados.
  - Cadastrar Veículo – Um usuário pode ter vários veículos cadastrados. Ao entrar na tela visualização de *log* uma lista com todos os veículos do usuário será mostrada.
  - Editar Dados de um Usuário
  - Editar Dados de um Veículo

- Excluir Usuário
- Excluir Veículo
- Alterar Senha do Operador

- Da parte do Usuário:
  - Ver Dados Pessoais
  - Ver Dados de Seus Veículos
  - Alterar Senha – Ao ser cadastrado por um operador o usuário recebe uma senha padrão e precisa mudá-la em seu primeiro acesso.
  - Visualizar *Log* – Para visualizar o *log*, primeiro o usuário deve se logar e em seguida escolher o veículo desejado (se houver mais de um), além da data de início e fim do *log* que deseja visualizar.

5 Desenvolvimento

Serão abordados nesse capítulo tópicos importantes no desenvolvimento do projeto para que pudéssemos garantir sua manutenibilidade.

5.1 Mensagens Trocadas Entre os Módulos

Abaixo estão detalhadas as mensagens utilizadas na comunicação entre os módulos:

- Mensagens entre GPS e Módulo Emulador GPRS

  - Protocolo NMEA
    Sentido GPS -> Modem

- Mensagens entre o Módulo Emulador GPRS e Módulo Localizador

  - DeviceIdMsg - Mensagem de identificação do número do serial do limitador
    Sentido Modem -> Servidor
    Campos:
    code - 1 byte (código da mensagem - valor: 0x00)
    serialId - 4 bytes (serial ID do limitador)

  - GpsPositionMsg - Mensagem dos dados do GPS
    Sentido Modem -> Servidor
    Campos:
    code - 1 byte (código da mensagem - valor: 0x01)
    latitude - 4 bytes (latitude ja no formato decimal)
    longitude - 4 bytes (longitude já no formato decimal)
    carSpeed - 1 byte (velocidade em km/h do veículo)

  - GpsErrorMsg - Mensagem de erro nos dados do GPS
    Sentido Modem -> Servidor
    Campos:
    code - 1 byte (código da mensagem - valor: 0x02)

  - LocNewWayMsg - Mensagem de localização de uma nova rua
    Sentido Servidor -> Modem
    Campos:
    code - 1 byte (código da mensagem - valor: 0x03)
    maxSpeed - 1 byte (velocidade máxima em Km/h no trecho da rua localizada)
    wayName - 33 bytes (nome da rua)

  - LocNewWayMsg - Mensagem de localização da mesma rua anterior
    Sentido Servidor -> Modem

Campos:
code - 1 byte (código da mensagem - valor: 0x04)
maxSpeed - 1 byte (velocidade máxima em Km/h no trecho da rua localizada)

- LocErrorMsg - Mensagem de erro na localização
  Sentido Servidor -> Modem
  Campos:
  code - 1 byte (código da mensagem - valor: 0x05)

- Mensagens entre o Módulo Emulador GPRS e Módulo Limitador

  - LimiterNewWayMsg - Mensagem de localização de uma nova rua
    Sentido Modem -> Limitador
    Campos:
    code - 1 byte (código da mensagem - valor: 0xF0)
    maxSpeed - 1 byte (velocidade máxima em km/h no trecho da rua localizada)
    wayName - 33 bytes (nome da rua)

  - LimiterNewWayMsg - Mensagem de localização da mesma rua anterior
    Sentido Modem -> Limitador
    Campos:
    code - 1 byte (código da mensagem - valor: 0xF1)
    maxSpeed - 1 byte (velocidade máxima em km/h no trecho da rua localizada)

  - LimiterCarSpeedMsg - Mensagem de informação da velocidade do carro
    Sentido Modem -> Limitador
    Campos:
    code - 1 byte (código da mensagem - valor: 0xF2)
    carSpeed - 1 byte (velocidade em km/h do veículo)

  - LimiterLocErrorMsg - Mensagem de erro na localização
    Sentido Modem -> Limitador
    Campos:
    code - 1 byte (código da mensagem - valor: 0xF3)

  - LimiterGpsErrorMsg - Mensagem de erro nos dados do GPS
    Sentido Modem -> Limitador
    Campos:
    code - 1 byte (código da mensagem - valor: 0xF4)

A definição dessas mensagens está contida no arquivo cavProtocolMessages.h , da seguinte forma:

```
#ifndef CAV_PROTOCOL_MESSAGES_H
 #define CAV_PROTOCOL_MESSAGES_H

#include "wx/wx.h"

#define MAX_WAY_NAME 32
```

```
enum enProtocolMessages{eDeviceId= 0x00, eGpsPosition, eGpsError,
                                    eLocNewWay, eLocSameWay, eLocError,
eLocGpsError,
                                    eLimiterNewWay= 0xF0, eLimiterSameWay,
eLimiterCarSpeed,
                                    eLimiterLocError, eLimiterGpsError};

//todas essas estruturas têm alinhamento de memória de 1 byte
#ifdef __VISUALC__
      #pragma pack(push, 1)
#endif

//Modem->Servidor
struct StDeviceIdMsg
{
      wxByte code;
      unsigned serialId;
};

//Modem->Servidor
struct StGpsPositionMsg
{
      wxByte code;
      int latitude;
      int longitude;
      wxByte carSpeed;
};

//Modem->Servidor
struct StGpsErrorMsg
{
      wxByte code;
};

//Servidor->Modem
struct StLocNewWayMsg
{
      wxByte code;
      wxByte maxSpeed;
      char wayName[MAX_WAY_NAME+1];
};

//Servidor->Modem
struct StLocSameWayMsg
{
      wxByte code;
      wxByte maxSpeed;
};

//Servidor->Modem
struct StLocErrorMsg
{
      wxByte code;
};

//Servidor->Modem
struct StLocGpsErrorMsg
{
      wxByte code;
```

```
};

//Modem->Limitador
struct StLimiterNewWayMsg
{
      StLocNewWayMsg newWay;
};

//Modem->Limitador
struct StLimiterSameWayMsg
{
      StLocSameWayMsg sameWay;
};

//Modem->Limitador
struct StLimiterCarSpeedMsg
{
      wxByte code;
      wxByte carSpeed;
};

//Modem->Limitador
struct StLimiterLocErrorMsg
{
      wxByte code;
};

//Modem->Limitador
struct StLimiterGpsErrorMsg
{
      wxByte code;
};

#ifdef __VISUALC__
      #pragma pack(pop)
#endif

#endif
```

5.2 Classes Virtuais

As classes virtuais foram utilizadas com o intuito de modularizarmos ao máximo o código desenvolvido. Assim, caso houvesse qualquer modificação no código, a necessidade de retrabalho seria bastante diminuída. Por exemplo, criamos um classe base virtual responsável pela interface de leitura do mapa. Caso o formato do mapa utilizado venha a mudar, a única alteração que devemos fazer no código é implementar uma nova classe derivada desta classe virtual, reduzindo muito a necessidade de adaptação de código. A seguir estão exemplificadas algumas classes virtuais de maior pertinência.

- Define a interface de leitura do arquivo do mapa

```
#ifndef CAV_IWAY_MAP_H
 #define CAV_IWAY_MAP_H

#include "wx/wx.h"
#include "cavWayMapTypes.h"
```

```
class IWayMap
{
private:
  static unsigned m_mapId;

protected:
  wxString m_mapFilename;
  StMapInfo m_mapInfo;
  WayVector m_wayVector;

public:
  IWayMap(const char *mapFilename);

  virtual bool LoadMap()= 0;
  virtual const StMapInfo &GetMapInfo();
  virtual WayVector *GetWayVector();

};
```

#endif

- Define a interface de comunicação com o banco de dados

```
#ifndef CAV_IDB_COMM_H
 #define CAV_IDB_COMM_H

#include "wx/wx.h"
#include "cavWayMapTypes.h"

enum enDbClients{ eClientNotSpecified,
                  eODBCClient,
                  eOracleClient,
                  eSQLServerClient,
                  eInterBaseClient,
                  eSQLBaseClient,
                  eDB2Client,
                  eInformixClient,
                  eSybaseClient,
                  eMySQLClient,
                  ePostgreSQLClient};

class IDBComm
{

protected:
      wxString    m_dbError;
      wxString    m_dbIp;
      int               m_dbPort;
      enDbClients m_dbClient;
      int               m_saveFrequency;

public:
      IDBComm(const  char  *dbIp,  int  dbPort,  enDbClients  dbClient,  int
saveFrequency);

      const wxString &GetDbError();
```

```
      virtual bool Connect(const char *database, const char *user, const
char *password)= 0;
      virtual bool Disconnect()= 0;
      virtual bool SaveLog(unsigned deviceId,
                                    const StGpsPosition &gpsPosition,
                                    const StMapInfo &mapInfo,
                                    const StWayInfo &wayInfo,
                                    const StWayPoint &wayPoint)= 0;
};


#endif
```

- Define a interface para o algoritmo de localização

```
#ifndef CAV_IWAY_LOCALIZER_H
 #define CAV_IWAY_LOCALIZER_H

#include <vector>
#include "cavIWayMap.h"
#include "cavWayMapTypes.h"

class IWayLocalizer
{

protected:
      IWayMap *m_wayMap;

public:
      IWayLocalizer();

      virtual void SetWayMap(IWayMap *wayMap);
      virtual float Localize( const StGpsPosition &gpsPosition,
                                          StMapInfo &mapInfo,
                                          StWayInfo &wayInfo,
                                          StWayPoint &wayPoint)= 0;
};


#endif
```

- Define a interface para processamento da mensagem do GPS

```
#ifndef CAV_I_GPS_MSG_TRANSLATE_H
 #define CAV_I_GPS_MSG_TRANSLATE_H


#include "wx/wx.h"
#include "cavProtocolMessages.h"

enum enGpsTranlateCode{ eTransGpsPosition, eTransGpsSpeed,
                                    eTransGpsPosSpeed, eTransGpsError,
                                    eTransNotRelevantGpsMsg};

class IGpsMsgTranslate
{

public:
```

```
      virtual     enGpsTranlateCode     Execute(const     char*     gpsMsg,
StGpsPositionMsg &gpsPosition)= 0;
};

#endif
```

6 Resultados

Após realizarmos uma bateria de testes consideramos que o sistema está com uma boa precisão na localização do veículo. Em todos os testes o veículo foi localizado com sucesso, salvo nos casos em que o receptor GPS perdia o sinal. Nesses casos mensagens de erro eram inseridas tanto no *log* do veículo quanto no *display* LCD.

Nos primeiros testes o sistema encontrava dificuldade em alguns cruzamentos, o que já era esperado. Para diminuir esse problema privilegiamos a localização da rua anterior em que o veículo se encontrava. Dessa forma reduzimos bastante os equívocos do sistema que teve um índice de acerto da ordem de 95% fora dos cruzamentos e de 80% nos mesmos.

7 Conclusão

Com este projeto esperamos ilustrar uma alternativa adicional aos radares de fiscalização eletrônica para diminuir os acidentes de trânsito, que têm no excesso de velocidade sua principal causa. Para isso, utilizamos métodos e tecnologias estudadas durante o curso e que tem vasta utilização na atualidade.

Os autores pretendem fazer melhorias no projeto como uma divisão do mapa em vários quadrados delimitados por latitudes e longitudes mínimas e máximas para que assim o programa de localização não precise varrer o mapa todo em busca da localização do veículo. Com essa divisão o algoritmo irá refinando a busca em níveis cada vez menores até chegar ao *waypoint* mais perto da localização e de forma mais rápida e otimizada.

Lembramos que não era pretensão dos autores do projeto discutir ética ou legalmente sua aplicabilidade, já que eles estão plenamente cientes que o assunto deveria ser discutido exaustivamente antes que a solução chegasse ao público.

Espera-se, porém, que a solução proposta ao menos faça os leitores pensarem a cerca do excesso velocidade e outras imprudências na direção que cada ano vêm matando mais pessoas em nosso país.

Bibliografia

[1] Prefeitura de Fortaleza, http://www.amc.fortaleza.ce.gov.br/modules/wfchannel/index.php?pagenum=18 , acessado em 22 de novembro de 2006

[2] CESVI Brasil, http://www.cesvibrasil.com.br/revista/ed_49_seg_viaria.asp , acessado em 22 de novembro de 2006

[3] DENATRAN, http://www.denatran.gov.br/Legislacao.htm , acessado em 22 de novembro de 2006

[4] DER-RJ, http://www.der.rj.gov.br/educacao_transito.asp, acessado em 22 de novembro de 2006

[5] Wikipedia, http://pt.wikipedia.org/wiki/Sistema_de_Posicionamento_Global , acessado em 25 de novembro de 2006

[6] Wikipedia, http://pt.wikipedia.org/wiki/WxWidgets , acessado em 25 de novembro de 2006

[7] SQLAPI++, http://www.sqlapi.com/ , acessado em 25 de novembro de 2006

[8] Bate Byte, http://www.pr.gov.br/batebyte/edicoes/2000/bb99/estagiario.htm , acessado em 25 de novembro de 2006

[9] GPS Security, http://www.mapas-rs.com.br/gps/gps-security.htm , acessado em 27 de novembro de 2006

[10] Matemática Essencial http://pessoal.sercomtel.com.br/matematica/geometria/ganalitica/ganalitica.htm , acessado em 01 de dezembro de 2006

[11] Informações sobre Protocolo NMEA 2.0, http://www.gpsinformation.org/dale/nmea.htm, acessado em 25 de novembro de 2006

[12] Manual do micro-controlador AT89S52 da ATMEL , www.datasheetcatalog.com/datasheets_pdf/A/T/8/9/AT89S52.shtml , acessado em 30 de novembro de 2006

[13] Manual do compilador Keil C , http://www.keil.com/c51/ , acessado em 30 de novembro de 2006

[14] Postgre 8.1 , http://www.postgresql.org/ , acessado em 30 de novembro de 2006

[15] Apache Tomcat, http://tomcat.apache.org/ , acessado em 30 de novembro de 2006

Apêndice A – Especificação de Requisitos de Software

1 Introdução

1.1 Finalidade

O objetivo deste documento é descrever as Especificações de Requisito de Software a fim de facilitar o entendimento do sistema pela equipe de desenvolvimento e por seus usuários, bem como para os próprios avaliadores. Neste documento será especificado o detalhamento da solução.

1.2 Escopo

O projeto desenvolvido é o Sistema de Controle Automotivo de Velocidade (SisCAV) cujo objetivo é o de controlar a velocidade de um veículo restringindo-a à velocidade máxima permitida na via segundo o Código Brasileiro de Trânsito. O SisCAV engloba 4 módulos distintos que são: Módulo Emulador GPRS, Módulo Localizador, Módulo Limitador e o Módulo Web.

O projeto foi desenvolvido para que tivesse uma interface fácil de usar, além de alta manutenibilidade.

1.3 Definições, Acronismos e Abreviaturas

- ERS – Especificação de Requisitos de Software
- DD – Dicionário de Dados
- DER – Diagrama de Entidade Relacionamento
- DS – Diagramas de Seqüências

1.4 Referências

Java Development Kit (JDK) 1.5, Apache Tomcat 5.5.20, wxWidgets 2.6.3, SQLAPI 3.7.18

1.5 Resumo

No restante deste documento, serão descritas com detalhes as principais funções do sistema SisCAV, definindo os seus diferentes requisitos, funções e restrições.

Para isso, utilizaremos uma modelagem conceitual orientada a objeto ao invés da modelagem estruturada.

Posteriormente definiremos todas os eventos e os diagramas de casos de uso. Em seguida serão realizados os diagramas de classes, o diagrama de seqüências e, por fim, a modelagem do banco de dados.

2 Descrição Geral

2.1 Perspectiva do Produto

O sistema SisCAV é dependente apenas do sistema de GPS e suas funcionalidades e operação utilizam obrigatoriamente recursos próprios. Esses recursos são:

- Hospedagem do servidor Web - necessita de uma máquina com um servidor Tomcat Apache 5.5.20 instalado. O servidor Tomcat é um servidor Java que traduz as instruções do código Java compilado para os *browsers* de internet interpretarem corretamente.
- Hospedagem do Sistema de Gerenciamento de Banco de Dados (SGBD) Postgre 8.1 – utiliza a mesma máquina do servidor Web, no qual vai ser instalado o Postgre.
- Java Development Kit (JDK) 1.5 – Kit de Desenvolvimento Java que contém a máquina virtual Java que interpretará os códigos compilados. Esse kit é usado pelo servidor Java que traduzirá os códigos compilados para execução do sistema.
- Receptor GPS Garmin eTreX Legend 12 Canais

Os módulos Localizador e Emulador GPRS foram desenvolvidos utilizando a linguagem de programação C++ e o compilador Visual C++ 6.0 , *service pack* 6 com as bibliotecas adicionais WxWidgets 2.6.3 e SQLAPI 3.7.18 .

O Módulo Web foi desenvolvido em Java JSDK1.5, utilizando IDE Eclipse 3.2. Os conceitos de Servlets, JSP e Java Beans foram também utilizados.

No Módulo Limitador utilizamos o micro-controlador AT89S52 da ATMEL que é um controlador CMOS de alta performance com 8Kbytes de memória Flash programável acoplada e 256 bytes de memória RAM. Esse dispositivo é fabricado usando a tecnologia de memória não-volátil de alta densidade da ATMEL e é compatível com a pinagem do padrão 80C51. A memória Flash no chip permite que a memória seja reprogramada dentro do próprio sistema ou por um programador convencional de memória não-volátil. Combinado uma CPU de 8 bits com uma memória Flash programável acoplada em um chip monolítico, a ATMEL AT89S52 é um poderoso micro-controlador que permite uma solução altamente flexível e com baixo custo. O código foi desenvolvido em C usando o compilador KeilC. Além do micro-controlador que é a peça principal do Módulo Limitador foram utilizados para o circuito um chip MAX 232 da Texas, um visor de LCD HD44780U da Hitachi, um regulador de tensão de 5V KIA7805AP da KEC.

2.2 Características do Usuário

O usuário que irá usufruir do sistema deve adquirir o conjunto que seria composto pelo receptor GPS, pelo modem GPRS e pelo Módulo Limitador e que será instalado em seu veículo. A seguir o usuário deve ser cadastrado por um operador do sistema assim como seu(s) veículo(s) para que possa acessar as funcionalidades do Módulo Web. Dessa forma o usuário precisa apenas estar familiarizado com a navegação pela Internet não necessitando de conhecimentos técnicos.

2.3 Restrições

O sistema SisCAV é multi-plataforma podendo funcionar tanto no Windows quanto no Linux, já que foram utilizadas bibliotecas e frameworks que são notoriamente portáveis.

A máquina servidora que contém o servidor Tomcat instalado deverá possuir uma razoável capacidade de disco e de memória RAM, pois nela serão instalados o servidor Tomcat, a JSDK 1.5 e o banco Postgree, além de que durante a execução do sistema em produção após um longo período de funcionamento a massa de dados deverá possuir um tamanho razoável.

2.4 Pressupostos e Dependências

O correto funcionamento do sistema e a precisão do mesmo estão altamente relacionados aos dados contidos no mapa utilizado para a localização do veículo. Os autores pressupõem que os dados contidos no mesmo sejam fidedignos, o que também foi assumido em relação à tecnologia GPS, por se tratar de uma tecnologia altamente empregada e notoriamente confiável e precisa.

O projeto depende ainda da velocidade em que se dará a comunicação do modem GPRS com o servidor, o que não constitui uma preocupação de fato já que o fluxo de dados é baixo e a atual evolução da tecnologia celular permite que essa comunicação seja feita de maneira fácil e segura. Voltamos a lembrar que, por se tratar de um protótipo, o modem GPRS será emulado por um software a ser implementado pela equipe, já que o custo do mesmo seria muito alto. Assim, a arquitetura do protótipo é assim representada:

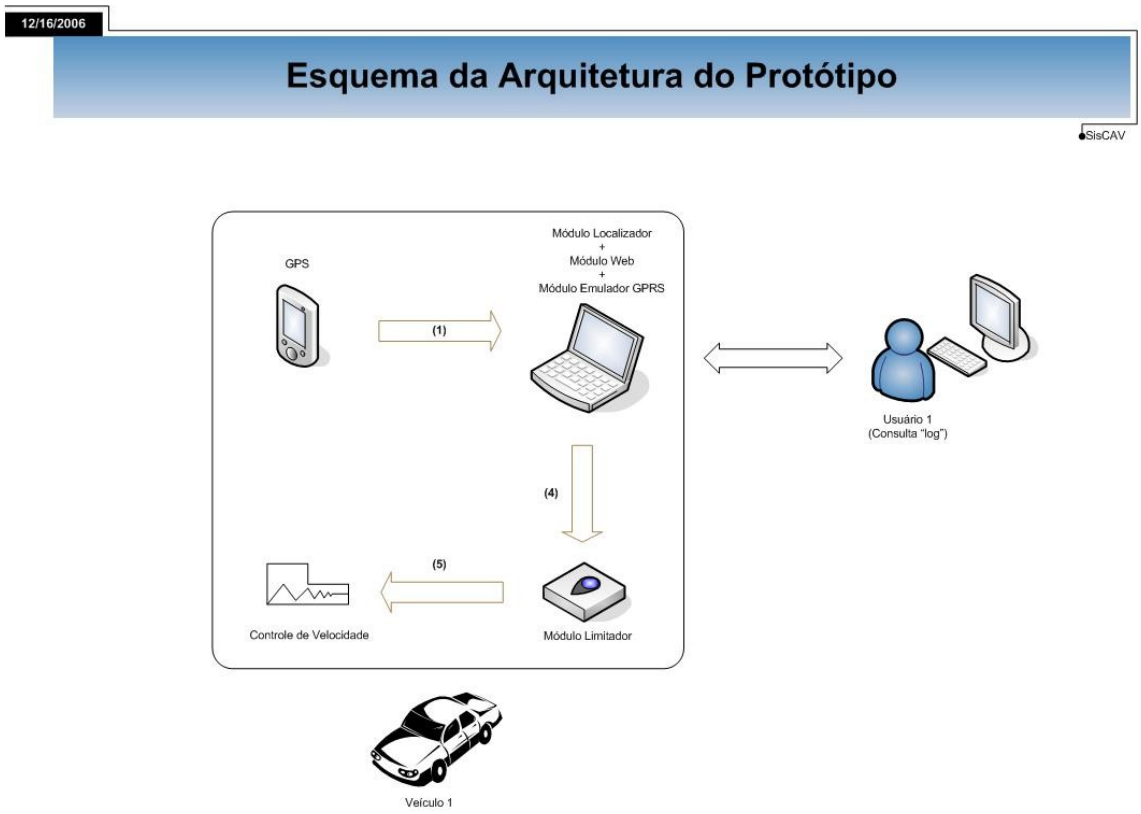


Figura 14 - Arquitetura do Protótipo

3 REQUISITOS ESPECÍFICOS

3.1 Requisitos Funcionais

Para descrever os requisitos funcionais é necessário primeiramente listar os eventos mais básicos do sistema.

3.1.1 Eventos

1.1 Módulo Emulador GPRS

- Recebe dados do GPS e envia para o servidor, junto com o ID do equipamento.
- Envia velocidade para Módulo Limitador a cada vez que ela é enviada pelo GPS
- Recebe dados do servidor
- Envia dados para o Módulo Limitador

1.2 Módulo Localizador

- Recebe dados do Módulo Emulador GPRS
- Localiza no mapa as coordenadas
- Envia para o Módulo Emulador GPRS o nome da rua e sua velocidade máxima permitida ou mensagem de erro
- Salva no banco de dados as informações pertinentes

1.3 Módulo Limitador

- Recebe dados do Módulo Emulador GPRS
- Calcula se deve limitar a velocidade do veículo levando em consideração os sensores de chuva e luminosidade
- Envia ordem para limitar ou não a velocidade
- Caso seja requerido o nome da via, envia para o LCD a última rua localizada

1.4 Módulo Web

- Solicita *Login*
- Se usuário
    - Exibe menu com opções
    - Caso, selecione visualizar, exibe informações do usuário e lista de veículos.
    - Com o veículo desejado selecionado, solicita a data de início e fim do *log* que se deseja visualizar.
    - Exibe *log*
- Se operador
    - Exibe menu para incluir, editar e excluir usuários e seus veículos

3.1.2 Regras de Negócio

Regras para Operadores

- Um operador pode cadastrar um usuário.
- Um operador pode cadastrar um veículo para um usuário específico.
- Um operador pode alterar dados cadastrais de um usuário
- Um operador pode alterar dados de um veículo associado a um usuário
- Um operador pode excluir um usuário
- Um operador pode excluir um veículo associado a um usuário
- Um operador pode alterar sua senha.

Regras para Usuários

- Um usuário pode ver seus dados cadastrais.
- Um usuário pode ver dados de seus veículos.
- Um usuário pode mudar sua senha.
- Um usuário deve mudar sua senha padrão inicialmente cadastrada.
- Um usuário pode visualizar o *log* de seu veículo

Regras do Sistema

- Um usuário pode ter zero, um ou muitos carros cadastrados.
- Um veículo pode ter zero, uma ou muitas localizações.
- O veículo deve possuir um equipamento limitador.

3.1.3 Diagrama de Entidades e Relacionamentos

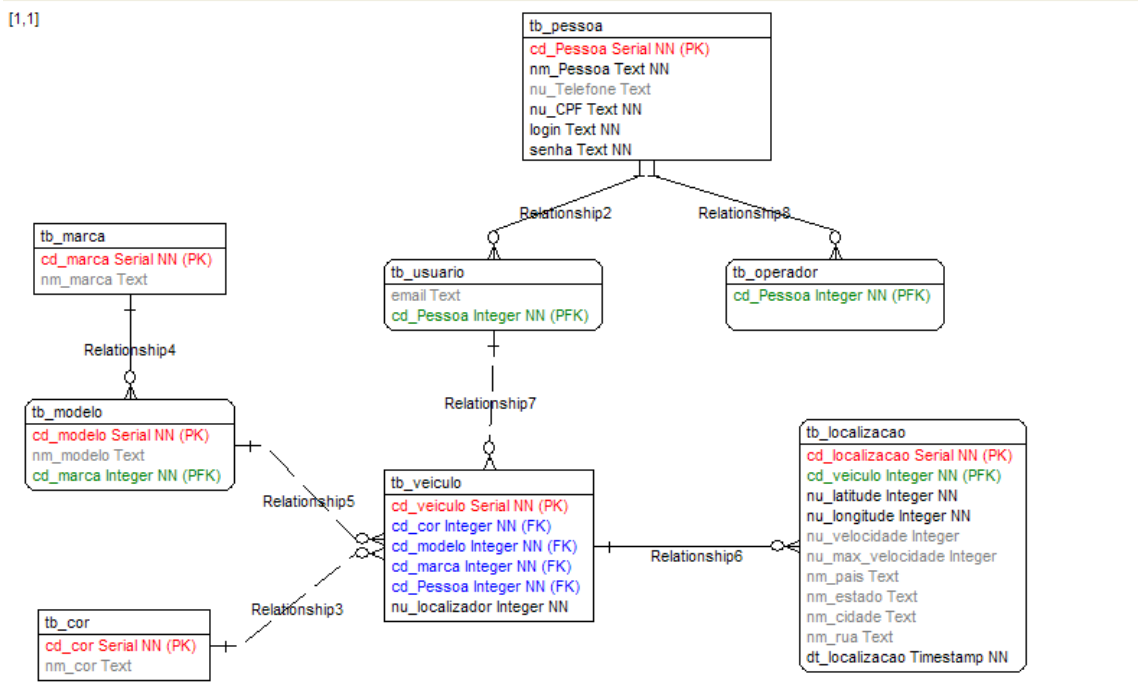


Figura 15– Diagrama de Entidade e Relacionamentos do SisCAV

Para o nosso DER temos o seguinte dicionário de dados:

- **tb_pessoa –** Entidade que define os dados relativos a uma pessoa
  - **cd_Pessoa –** código que identifica unicamente uma pessoa
  - **nm_Pessoa –** nome da pessoa
  - **nu_Telefone –** número de telefone de contato da pessoa
  - **nu_CPF –** número do CPF da pessoa
  - ***login* –** *login* de acesso ao sistema
  - **senha –** senha de acesso ao sistema

- **tb_usuario –** Diferenciação da tabela pessoa que define um usuário
  - **email –** email do usuário

- **tb_operador –** Diferenciação da tabela pessoa que define um operador

- **tb_veiculo –** Entidade que define os dados de um veículo
  - **cd_veiculo –** código que identifica unicamente um veículo
  - **cd_cor –** código que identifica unicamente uma cor
  - **cd_modelo –** código que identifica unicamente um modelo
  - **cd_marca –** código que identifica unicamente uma marca
  - **cd_pessoa –** código que identifica unicamente uma pessoa
  - **nu_localizador –** número do localizador instalado no veículo

- **tb_localizacao –** Entidade que define a localização de um veículo
  - **cd_localizacao –** código que identifica unicamente uma localização
  - **cd_veiculo –** código que identifica unicamente um veículo
  - **nu_latitude –** latitude da localização
  - **nu_longitude –** longitude da localização
  - **nu_velocidade –** velocidade do veículo
  - **nu_max_velocidade –** velocidade máxima permitida na localização
  - **nm_pais –** país da localização
  - **nm_estado –** estado da localização
  - **nm_cidade –** cidade da localização
  - **nm_rua –** rua da localização
  - **dt_localizacao –** data e hora da localização

- **tb_marca –** Entidade que define a marca de um veículo
  - **cd_marca –** código que identifica unicamente uma marca
  - **nm_marca –** nome da marca do veículo

- **tb_modelo –** Entidade que define o modelo de um veículo
  - **cd_modelo –** código que identifica unicamente o modelo de um veículo
  - **nm_modelo –** nome do modelo do veículo
  - **cd_marca –** código que identifica unicamente uma marca

- **tb_cor –** Entidade que define a cor de um veículo
  - **cd_cor –** código que identifica unicamente uma cor

- **nm_cor –** nome da cor

3.1.4 Casos de Uso

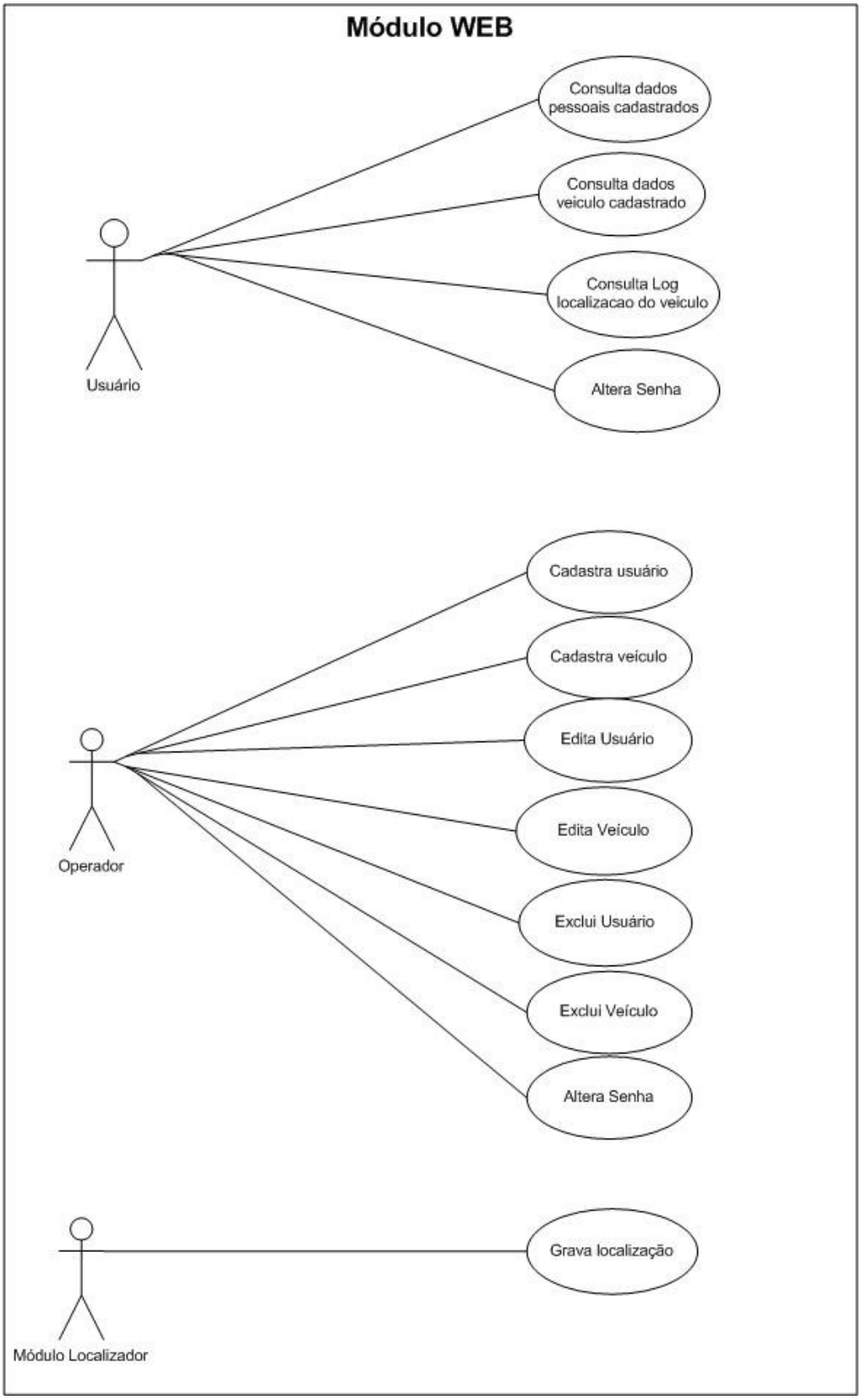


Figura 16 – Casos de Uso Módulo Web

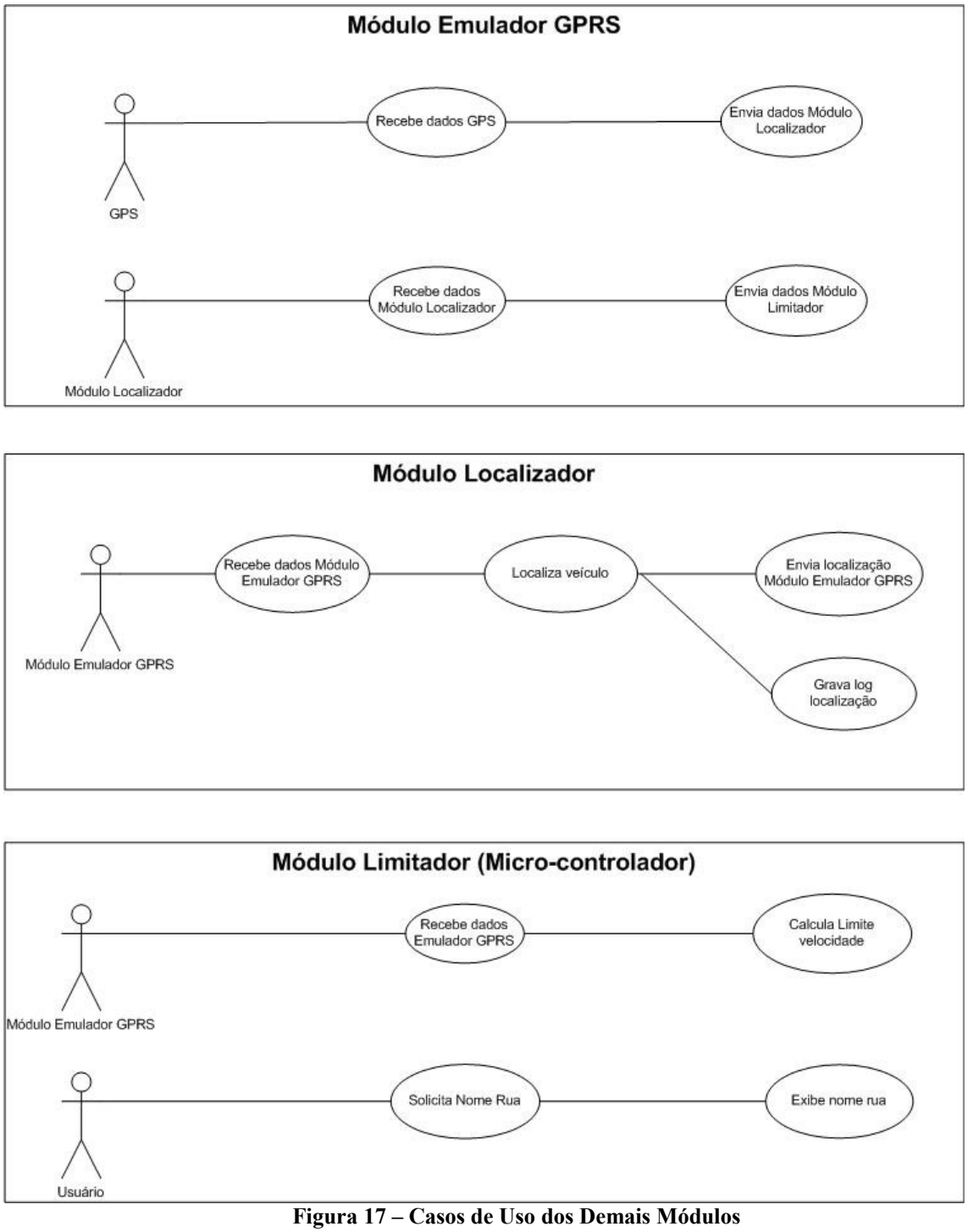


Figura 17 – Casos de Uso dos Demais Módulos

3.1.5 Diagrama de Classes

O diagrama de classes representa toda estrutura dos objetos básicos que compõem o software. No diagrama de classes podem estar descritos, tanto relacionamentos entre objetos quanto entre classes, quando representamos classes abstratas.

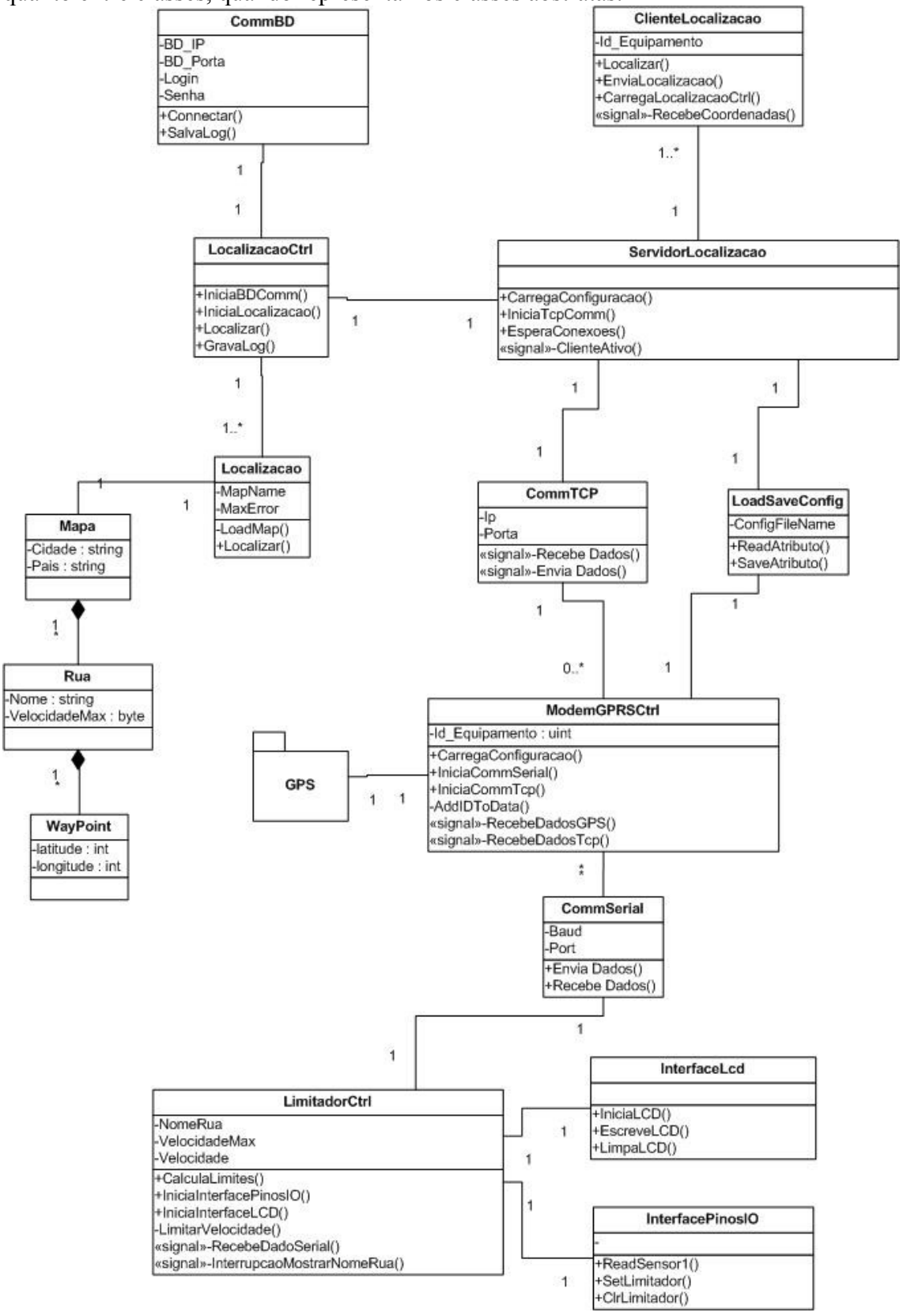


Figura 18 – Diagrama de Classes SisCAV

O diagrama de classes nos permite ter uma visão mais ampla sobre a estrutura de todo o sistema, mas o exato funcionamento deste sistema é difícil de ser derivado a partir dele. Por isso precisamos de um outro diagrama, o de seqüência.

3.1.6 Diagramas de Seqüência

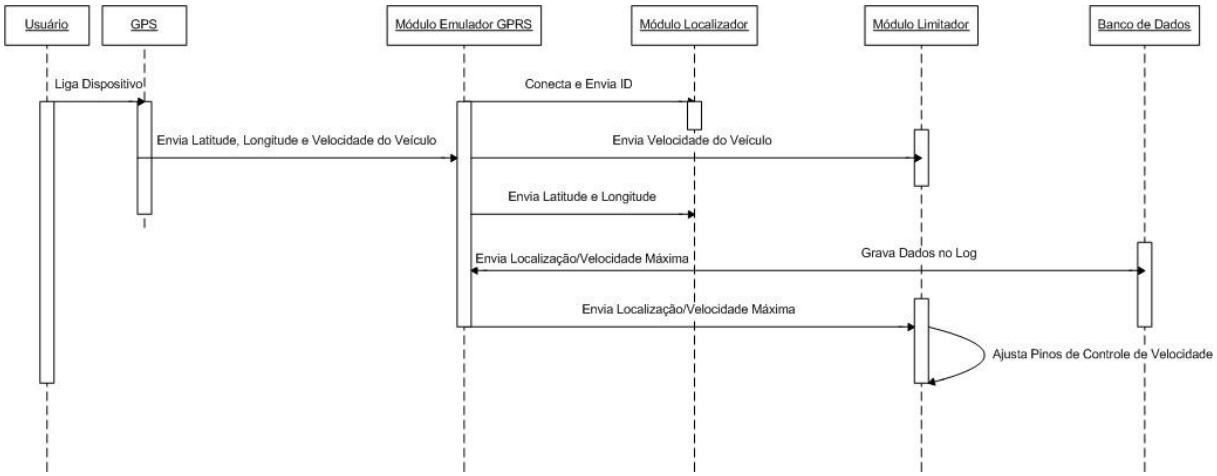


Figura 19 – Diagrama de Seqüência do Sistema

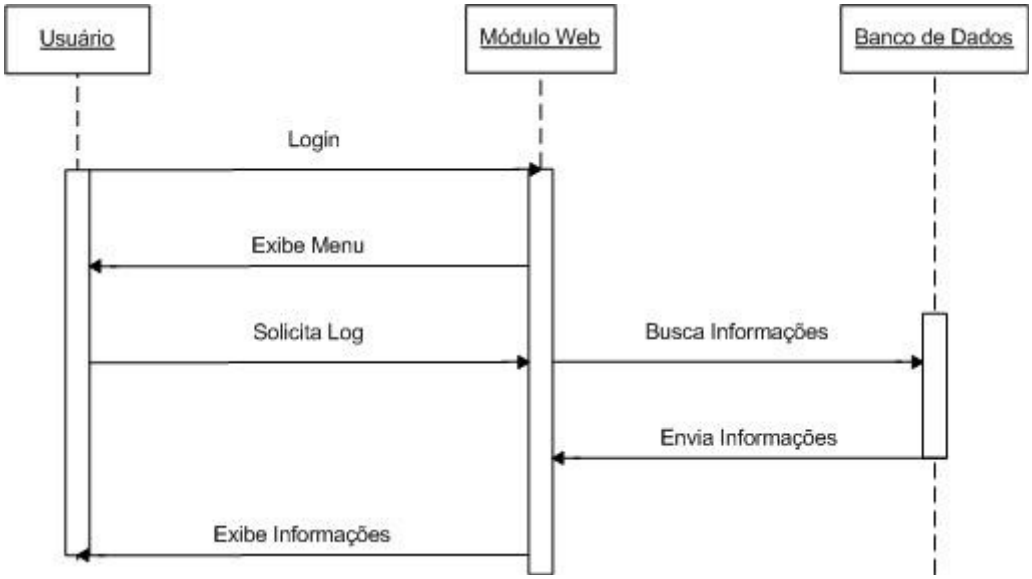


Figura 20 – Diagrama de Seqüência do Módulo Web

3.2 Interfaces Externas

3.2.1 Interfaces dos Usuários

O Sistema de Controle Automotivo de Velocidade possui uma interface bastante simples e amigável ao usuário que pode acessá-la através da Internet. O principal objetivo do Módulo Web é disponibilizar os *logs* dos veículos que possuem o sistema e portanto além

dessa funcionalidade foram implementada apenas as funcionalidades mais comuns de inclusão, edição e exclusão de cadastros. A seguir essas principais funções serão mostradas.

- ***Login***

Para garantir a integridade dos dados do sistema foi desenvolvido o módulo de *login* que garante que somente pessoas cadastradas na base de dados possam acessar o sistema.

Para se logar ao sistema, usuário deve fornecer o seu *username* cadastrado e a sua senha. Após submeter o formulário de *login*, o sistema fará as devidas verificações para confirmar a correta identificação do usuário.

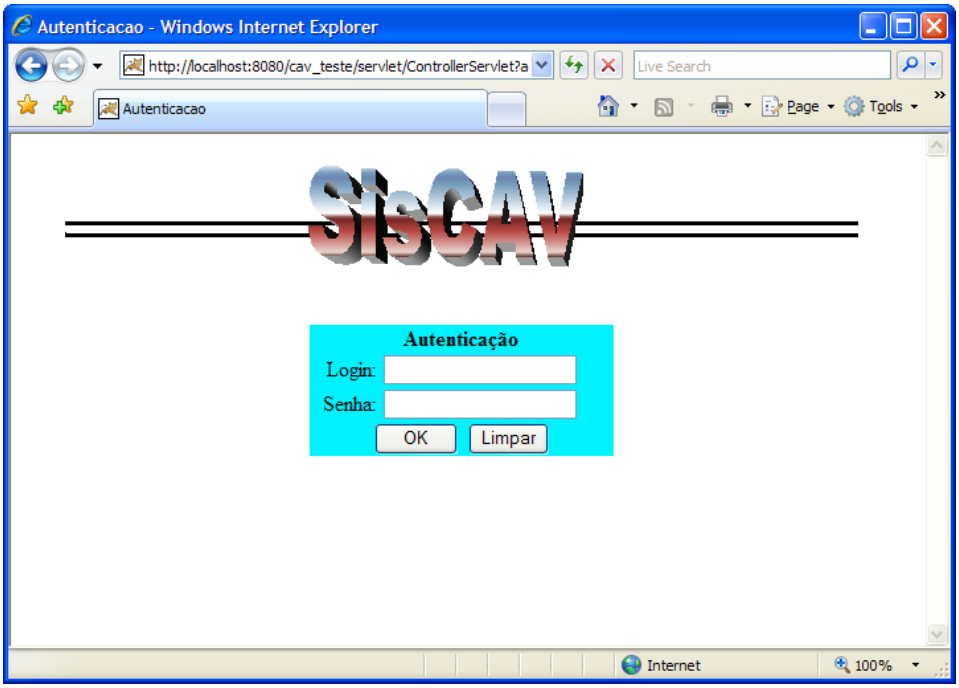


Figura 21 - Tela de *Login* do Sistema

- **Alterar Senha**

Tanto o usuário como o operador têm a opção de mudar sua senha.

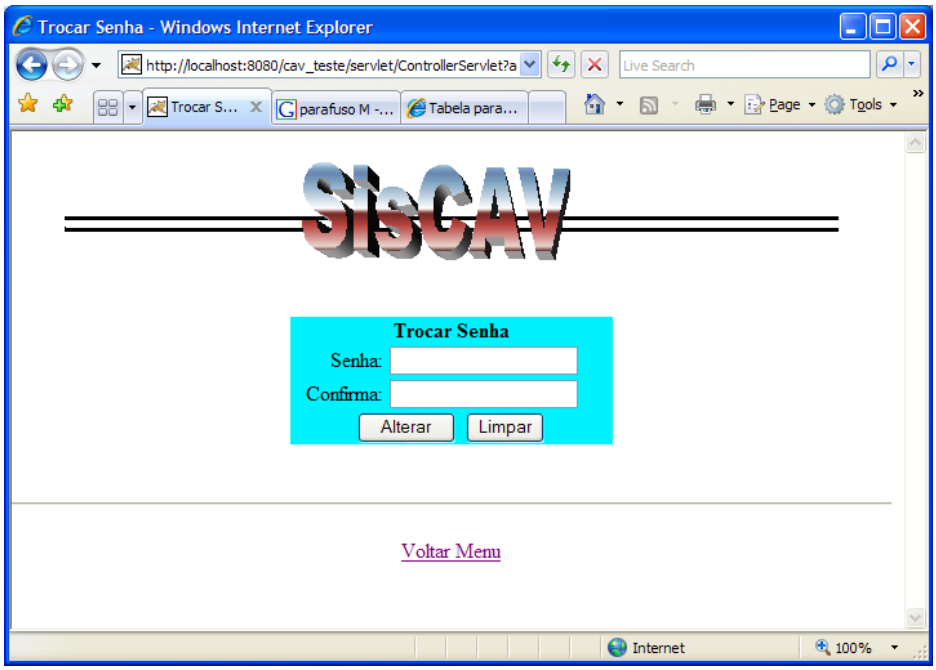


Figura 22 - Alteração de Senha

3.2.1.1 Operador

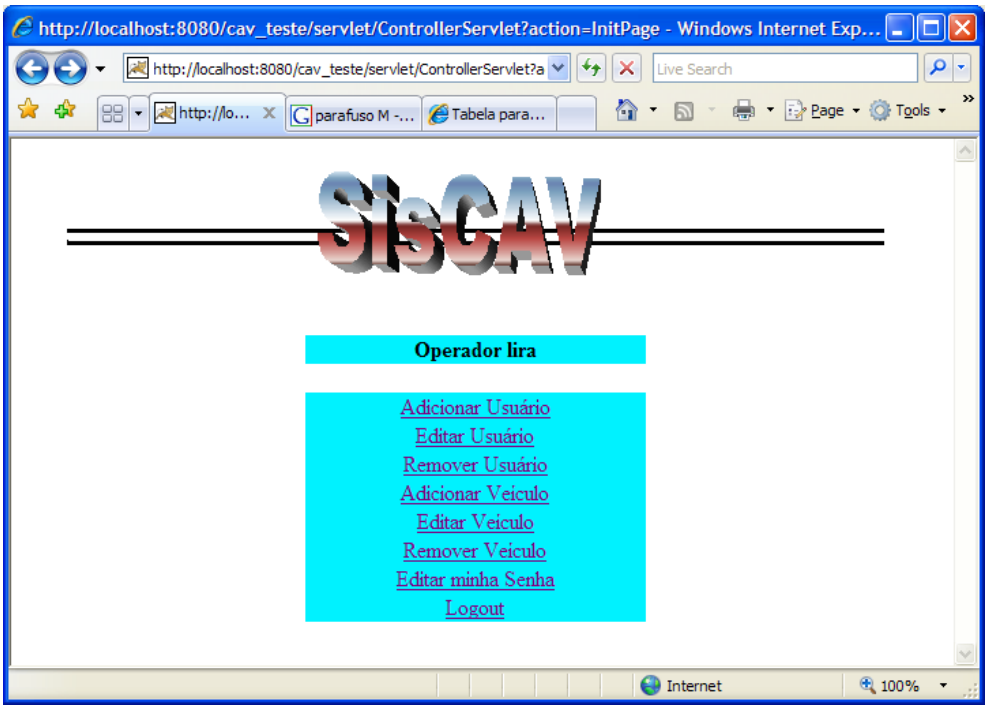


Figura 23 - Menu de Acesso às Funcionalidades do Sistema no Modo Operador

- **Adicionar Usuário**

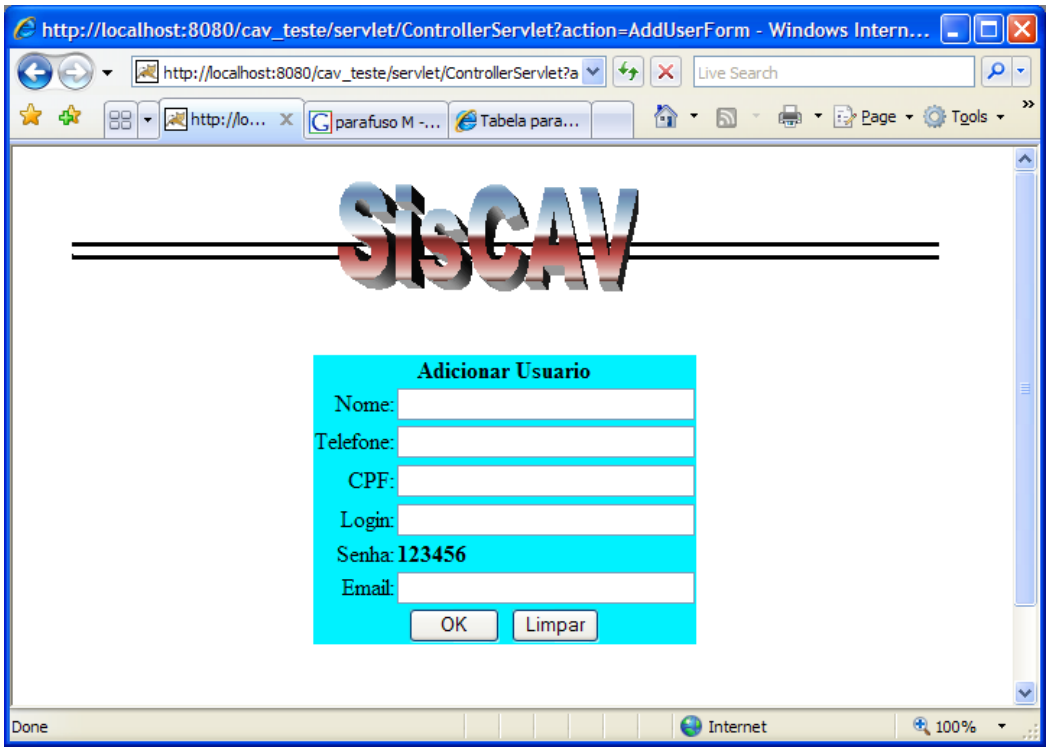


Figura 24 - Cadastro de Novo Usuário

- **Adicionar Veículo**

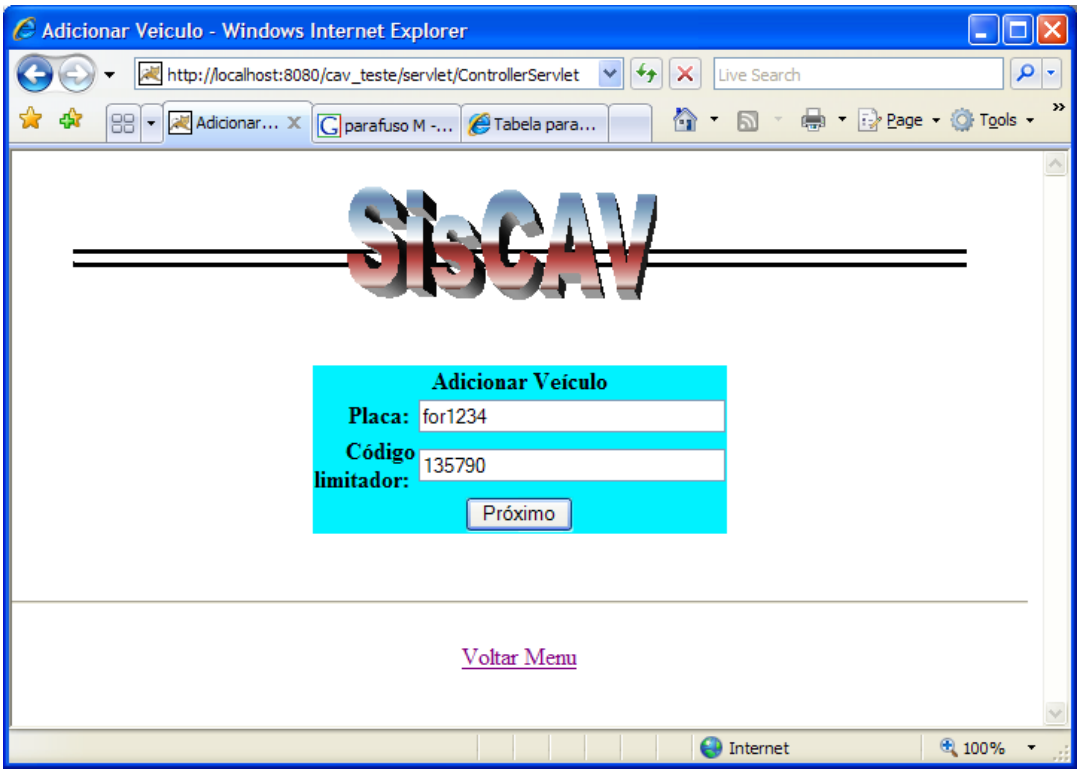


Figura 25 - Entrada de Dados do Veículo

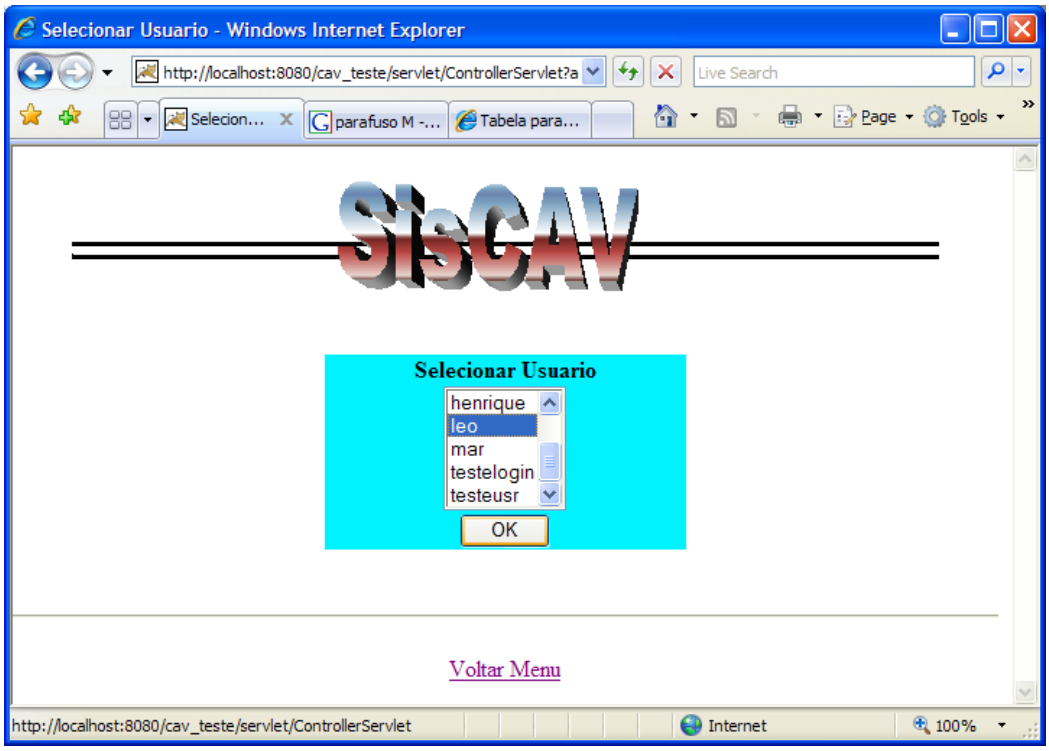


Figura 26 - Seleção de Usuário

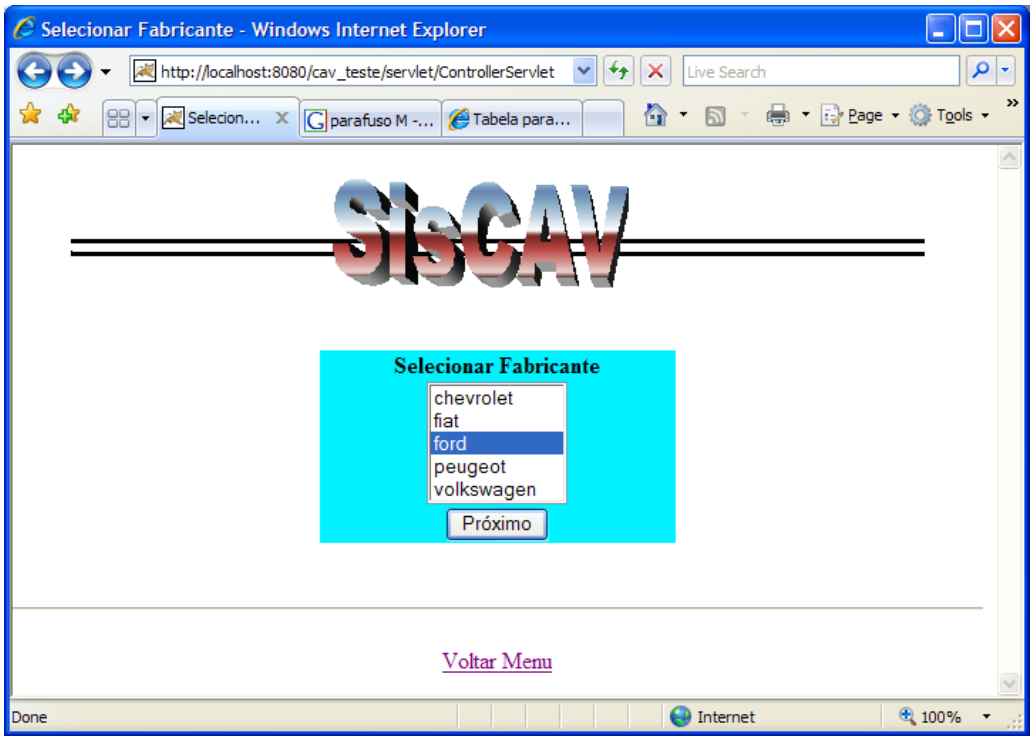


Figura 27 - Seleção de Fabricante

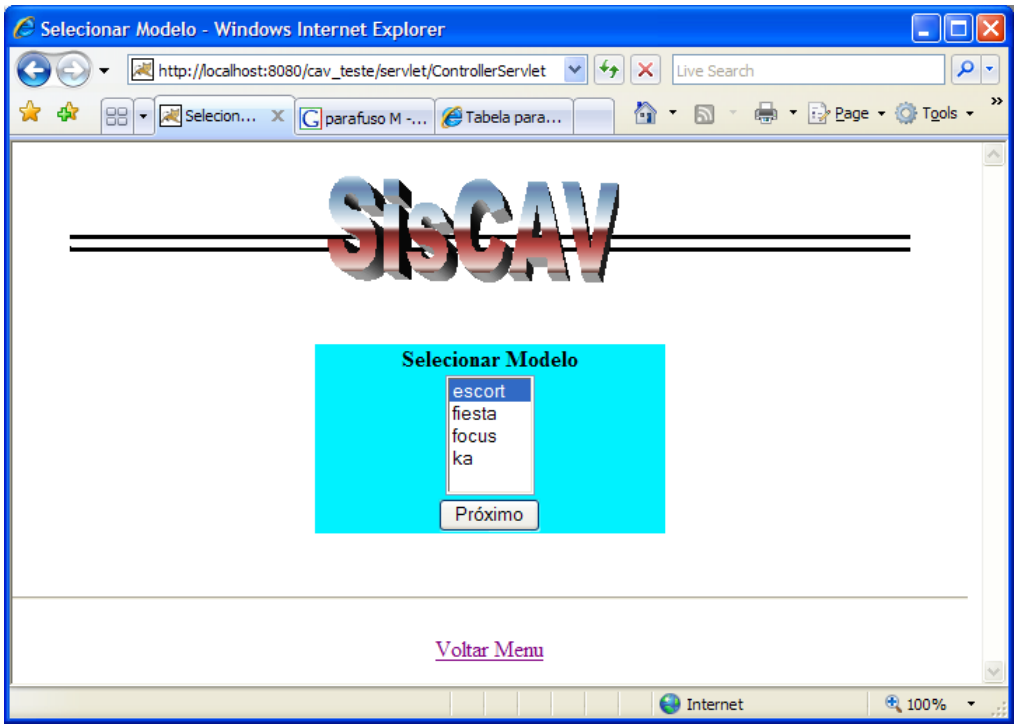


Figura 28 - Seleção de Modelo

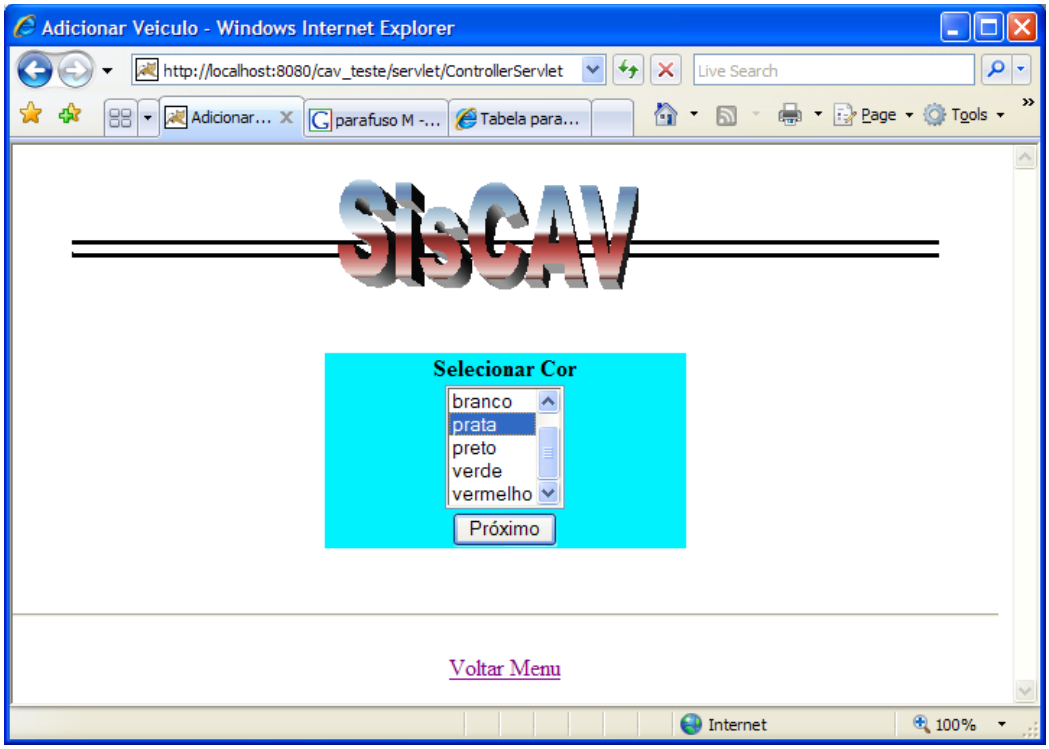


Figura 29 - Seleção de Cor

- **Editar Usuário**

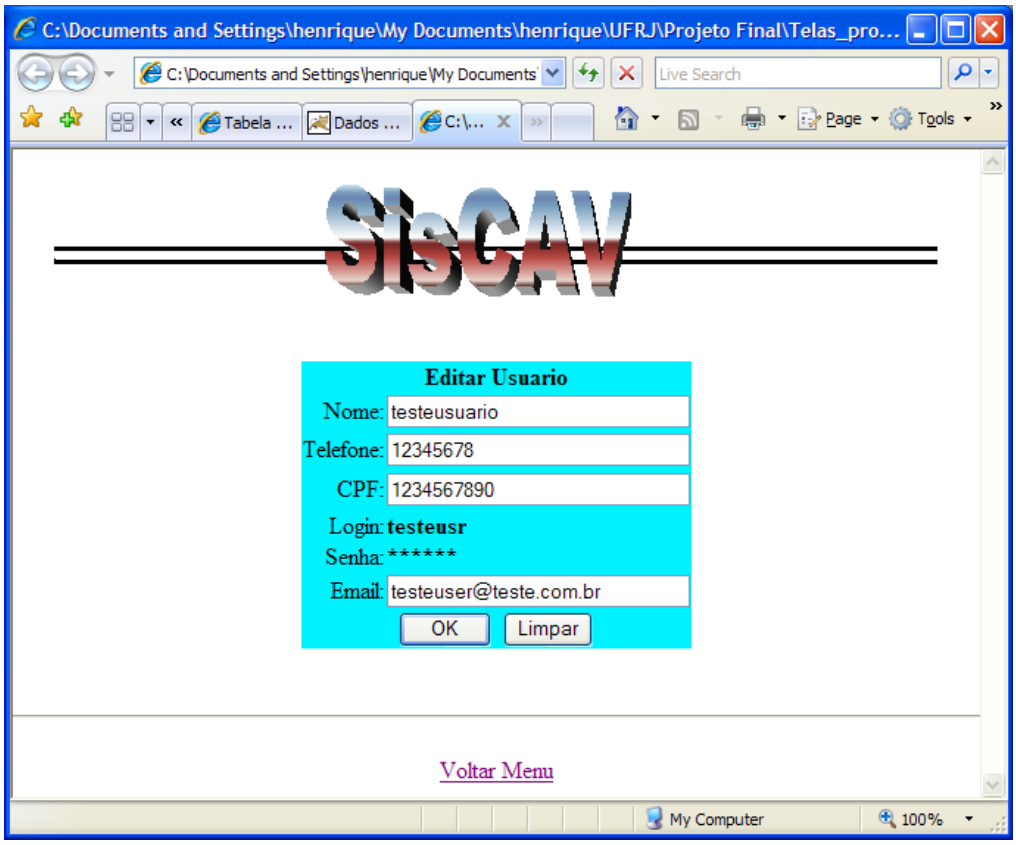


Figura 30 - Alteração de Dados de um Usuário

- **Editar Veículo**

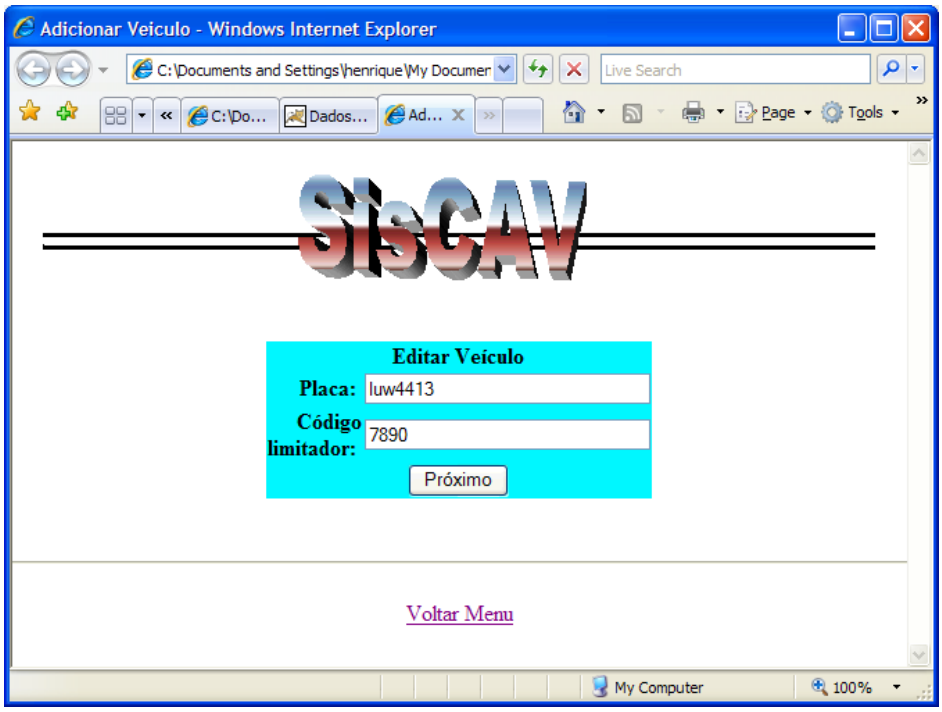


Figura 31 - Alteração de Dados de um Veículo

- **Remover Usuário**

Após escolher a opção “Remover Usuário” uma lista de usuários será mostrada. Após escolher o usuário a ser deletado a seguinte mensagem é exibida:

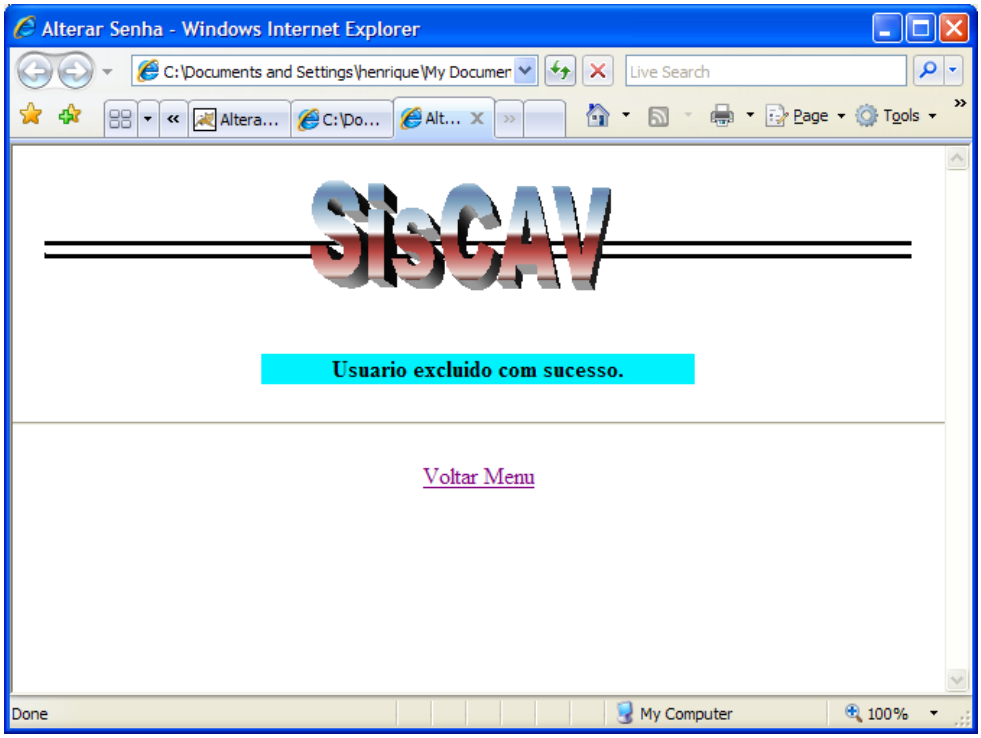


Figura 32 - Exclusão de Usuário

- **Remover Veículo**

Após escolher a opção “Remover Veículo” uma lista de usuários será mostrada. Em seguida a lista de veículos desse usuário será exibida e a seguinte mensagem aparece confirmando a exclusão:

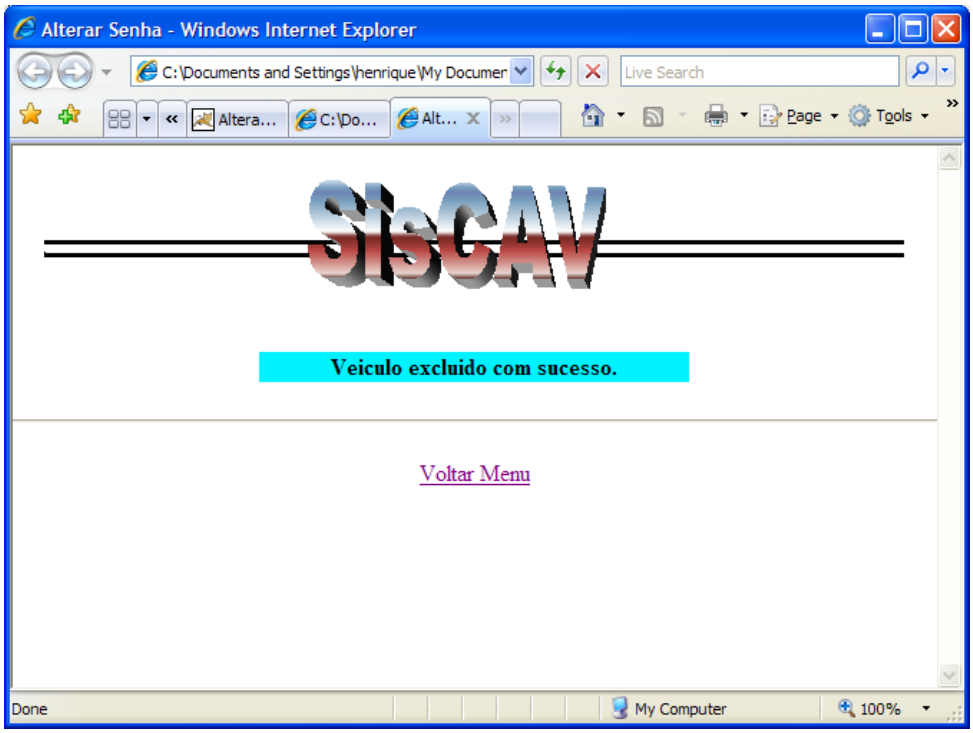


Figura 33 - Exclusão de Veículo

3.2.1.2 Usuário

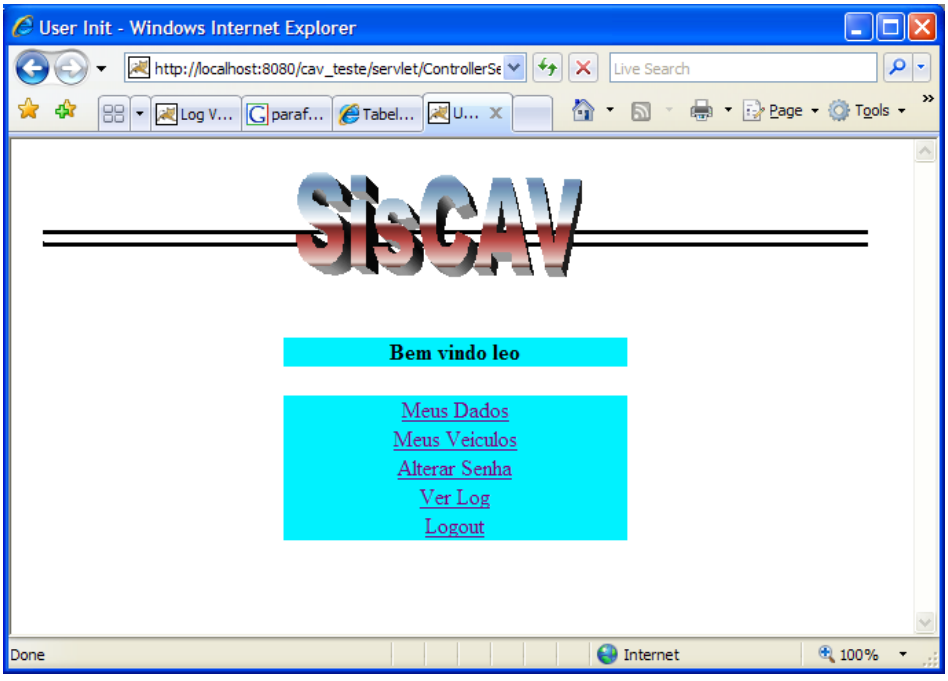


Figura 34 - Menu de Acesso às Funcionalidades do Sistema no Modo Usuário

- **Visualizar Dados Cadastrais**

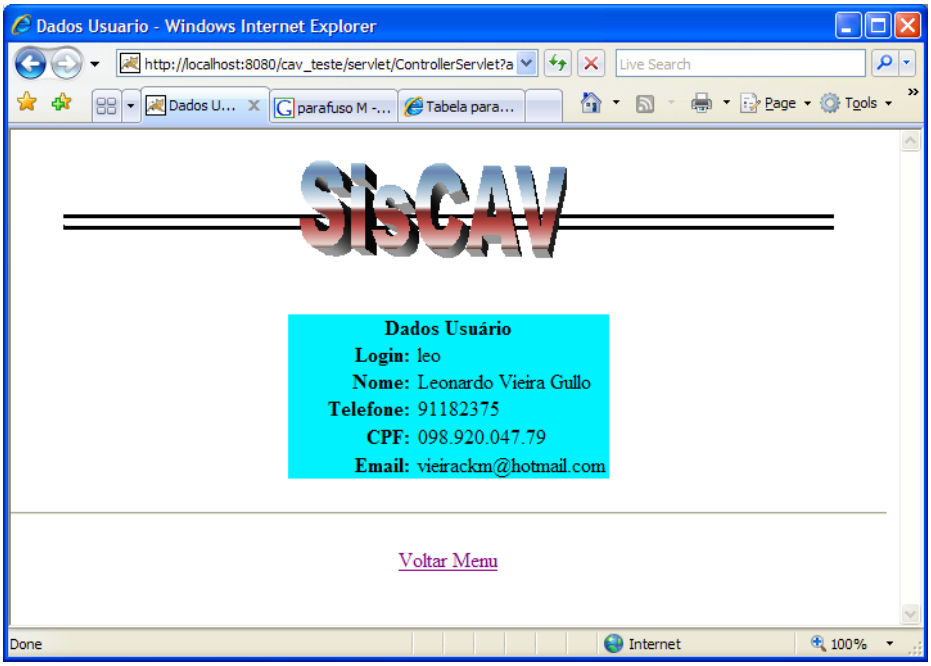


Figura 35 - Visualização de Dados Cadastrais

- **Visualizar Dados do Veículo**

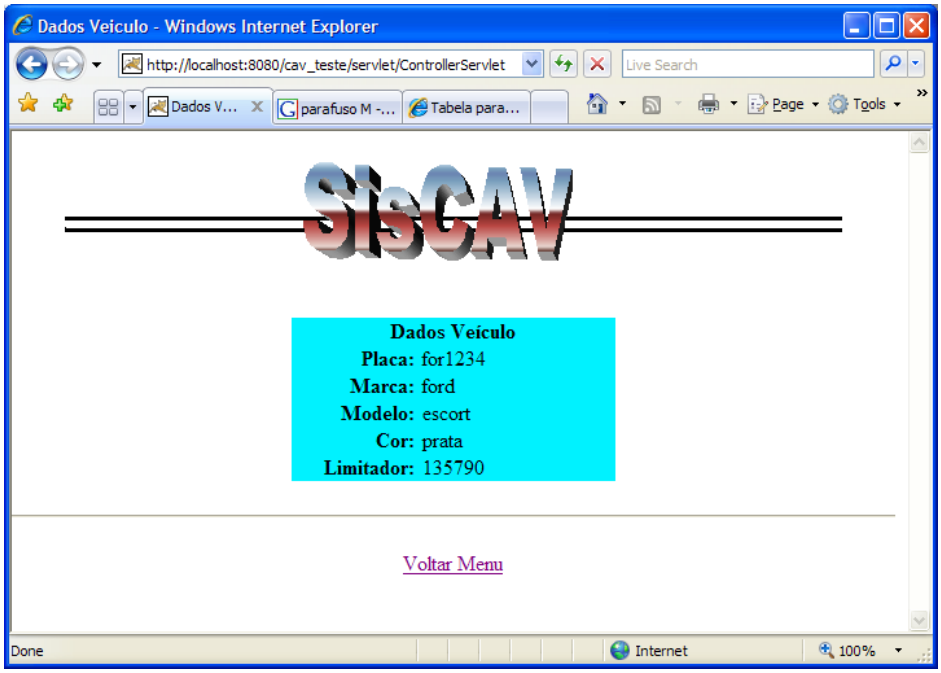


Figura 36 - Visualização de Dados do Veículo

- **Visualizar *Log***

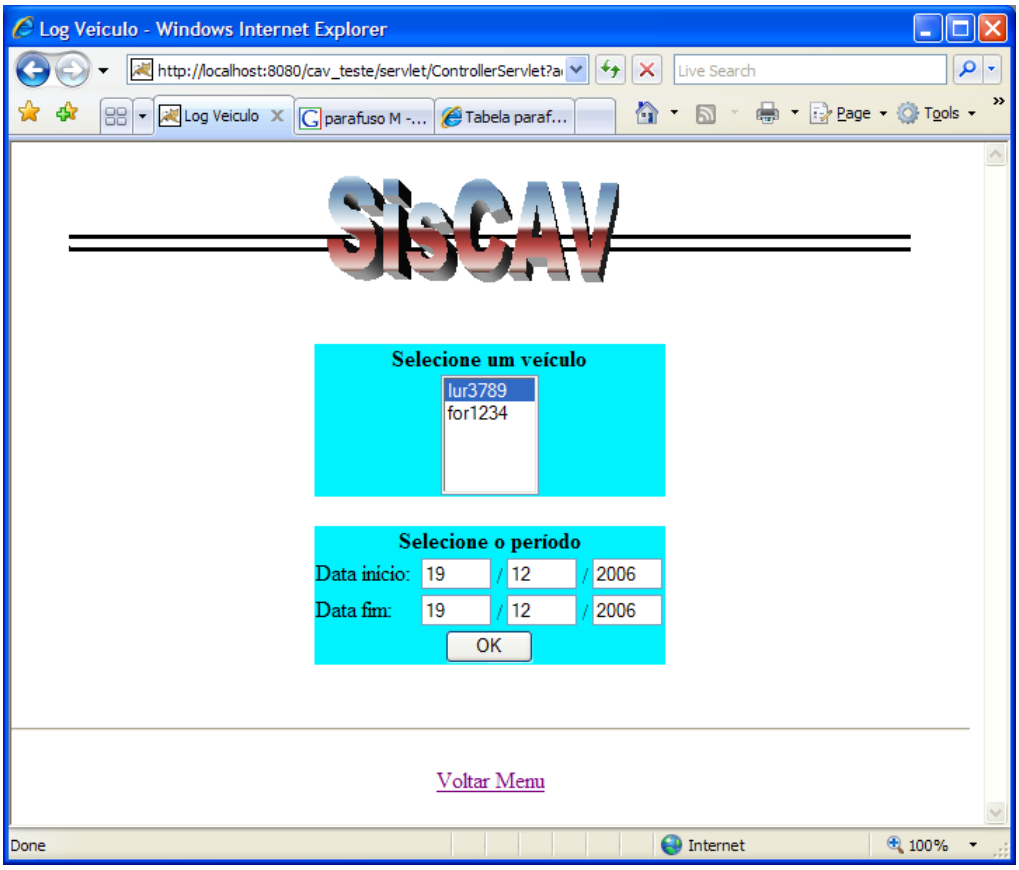


Figura 37 - Lista de Veículos

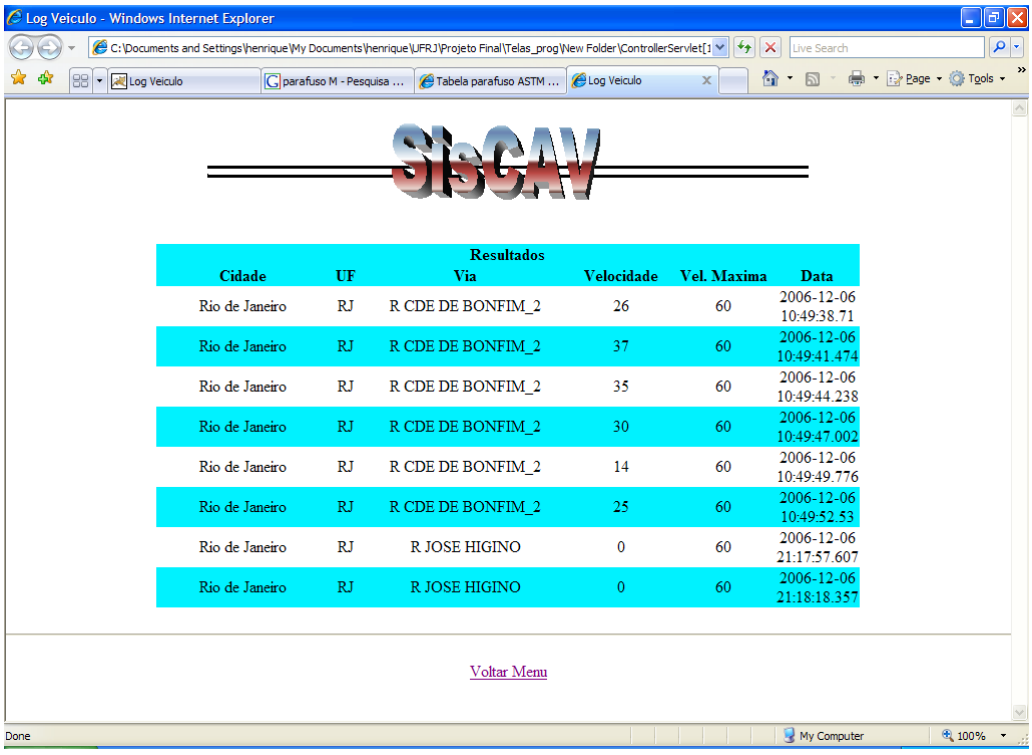


Figura 38 - Visualização de *Log*

3.2.2 Interfaces de Software

Não existirão interfaces entre o software e outro software aplicativo.

3.2.3 Interfaces de Comunicação

A comunicação entre os módulos Localizador e Emulador GPRS é feita via *Socket* TCP/IP. Já a comunicação entre o GPS e o modem assim como a comunicação entre o modem e o Módulo Limitador é feita via RS232.

3.3 Requisitos de Desempenho

Não há grandes requisitos de desempenho para este software. O único parâmetro importante é o tempo de resposta da parte Web. Assim, devemos procurar diminuir o tempo de acesso ao banco de dados que é o gargalo principal do sistema pela necessidade de acesso a disco.

Além disso, tem-se como objetivo máximo deste sistema ter uma alta amigabilidade, pois considerarmos que o usuário foco é imprevisível, podendo ser leigo em diversos aspectos.

3.4 Restrições de Projeto

Não há restrições de projeto.

3.5 Atributos

Os atributos principais para a garantia de qualidade do Sistema de Controle Automotivo de Velocidade são:

- Amigabilidade: o sistema deve ser amigável, tendo uma interface limpa, clara e objetiva.
- Consistência de dados: sistema deve ser consistente, ou seja, os dados do sistema devem ser inseridos de modo fidedigno.
- Segurança e Confiabilidade: o sistema deve ser confiável, já que é o responsável pelo controle de velocidade de um automóvel.
- Usabilidade: este atributo nos levou a utilização da análise orientada a objeto que nos permite maior reutilização de código.
- Escalabilidade: o sistema deve ter uma fácil adaptação para uma demanda maior de usuários.

3.6 Outros Requisitos

Não há outros requisitos de software além dos que já foram citados no documento.

Anexo A – Artigos 60, 61 e 62 do Código Brasileiro de Trânsito

Art. 60. As vias abertas à circulação, de acordo com sua utilização, classificam-se em:

I - vias urbanas:

a) via de trânsito rápido;

b) via arterial;

c) via coletora;

d) via local;

II - vias rurais:

a) rodovias;

b) estradas.

Art. 61. A velocidade máxima permitida para a via será indicada por meio de sinalização, obedecidas suas características técnicas e as condições de trânsito.

§ 1º Onde não existir sinalização regulamentadora, a velocidade máxima será de:

I - nas vias urbanas:

a) oitenta quilômetros por hora, nas vias de trânsito rápido:

b) sessenta quilômetros por hora, nas vias arteriais;

c) quarenta quilômetros por hora, nas vias coletoras;

d) trinta quilômetros por hora, nas vias locais;

II - nas vias rurais:

a) nas rodovias:

1) cento e dez quilômetros por hora para automóveis e camionetas;

2) noventa quilômetros por hora, para ônibus e microônibus;

3) oitenta quilômetros por hora, para os demais veículos;

b) nas estradas, sessenta quilômetros por hora.

§ 2º O órgão ou entidade de trânsito ou rodoviário com circunscrição sobre a via poderá regulamentar, por meio de sinalização, velocidades superiores ou inferiores àquelas estabelecidas no parágrafo anterior.

Art. 62. A velocidade mínima não poderá ser inferior à metade da velocidade máxima estabelecida, respeitadas as condições operacionais de trânsito e da via.