Universidade Federal do Rio de Janeiro

Escola Politécnica

Departamento de Eletrônica e de Computação

# Sistema de Gerenciamento de Frotas Empregando Informações Georeferenciadas

Autores:

Danilo Vannier Cunha

Leandro Tavares Aragão dos Santos

Orientador:

Prof. Flávio Luis de Mello, D. Sc.

Examinador:

Prof. Edilberto Strauss Ph.D.

Examinador:

Prof. Sérgio Barbosa Villas-Boas Ph.D.

DEL

Junho de 2009

## DEDICATÓRIA

Dedico este trabalho aos meus pais, Carlos Sérgio Aragão dos Santos e Maria Suely de Oliveira Tavares, e à minha avó, Maria Ofélia de Oliveira. Estes, de forma incondicional, apoiaram meus estudos. Sempre me protegendo, evitaram que eu enfrentasse as mesmas dificuldades que experimentaram no decorrer da vida, tornando minha caminhada bem mais amena.

Leandro Tavares Aragão dos Santos

**AGRADECIMENTO**

Agradeço ao Flávio pela idéia do projeto e por toda a orientação, à Nathalie pelas noites que passei em claro e aos meus pais pela educação depositada em mim. Além disso, dedico este trabalho ao povo brasileiro que contribuiu de forma significativa à minha formação e estada nesta Universidade.

Danilo Cunha

Agradeço à minha família pelo apoio incondicional à minha jornada. À minha noiva Monique da Silva Garcia, pela compreensão em relação à minha ausência nos momentos dedicados a esta empreitada. Ao Prof. Flávio Luis de Mello, pela idealização do projeto e por toda a orientação e presteza no decorrer da elaboração deste trabalho. E, finalmente, agradeço ao povo brasileiro que financiou os meus estudos de graduação nesta universidade.

Leandro Tavares Aragão dos Santos

# RESUMO

Sistemas de gerenciamento de frotas geralmente são muito úteis, mas também podem custar muito caro. Esse projeto propõe um sistema simples para o gerenciamento de frotas, utilizando informações georeferenciadas, sem com tudo deixar de ser eficiente e de apresentar baixo custo. Para tanto, foi utilizado a API de desenvolvimento do Google Maps de modo a integrar mapas. O sistema foi desenvolvido para monitorar viaturas, cadastrar chamados e ajudar um operador a decidir qual viatura vai atender qual chamado.



Palavras-Chave: API do Google Maps, gerenciamento de frotas, GPS, georeferenciamento.

# ABSTRACT

Although fleet management systems are extremely useful, they are also very expensive. This project is about a simple and efficient fleet management system that is also inexpensive. In order to achieve this goals we used the Google Maps API. The system was designed to track vehicles, register new calls and help operators decide which vehicles will attend which call.


Key-words: Google Maps API, fleet management, GPS, georeferece.

## SIGLAS

API – Application Programming Interface

CNPJ – Cadastro Nacional de Pessoa Jurídica

CORBA – Common Object Request Broker Architecture

CPF – Cadastro de Pessoa Física

DAO – Data Access Object

EGNOS – European Geostationary Navigation Overlay Service

GPS – Global Positioning System

HTTP – Hypertext Transfer Protocol

IP – Internet Protocol

JDBC – Java Database Connectivity

JNDI – Java Naming and Directory Interface

JSP – JavaServerPage

LDAP – Light Weight Directory Protocol

MVC – Model View Controller

NMEA – National Marine Electronics Association

RMI – Remote Metode Invocation

SDK – Software Development Kit

SPI – Service Provider Interface

SQL – Structured Query Language

UFRJ – Universidade Federal do Rio de Janeiro

W3C – World Wide Web Consortium

XML – Extensible Markup Language

# Sumário


**1 Introdução** **1**

1.1 – Tema . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 – Delimitação . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.3 – Justificativa . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.4 – Objetivos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.5 – Descrição . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**2 Fundamentação Teórica** 4

2.1 – Central de Atendimento . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2 – API do Google Maps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2.2.1 – Condições de uso da API do Google Maps . . . . . . . . . . . 4

2.2.2 – Principais elementos da API do Google Maps . . . . . . . 6

2.3 – GPS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2.4 – XML . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2.5 – MVC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.6 – Servlets e Jsps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.7 – JDBC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.8 – JNDI . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

**3 Metodologia** **18**

3.1 – Arquitetura do projeto . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.2 – Modelagem de dados . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

3.3 – Gerenciamento de conexões ao SGBD . . . . . . . . . . . . . . . . . 27

3.4 – Gerenciamento das atualizações da aplicação principal . . . . . . 29

3.5 – Concorrência de acesso ao data.xml . . . . . . . . . . . . . . . . . . . . . . 32

3.6 – Levantamento de custos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34

**4 Conclusões 36**

**Bibliografia 37**

# Lista de Figuras

2.1 – Página inicial do Google Maps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5

2.2 – Atribuições dos objetos da API do Google Maps . . . . . . . . . . . . . . . . . . . . . 7

2.3 – Exemplo de sentença utilizada no protocolo NMEA 183 . . . . . . . . . . . . . . . 9

2.4 – Arquitetura MVC . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

2.5 – Arquitetura web com JSP e Servlet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13

2.6 – Exemplo de aplicação com JSP e Servlet . . . . . . . . . . . . . . . . . . . . . . . . . . 14

2.7 – Arquitetura JNDI. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

3.1 – Página da aplicação Controle de Frotas . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.2 – Legenda . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.3 – Lista de chamados . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

3.4 – Lista de viaturas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.5.a – Informações de viaturas . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.5.b – Informações de chamados . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 20

3.6 – Centralizando e dando um zoom no chamado 31 . . . . . . . . . . . . . . . . . . . . 21

3.7 – Esquema resumido do funcionamento do sistema . . . . . . . . . . . . . . . . . . . 21

3.8– Formulário de chamados . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22

3.9 – Endereços sugeridos . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.10 – Arquitetura do projeto . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24

3.11 – Modelagem de dados . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

3.12 – Detalhes dos principais aspectos da modelagem . . . . . . . . . . . . . . . . . . . 26

3.13 – Detalhes da modelagem do endereço . . . . . . . . . . . . . . . . . . . . . . . . . . . 26

3.14 – Detalhes da modelagem de pessoa . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 27

3.15 – Primeira alternativa de conexão ao SGBD . . . . . . . . . . . . . . . . . . . . . . . . 27

3.16 – Gerenciamento de conexões . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 28

3.17 – Gerenciamento de atualizações, solução imediata . . . . . . . . . . . . . . . . . . 29

3.18 – Gerenciamento de atualizações, segunda alternativa . . . . . . . . . . . . . . . . . . 30
3.19 – Gerenciamento de atualizações, solução mais eficiente . . . . . . . . . . . . . . . 31
3.20 – Gerenciamento de concorrência de acesso ao data.xml . . . . . . . . . . . . . . . . 33

# Lista de Tabelas

3.1 – Comparação de Conexões . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29

3.2 – Custo das alternativas á API do Google Maps . . . . . . . . . . . . . . . . . . . . . . . 34

# Capítulo 1

# Introdução

## 1.1 – Tema

Este trabalho é sobre um sistema de apoio à decisão para controle de frotas. O sistema foi pensado para ser usado por cooperativas, ou empresas, de táxi e para frotas de ambulância, podendo também ser adaptado para outros casos semelhantes. O sistema provê informações georeferenciadas para os operadores que decidem qual viatura atenderá qual chamado.

## 1.2 – Delimitação

Esse trabalho se destina as situações onde haja decisão, ou simplesmente acompanhamento, de agentes móveis sobre um mapa do Google Maps.

O projeto foi concebido inicialmente para gerenciamento de ambulâncias ou táxis, podendo ser usado em diversas outras aplicações. O sistema poderia ser empregado no gerenciamento de aviões militares e seus respectivos alvos, helicópteros e heliportos, caminhões de bombeiros e emergências, carros fortes e bancos, moto-boys e locais de entrega, enfim, sistemas de entrega ou distribuição em geral.

O sistema de posicionamento dos agentes não está no escopo desse projeto, porém, sua futura adaptação foi prevista. Pode ser usado qualquer sistema de posicionamento que se baseie em coordenadas geográficas e seja capaz de atualizar as coordenadas dos agentes no banco de dados.

## 1.3 – Justificativa

Com o crescimento dos centros urbanos, das populações e consequentemente do trânsito, as soluções envolvendo sistemas de posicionamento global estão cada vez mais presentes.

Com a popularização do GPS, os sistemas de múltiplas viaturas começaram a demandar uma solução integrada de controle. Algo que apresentasse, em tempo real, a posição de todas as viaturas e seus destinos no mesmo sistema.

O sistema implementado nesse projeto atende a essa demanda de centralizar o gerenciamento de várias viaturas utilizando tecnologias de baixo custo. Os mapas utilizados são disponíveis gratuitamente através da API do Google Maps, o sistema foi feito para se operar através de uma interface web e todas as ferramentas de desenvolvimento e padrões adotados são gratuitas.

As empresas ou cooperativas que possuem múltiplas viaturas podem se interessar nesse projeto visando aumentar a eficiência e agilidade nos atendimentos. Com isso, reduziriam custos e aumentariam a qualidade do serviço.

## 1.4 – Objetivos

O objetivo desse trabalho é apresentar um sistema de controle de frota utilizando a API de desenvolvimento do Google Maps. Espera-se desse sistema a possibilidade de cadastrar viaturas, pessoas e chamados.

O sistema visa a auxiliar os operadores que direcionam as viaturas até os chamados. Os operadores terão acesso a um mapa onde estarão as viaturas e os chamados. Com esse mapa o operador poderá decidir com mais clareza qual é a melhor viatura para cada chamado.

Além do programa principal, com o mapa dispondo as viaturas e os chamados, haverá uma interface com o banco de dados. Nessa serão feitos os cadastros das pessoas, das viaturas e dos chamados. Além disso, será nessa mesma interface os chamados poderão ser associados às viaturas que venham a atender os mesmos.

Fora a interface com o banco de dados, e o programa principal, haverá ainda uma outra aplicação dedicada a simular o movimento das viaturas. Esse programa fará o papel do módulo receptor dos sinais de GPS, possibilitando a realização de testes no projeto como um todo.

O objetivo global do projeto é apresentar uma solução para controle de frotas. Para atender a esse objetivo, será necessário o desenvolvimento dos três sub-projetos mencionados acima: o projeto principal com o mapa, a interface com o banco de dados e o simulador do movimento das viaturas.

## 1.5 – Descrição

No capítulo 2, será apresentada a fundamentação teórica. Serão abordados as principais tecnologias adotadas no desenvolvimento do projeto e os principais problemas enfrentados. Mais especificamente, serão apresentados os conceitos da API do Google Maps, o uso de XML para integrar diferentes partes do mesmo sistema, o funcionamento do GPS e a dinâmica e organização de um Call Center.

O capítulo 3 apresenta a metodologia usada no projeto. Os sub-projetos implementados para alcançar todos os objetivos, a arquitetura de software adotada, a modelagem de dados, o gerenciamento das conexões ao banco de dados e a solução para sincronizar as alterações feitas no banco com o programa principal.

As conclusões, bem como os resultados alcançados e os próximos passos, serão tratados no quarto e último capítulo.

# Capítulo 2

# Fundamentação Teórica

## 2.1 – Central de Atendimento

Uma Central de Atendimento (ou um Call-Center) é uma instalação onde as ligações são concentradas. As ligações são automaticamente distribuídas para os atendentes. Os atendimentos podem ser feitos também por chat ou por email. Muitas vezes os atendimentos são terceirizados para empresas especializadas em Call-Center.

No caso de centrais de atendimento de ambulâncias, os atendentes registram os chamados por telefone. Para cada chamado devem ser registradas informações suficientes para levar a ambulância até o local do sinistro, identificar e levar o solicitante até o hospital.

Nas empresas de táxi, as informações registradas são suficientes para localizar, identificar o solicitante e levá-lo para seu destino. Além disso, algumas vezes, informações como CPF ou CNPJ são necessárias para emissão de nota fiscal.

No caso desse projeto, os atendentes da central de atendimento deverão cadastrar os chamados utilizando o sub-projeto “Formulários”. Este é uma interface web onde os atendentes podem fazer os cadastros no banco de dados. No terceiro capítulo, o funcionamento desse sub-projeto e do acesso ao banco de dados será mais detalhado.

## 2.2 – API do Google Maps

### 2.2.1 – Condições de uso da API do Google Maps

O Google Maps é um serviço de visualização de mapas prestado pela Google sem rigor cartográfico certificado. Através do site http://maps.google.com, a Google disponibiliza o acesso a um mapa detalhado com opções de busca de endereço, localização de pontos comerciais e traçado de rotas.

Na figura 2.1, podemos ver a página inicial do Google Maps. Colocando um endereço no campo de busca, o mapa automaticamente centraliza no resultado. Caso a busca retorne mais de um endereço possível, as opções aparecem no menu da esquerda. Ao clicar em uma das opções, o mapa centraliza no endereço desejado.

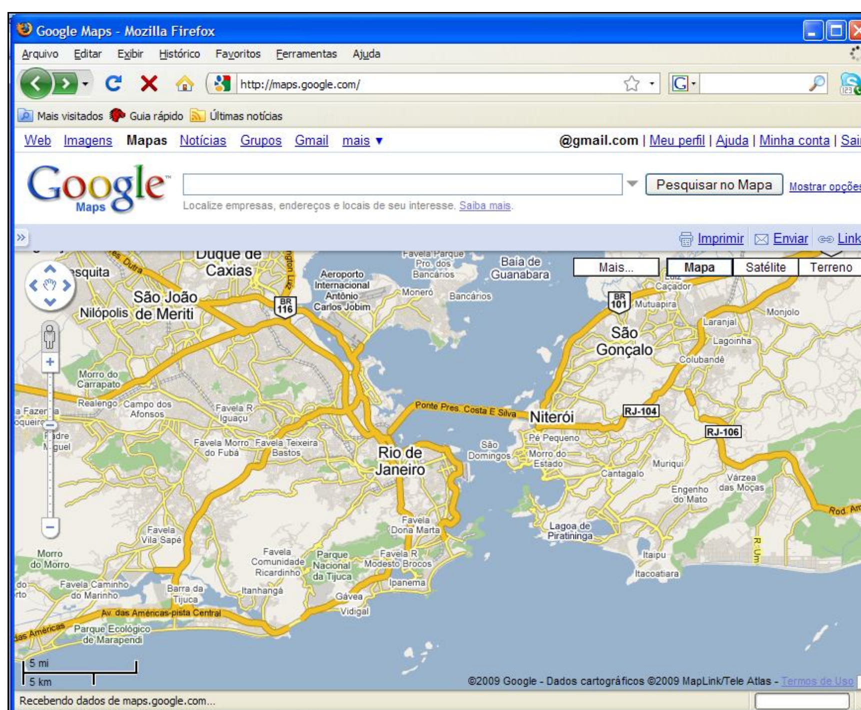


Figura 2.1 – Página inicial do Google Maps

O Google Maps possui uma API de desenvolvimento [3]. Essa API permite desenvolver aplicativos utilizando os serviços do Google Maps. A API pode ser acessada utilizando-se as linguagens de programação JavaScript ou Flash.

Os requisitos para se utilizar a API são poucos: o usuário final deve ter livre acesso ao sistema que venha a utilizar a API, os logotipos ou créditos do mapa devem ser mantidos, os serviços prestados não podem envolver qualquer tipo de atividade ilícita ou identificar informações particulares sobre indivíduos.

Não há limite de número de visualizações geradas através da API, entretanto, é recomendado entrar em contato com a equipe do Google Maps caso haja mais de 500 mil acessos por dia. Dessa maneira, mais recursos serão disponibilizados para processar todos os acessos.

Existe uma limitação imposta ao número de Geocodes solicitados. Estes Geocodes são usados para converter um endereço em coordenadas geográficas. O limite é de 15 mil solicitações de Geocodes em um mesmo dia pelo mesmo endereço de IP.

Quando os Geocodes são solicitados através da API, a solicitação ocorre no navegador do usuário final, portanto o IP solicitando Geocodes será o dele. Mas se as solicitações de Geocodes forem feitas através de um webserver, o IP solicitando Geocodes será o do servidor. No segundo caso o limite de solicitações diárias pode ser um problema.

Caso o usuário final não tenha livre acesso a um sistema que use a API do Google Maps, seja por motivos comerciais ou não, ou esse sistema necessite fazer mais solicitações de Geocodes do que o limite permitido, existe a API Premier. A API Premier não possui as limitações de solicitações de Geocodes nem as limitações comerciais. Entretanto, a API mais versátil não é gratuita.

Para utilizar a API do Google Maps é necessário obter uma chave de acesso, a qual pode ser obtida no link http://code.google.com/intl/pt-BR/apis/maps/signup.html. A chave é gerada a partir do domínio do site que a API será utilizada e funcionará no domínio que foi utilizado para gerar a chave. Caso haja uma mudança de domínio uma nova chave precisará ser gerada. Contudo, se ocorrerem mudanças de subpastas no mesmo domínio, não haverá restrições e a chave inicial continuará válida.

### 2.2.2 – Principais elementos da API do Google Maps

**GMap2:**

O principal elemento presente na API do Google Maps é o mapa. O objeto que representa o mapa é o GMap2. O mapa precisa ser inicializado e depois carregado para então poder ser utilizado.

**GLatLng:**

O objeto GLatLng representa uma coordenada geográfica, ou seja, um ponto no mapa. É muito utilizado por outros objetos da API. Esse objeto guarda os valores de latitude e longitude.

**GMarker:**

O objeto GMarker representa um marcador no mapa, isto é, um símbolo pontual. O objeto GMarker apresenta diversas opções de configuração e de gerenciamento de eventos. Um exemplo de funcionalidade interessante utilizado no projeto é um balão de informações que aparece quando o marcador é clicado. O ícone que representa o marcador pode ser trocado por qualquer outro ícone. As viaturas e os chamados no projeto são representados por marcadores.

**GClientGeocoder:**

Através do objeto GClientGeocoder é possível converter endereços em coordenadas geográficas. O mesmo pode ser feito através do servidor sem o auxílio do objeto GClientGeocoder, enviando uma solicitação para a url http://maps.google.com/maps/geo, e passando o endereço, a chave e a forma que a saída será gerada.

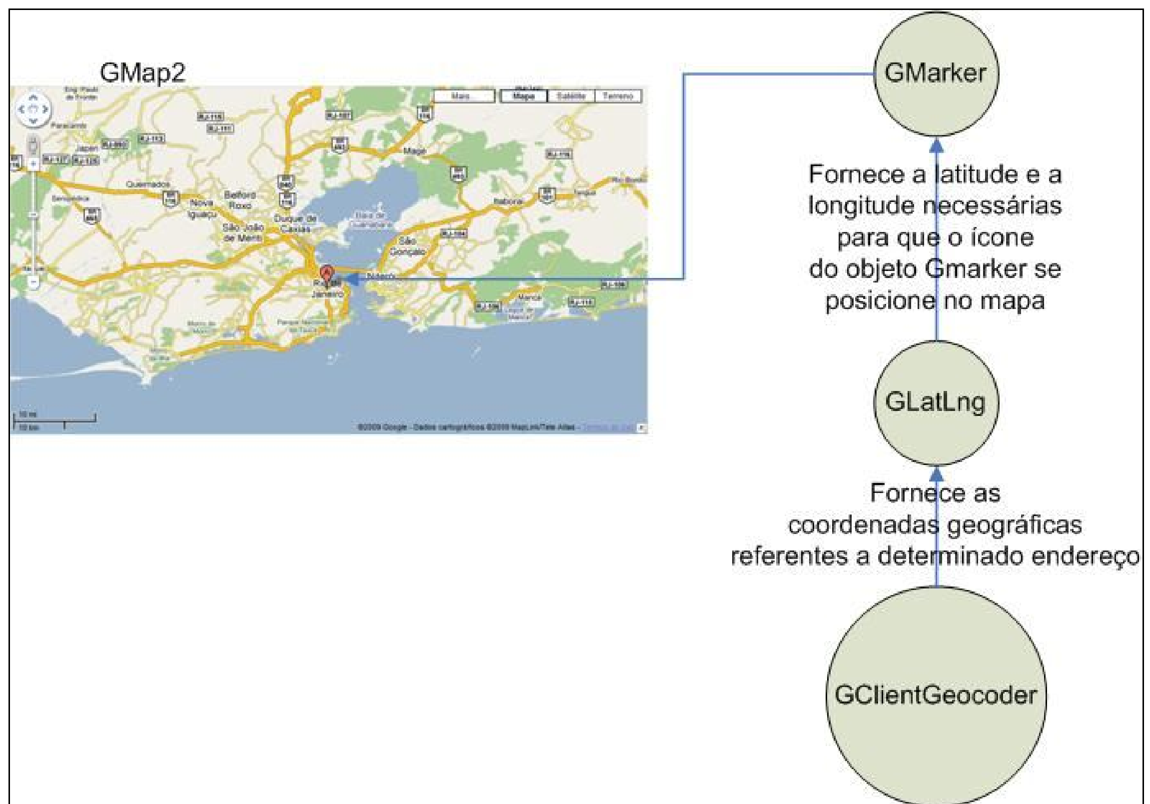


Figura 2.2 – Atribuições dos objetos da API do Google Maps

As principais diferenças entre a utilização do objeto GClientGeocoder e as consultas ao servidor utilizando o link mencionado acima, são a utilização da cota diária de solicitação de Geocodes e algumas opções disponibilizadas pelo objeto. Como o limite diário de solicitações é por IP, o uso do GClientGeocoder faz com que o IP considerado seja o do usuário final, e ao usar o webserver o IP considerado será o do servidor. As principais opções disponibilizadas pelo objeto GClientGeocoder são: o uso de cache de Geocodes, que possibilita o armazenamento das solicitações mais recentes no computador do usuário reduzindo o tempo da conversão, e a restrição dos endereços

sugeridos em uma caixa de visão, ou seja em uma janela com os limites pré-estabelecidos.

## 2.3 – GPS

O GPS, ou Sistema de Posicionamento Global, é um sistema de navegação através de satélites. Um aparelho receptor de GPS se comunica com algum dos satélites presentes em uma constelação de 24 a 32 satélites de órbita terrestre média.

Satélites de órbita terrestre média levam um período de 12 horas para completar uma volta em torno da Terra. A altitude que esses satélites se encontram é de aproximadamente 20.000 km. Neste tipo de órbita, os satélites são mais estáveis, pois ficam fora da atmosfera terrestre [7].

Os satélites, através de sinais de rádio, provêem a posição e a velocidade do aparelho receptor, além da hora exata. A comunicação entre o satélite e o aparelho receptor é definida de acordo com um protocolo. Na maioria dos dispositivos existentes no mercado, o protocolo utilizado é o NMEA 183. Este protocolo define padrões de sentenças a serem transmitidas. As sentenças contêm identificação de quem as enviou, identificação do tipo de sentença utilizada e os dados definido para este tipo de sentença. Segue na figura 2.3, um exemplo de sentença do padrão NMEA183 utilizado por aparelhos receptores de GPS.

**GGA Global Positioning System Fix Data. Time, Position and fix related data for a GPS receiver**

```
                                                      11
       1         2       3 4        5 6 7  8   9  10 |  12 13  14   15
       |         |       | |        | | |  |   |   | |   | |   |    |
$--GGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx*hh

 1) Time (UTC)
 2) Latitude
 3) N or S (North or South)
 4) Longitude
 5) E or W (East or West)
 6) GPS Quality Indicator,
    0 - fix not available,
    1 - GPS fix,
    2 - Differential GPS fix
 7) Number of satellites in view, 00 - 12
 8) Horizontal Dilution of precision
 9) Antenna Altitude above/below mean-sea-level (geoid)
10) Units of antenna altitude, meters
11) Geoidal separation, the difference between the WGS-84 earth
    ellipsoid and mean-sea-level (geoid), "-" means mean-sea-level below ellipsoid
12) Units of geoidal separation, meters
13) Age of differential GPS data, time in seconds since last SC104
    type 1 or 9 update, null field when DGPS is not used
14) Differential reference station ID, 0000-1023
15) Checksum
```

Figura 2.3 – Exemplo de sentença utilizada no protocolo NMEA 183 [8].

O aparelho receptor precisa se conectar com pelo menos 4 satélites para poder calcular com precisão suficiente (em torno de 10 metros podendo variar de acordo com o aparelho e com sua localização, [9]) a posição do aparelho. Apesar de só haverem 3 dimensões, 3 satélites não são suficientes para a localização, pois um pequeno erro significa um erro de posicionamento significativo. O aparelho receptor usa a distorção do efeito Doppler para calcular a posição do satélite em sua órbita.

Os aparelhos receptores geralmente possuem mapas carregados na memória. O aparelho relaciona a coordenada geográfica com o mapa carregado. Tendo a própria posição no mapa, os aparelhos são capazes de traçar rotas, localizar pontos de interesse, como postos de gasolina, hotéis, etc.

O sistema de GPS foi criado pelo Departamento de Defesa do governo dos Estados Unidos e, hoje em dia, é gerenciado pelo *United States Air Force 50th Space Wing*.

Inicialmente o sistema de GPS foi desenvolvido durante o mandato do presidente norte americano Ronald Reagan. O projeto começou após o avião coreano KAL 007 ser derrubado por ter erroneamente entrado no espaço aéreo soviético em 1983. Os erros foram atribuídos à falhas no sistema de navegação da aeronave. Projetos

de navegação por satélites já existiam para uso militar e, após o incidente com o KAL 007, um projeto de GPS foi feito para uso público.

O sistema Europeu de navegação estacionária, o EGNOS, entrou em operação em 2006. O sistema Europeu conta com três satélites e uma malha de estações terrestres que complementam os sistemas de GPS existentes. O EGNOS apresenta informações sobre a confiabilidade do sinal recebido pelos satélites dos outros sistemas de GPS. Com o sistema Europeu, o erro de posicionamento do aparelho receptor diminuiu para cerca de 2 metros [10].

## 2.4 – XML

XML (eXtensible Markup Language) é uma linguagem genérica capaz de descrever e armazenar qualquer tipo de dado. Trata-se de uma ferramenta muito útil para integrar diferentes sistemas ou para compartilhar dados através da internet. O XML é a linguagem de marcação desenvolvida pelo W3C (World Wide Web Consortium). É uma linguagem caracterizada pelas *tags* (marcações). Estas *tags* são definidas de acordo com a necessidade e de acordo com os dados armazenados, podendo inclusive conter outras *tags*. Geralmente, a organização e hierarquização destas marcações descrevem as relações entre as variáveis armazenadas. Além das *tags*, XML possui atributos que complementam com informações adicionais as *tags*.

No exemplo abaixo os dados que descrevem uma viatura estão organizados em um XML.

```
<Viatura tipo=”Ambulância”>
    <Placa>XYZ1234</Placa>
    <Coordenadas>
        <Latitude>-43,25822</Latitude>
        <Longitude>15,12544</Longitude>
    </Coordenadas>
    <Status>Livre</Status>
</Viatura>
```

Exemplo de Viatura armazenada em um XML.

Um arquivo XML contém apenas texto, sendo os programas que forem utiliza-lo responsáveis por providenciar as conversões de tipo de dado necessárias. A extensão de um arquivo XML é “.xml”.

## 2.5 – MVC

MVC (Model View Controller) é um padrão de projeto, voltado para o desenvolvimento de software, muito comum em aplicações web. Aplicações que não separam a interface, com o usuário, do acesso ao banco de dados e da lógica de negócio tendem a ser de difícil manutenibilidade. Além disso, sistemas com esse tipo de arquitetura, conforme sofrem sucessivas evoluções e manutenções, acabam ficando com uma quantidade excessiva de código repetido. Tratamento de erros e exceções também se tornam complicados e pouco confiáveis em sistemas sem separação de camadas. Muitas vezes, a divisão do projeto em camadas é confundida erradamente com a arquitetura MVC. De fato, esta última trata de como as camadas interagem.

Na arquitetura MVC (ver figura 2.4), o model é responsável por organizar os elementos do sistema em classes, cuidando de toda a lógica do negócio. Muitas vezes também, realizando a persistência dos dados. As relações entre os elementos do sistema e toda a problemática da orientação a objetos ficam restritas ao model.

O controller, por sua vez, cuida dos eventos (ou ações) que são disparados pelo usuário na interface, realizando as mudanças necessárias no model. A validação dos dados também é feita no controller.

Por fim, a view geralmente corresponde a interface. Ela usa as informações que estão organizadas no model para formatar as telas (ou páginas) que serão apresentadas para o usuário. Além disso, as ações (ou eventos) do usuário, em geral, começam na view. De maneira resumida, o controller dispara as solicitações para o model e a view apresenta o model para o usuário.

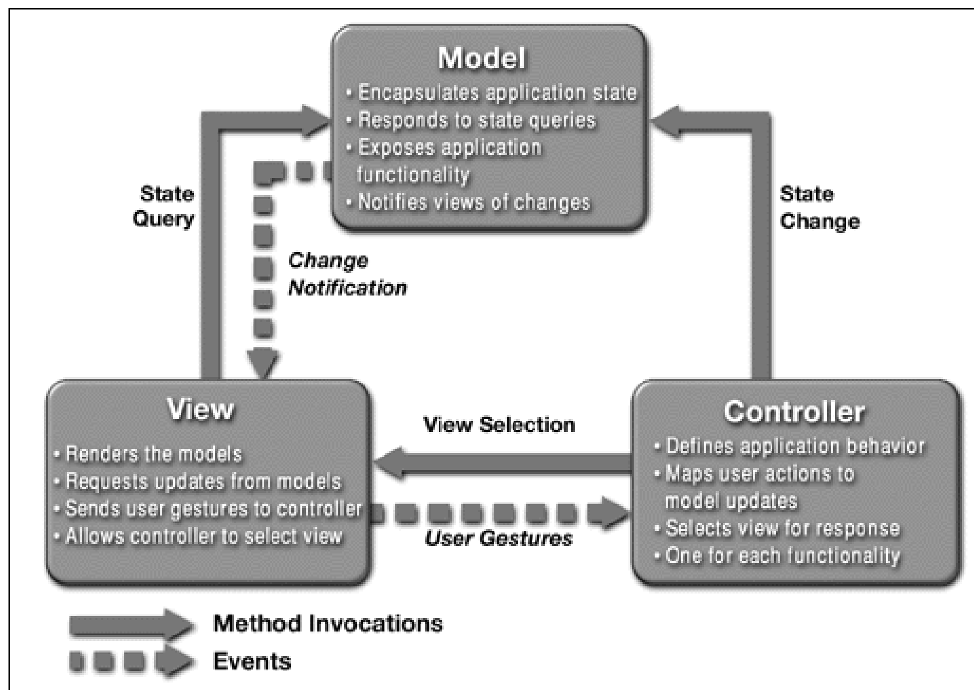


Figura 2.4 – Arquitetura MVC [4].

## 2.6 – Servlets e JSPs

Servlets são objetos que possibilitam, ao desenvolvedor, adicionar conteúdo dinamicamente em um servidor web através de uma plataforma Java. Geralmente o conteúdo gerado dinamicamente é exibido em um arquivo HTML, podendo ser também um XML ou um JSP. Na maioria das vezes, os servlets usam o protocolo de comunicação HTTP.

Todos os servlets herdam da classe GenericServlet. A HTTPServlet, uma servlet derivada da classe GenericServlet, possui métodos específicos do protocolo HTTP como o doPost e o doGet. Os métodos doPost e doGet servem para usar os métodos Post e Get do protocolo HTTP. Sob esta ótica, é importante ter ciência de que o Get deve apenas ser usado quando o estado do servidor não será alterado (ou seja, no caso de uma solicitação idempotente), e o Post nos casos restantes.

JSP é a tecnologia de desenvolvimento web server-side da plataforma Java, isto é, uma tecnologia que é executada no servidor e não no browser do cliente. Com JSP é possível fazer manipulações de variáveis, acesso a banco de dados, manipular arquivos, etc. Um arquivo JSP é um arquivo que contém código HTML e código Java sendo que este último deve ficar entre os símbolos <% e %>. Tudo que estiver entre esses

símbolos será executado no servidor e não aparecerá na página acessada pelo usuário final.

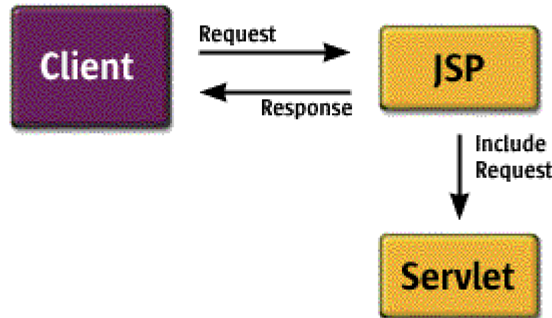


Figura 2.5 – Arquitetura web com JSP e Servlet [5].

A comunicação entre o cliente e o servidor web é realizada através do protocolo HTTP. Dois métodos HTTP fundamentais, no tocante à transferência de dados de formulários, são os métodos GET e POST. O GET, mais simples, tem como objetivo solicitar algum recurso do servidor e retornar o mesmo ao cliente. Já o POST pode solicitar algo, e ao mesmo tempo, enviar um bloco de dados ao servidor para ser processado. A figura 2.5 ilustra a dinâmica entre o cliente, o JSP e o Servlet. A interface com o cliente é o JSP e este através de POSTs e GETs se comunica com o Servlet. O Servlet se comunica com o JSP através de atributos.

A grande vantagem de usar Servlets e JSPs é o total isolamento entre a interface com o cliente e a lógica do sistema. Em uma aplicação de banco de dados, por exemplo, o JSP, interface com o usuário, faria suas solicitações ao Servlet. Este, por sua vez, verificaria as mesmas e, caso necessário, acessaria o banco de dados para obter as informações necessárias ao JSP. Ao concluir a obtenção de informações, o Servlet retornaria estas ao JSP e a interface com o usuário seria atualizada.

O fluxo de comunicação, mencionado no parágrafo anterior, deixa claro que, a forma com que a persistência de dados está implementada no servidor, é indiferente à camada do usuário, o JSP. Sob a ótica apresentada na figura 2.6, se esta persistência se basear em um sistema de gerenciamento de banco de dados ou um sistema de arquivos qualquer, a camada do usuário não será afetada. Se houver uma camada de classes DAO entre o Servlet e o SGBD, isto em nada afetará a camada do usuário. Analogamente,

quaisquer alterações na interface do sistema, o JSP, não afetarão a implementação da camada de dados.

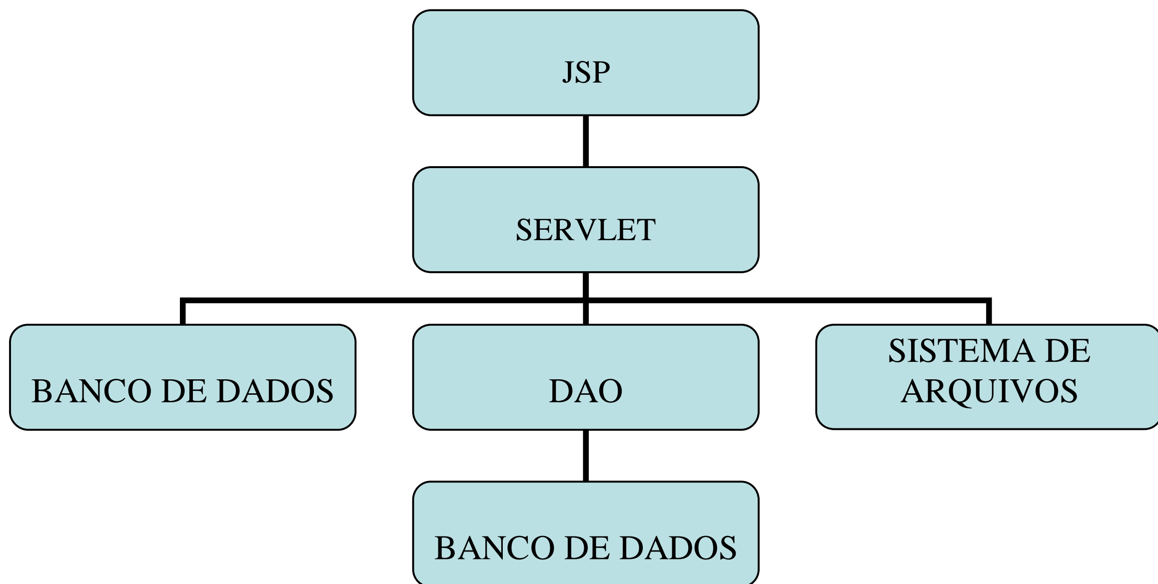


Figura 2.6 – Exemplo de aplicação com JSP e Servlet .

Com JSPs e Servlets, desenvolver aplicações web ficou mais simples. Desacoplando a interface da camada de negócio, tanto o desenvolvimento quanto a manutenção da aplicação ficam mais independentes um do outro, tornando o sistema mais modularizado.

## 2.7 – JDBC

JDBC ou Java Database Connectivity é uma API escrita em Java usada para acessar banco de dados. Através da JDBC é feita a conexão, consultas, inserções e atualizações no banco dados. A API JDBC é parte da plataforma de desenvolvimento Java. As classes da JDBC estão divididas nos pacotes java.sql e javax.sql. Ambos os pacotes estão incluídos nas plataformas Java SE(Standard Edition) e Java EE(Enterprise Edition).

Tipicamente, uma aplicação que acessa um banco de dados utilizando JDBC seguirá os seguintes passos: conectar com o banco de dados, executar um comando da linguagem SQL (podendo ser tanto um comando de manipulação de tabelas quanto um de manipulação de dados) e processar os resultados (se houver algum).

O estabelecimento da conexão pode ser feito de duas maneiras: através de um

DriverManager ou de um DataSource. O DriverManager define e gerencia as conexões com o banco de dados através dos drivers instalados. Cada banco de dados diferente requer um driver diferente instalado. Por outro lado, o DataSource permite que os detalhes do banco de dados seja transparente para a aplicação. Isso é feito através de um objeto que é configurado com as propriedades específicas do banco de dados. Cada banco utilizado teria o seu próprio DataSource.

A execução do comando SQL emprega o objeto Statement. Obtivemos esse objeto através da conexão criada anteriormente. O objeto Statement possui o método executeQuery(). Esse método recebe uma string com o comando a ser executado no banco de dados e retorna um objeto do tipo ResultSet.

O processamento dos resultados de uma consulta ao banco de dados utiliza o objeto ResultSet. Este objeto possui métodos para navegar entre os resultados da consulta. Como por exemplo, os métodos next(), previous(), first(), last(), etc. Para copiar o registro desejado, o objeto ResultSet possui métodos específicos para cada tipo de variável, como por exemplo, os métodos getString(), getFloat(), etc. A identificação da coluna que está sendo lida é efetuada através do nome da coluna ou do número da mesma. Seguindo o raciocínio anterior, teríamos: getString("NOMECOLUNA"), getFloat("NOMECOLUNA") ou getString(NUMEROCOLUNA), getFloat (NUMEROCOLUNA).

No exemplo abaixo podemos observar os três passos citados acima.

```
// Obtendo a conexão através do DriverManager
String url = "url do banco";
Connection con = DriverManager.getConnection(url, "usuario", "senha");
// Criando o Statement
Statement stmt = con.createStatement();
// Executando a consulta no banco e obtendo o ResultSet
ResultSet rs = stmt.executeQuery("SELECT Placa FROM Viaturas");
// Navegando pelo ResultSet
while (rs.next()) {
        // Extraindo os valores
        String placa = rs.getString("Placa");
        System.out.println(placa);
}
```

Exemplo de utilização da JDBC.

## 2.8 – JNDI

JNDI (*Java Naming and Directory Interface*) é uma API que permite que aplicações desenvolvidas na linguagem Java tenham acesso a serviços de nomes e diretórios. Serviços de nomes e diretórios são serviços que associam um nome e possivelmente sua localização em um diretório a um objeto, um endereço, um identificador, etc.

A JNDI pode ser utilizada com qualquer sistema de diretórios e nomes. Isso é possível devido ao uso de SPI (*Service Provider Interface*). Cada sistema diferente se comunica com a JNDI através de um SPI específico. As aplicações Java se comunicam com a API e esta se comunica com os serviços de nomes e diretórios através de SPIs.

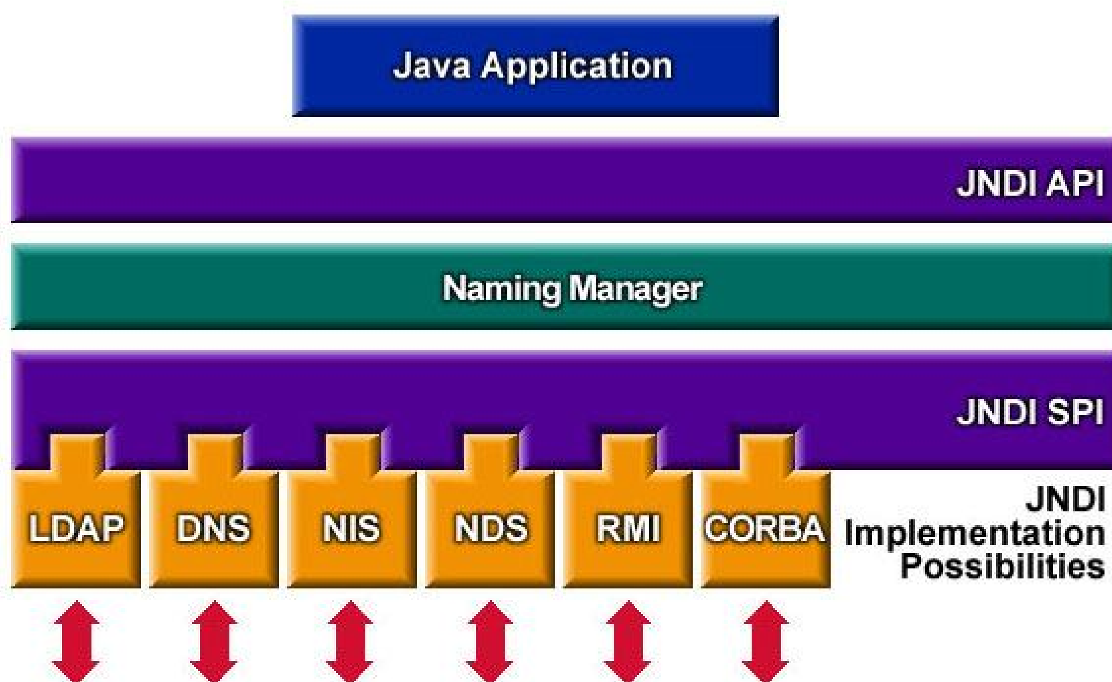


Figura 2.7 – Arquitetura JNDI[6].

Para poder usar a JNDI, é necessário ter as classes instaladas e o SPI referente ao serviço de nomes e diretórios desejado. A JNDI está incluída a partir da Java 2 SDK, v1.3. Por padrão, SPIs para os serviços LDAP (*Lightweight Directory Access Protocol*), CORBA (*Common Object Request Broker Architecture*) e RMI (*Java Remote Method Invocation*) já vem incluídas na SDK.

A JNDI vem nos seguintes pacotes:

- Javax.naming;

- Javax.naming.directory;
- Javax.naming.event;
- Javax.naming.ldap;;
- Javax.naming.spi.

# Capítulo 3

# Metodologia

## 3.1 – Arquitetura do projeto

O ponto central deste projeto é a aplicação Web denominada “Sistema de Gerenciamento de Frotas Empregando Informações Georeferenciadas”. Essa disponibiliza, através de uma página Web, diversas informações acerca das viaturas e chamados existentes. A aplicação é mostrada na figura 3.1.

Figura 3.1 – Página da aplicação Controle de Frotas

O principal componente dessa aplicação é um mapa de uma localidade, no caso da Cidade do Rio de Janeiro, gerado pela API Google Maps. Dispostos neste mapa, estão os símbolos correspondentes aos dois elementos monitorados: viaturas e chamados. Além de informar, ao usuário, o posicionamento desses elementos, a imagem de cada símbolo fornece o status do mesmo, como mostrado na figura 3.2. De modo a facilitar o entendimento, por parte do usuário, dos estados de cada elemento, esta legenda encontra-se no rodapé da página do “Controle de Frotas”.

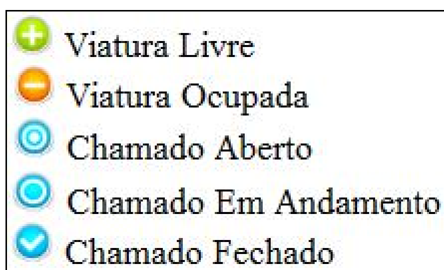

Figura 3.2 – Legenda

No canto superior direito da tela, encontra-se a lista de chamados, mostrada em detalhes na figura 3.3. Cada item da listagem possui o índice do chamado e o momento em que o mesmo foi originado. Caso este chamado esteja em andamento, sendo atendido por alguma viatura, o campo “Atendido por:” é concatenado à placa deste veículo e exibido logo abaixo do item.

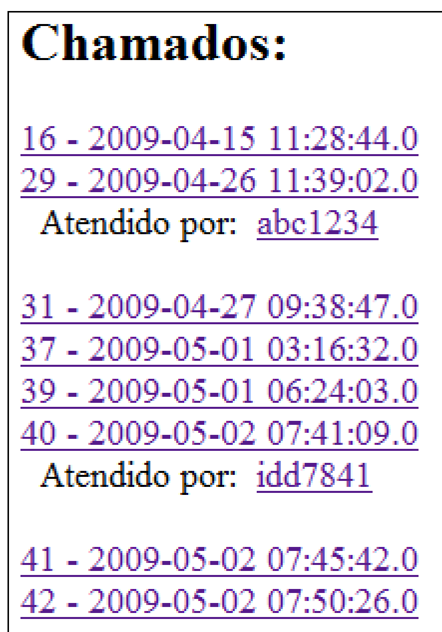

Figura 3.3 – Lista de chamados

Logo abaixo da lista de chamados, encontra-se a lista de viaturas, mostrada em detalhes na figura 3.4. Ela é mais simples que a sua correspondente de chamados. Cada item da lista corresponde à placa de uma viatura existente.

Figura 3.4 – Lista de viaturas

Os componentes mencionados anteriormente possuem diversas funcionalidades. Quando cada ícone, seja ele de chamado ou de viatura, é clicado, exibe ao usuário um balão de informação com alguns dados correspondentes, como é mostrado na figura 3.5.

Figura 3.5 – Informações sobre os símbolos: (a) viaturas; (b) chamados.

Outra funcionalidade está presente nas listagens. Quando o usuário clica em um dos itens, o mapa fica centralizado no ícone correspondente ao mesmo e, além disso, um balão com um mini-mapa mostra a imagem do mesmo com zoom. Este comportamento é mostrado na figura 3.6.

Figura 3.6 – Centralizando e dando um zoom no chamado 31

Os aspectos e funcionalidades mencionados precisam de uma fonte de dados que forneça todas as informações referentes a chamados e viaturas. Esta informação é fornecida através de um XML. Este, por sua vez, é atualizado por duas outras aplicações, uma Web, “Formulários” e outra Desktop, “GPSBot”. Basicamente, o usuário fornece as informações relativas a chamados e viaturas através da aplicação “Formulários”. A aplicação GPSBot gera novas coordenadas para as viaturas existentes, simulando a movimentação de uma frota real. Estas duas aplicações fornecem os dados mencionados à aplicação principal, “Controle de Frotas”, através de um XML. Um esquema bem resumido do sistema está ilustrado na figura 3.7. No decorrer deste item, explicaremos com mais detalhes o funcionamento de cada um desses componentes, assim como o relacionamento entre os mesmos.

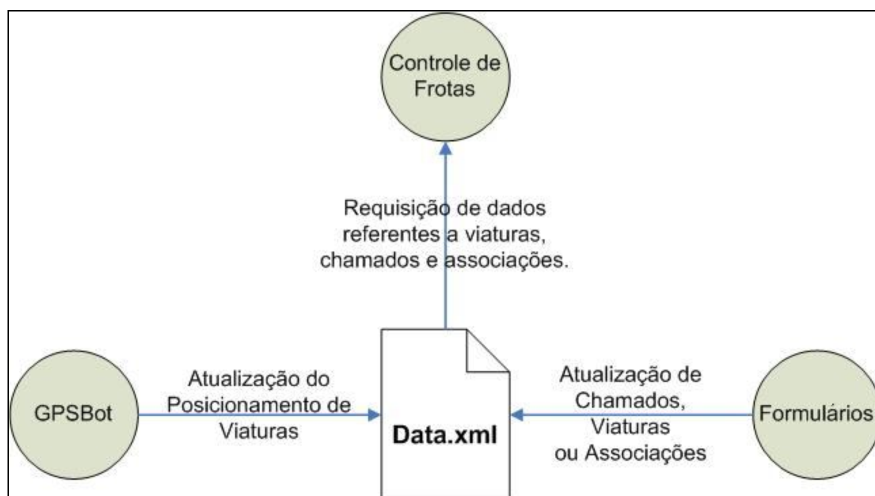


Figura 3.7– Esquema resumido do funcionamento do sistema

O propósito da aplicação “Formulários” é fornecer ao usuário uma forma de cadastrar as viaturas e chamados, assim como editar estes elementos. Composta por diversas páginas, sua estrutura se aproxima bastante daquela presente na modelagem de dados do sistema. A estrutura padrão de cada página é semelhante àquela apresentada na figura 3.8.

Figura 3.8 – Formulário de chamados

Utilizando como exemplo o formulário de chamados, presente na figura 3.8, podemos descrever a estrutura mencionada anteriormente. O menu, presente no lado esquerdo, está presente em todas as páginas e possui os links para as diversas páginas da aplicação. Na parte superior, encontra-se a vista de dados relacionados à modificação que está sendo realizada. No nosso exemplo, estão sendo exibidos dados dos chamados já cadastrados.

Na parte inferior, encontra-se a interface através da qual o usuário realiza as modificações desejadas. Quando existem campos de endereço, como no caso do formulário de chamados, a aplicação apresenta uma funcionalidade interessante, o campo “Endereços Sugeridos”. Este fornece sugestões de endereço, ao usuário, baseando-se nos dados “Logradouro”, “Número” e “CEP”, que o mesmo escreveu. Na ocasião do preenchimento dos campos “Logradouro”, “Número” ou “CEP”, por parte do usuário, é acionada uma função javascript. Esta concatena as informações existentes nesses três campos, utilizando esse dado como fonte para busca de endereços possíveis na base de dados da API do Goole Maps. A função mencionada retorna os endereços encontrados preenchendo o campo “Endereços Sugeridos” do form. Mediante clique na

sugestão desejada, a interface preenche automaticamente os campos de endereço, como mostrado na figura 3.9.

Endereço de Origem:

Endereços Sugeridos: R. Des. Omar Dutra - Taquara, Rio de Janeiro - RJ, 22715-440, Brasil

Logradouro: R. Des. Omar Dutra

Número:

Complemento:

CEP: 22715440

Bairro: Taquara

Figura 3.9 – Endereços sugeridos

Todas as modificações realizadas no aplicativo "Formulários" são armazenados no banco e, logo após, replicados no XML.

Dos três aplicativos que compõem o sistema de controle de frotas, um ainda não foi explicitado, o "GPSBot". Este tem o propósito de alterar o posicionamento das viaturas existentes, mediante a alteração de suas coordenadas, com o intuito de simular um sistema real de GPS. Esse componente de software está implementado de forma a poder ser substituído por um sistema de GPS, a qualquer momento, sem a necessidade de alterações nos demais aplicativos.

O "GPSBot" se constitui em um executável que fica rodando no servidor, gerando posições randômicas para cada viatura existente. A cada atualização, esse aplicativo procede da mesma forma que o "Formulários", após armazenamento dos novos dados no banco, replica as novas informações no XML. Esta peça de software foi desenvolvida com o intuito de emular a geração de georeferências por parte de um conjunto de GPSs instalados nas viaturas monitoradas. Elaborou-se este componente com o objetivo de sua substituição, por um sistema real de GPS, poder ocorrer sem quaisquer alterações nos demais componentes do sistema. Além disso, o "GPSBot" permitiu a realização dos diversos testes necessários ao sistema, sem a necessidade de compra e instalação dos referidos GPSs.

Descrevendo com mais detalhes a persistência dos dados, o XML comum às três aplicações é o "data.xml". O motivo pra fornecer as informações de chamados e viaturas através deste, e não do banco de dados, será explicitada a seguir, no item 3.4. O SGBD utilizado é o MySQL 5.1. Com a modelagem de dados descrita no item 3.2 e um conjunto de classes JDBC, armazenamos nele todos os dados necessários às três

aplicações. Basicamente, o sistema para controle de frotas está estruturado como ilustrado na figura 3.10.

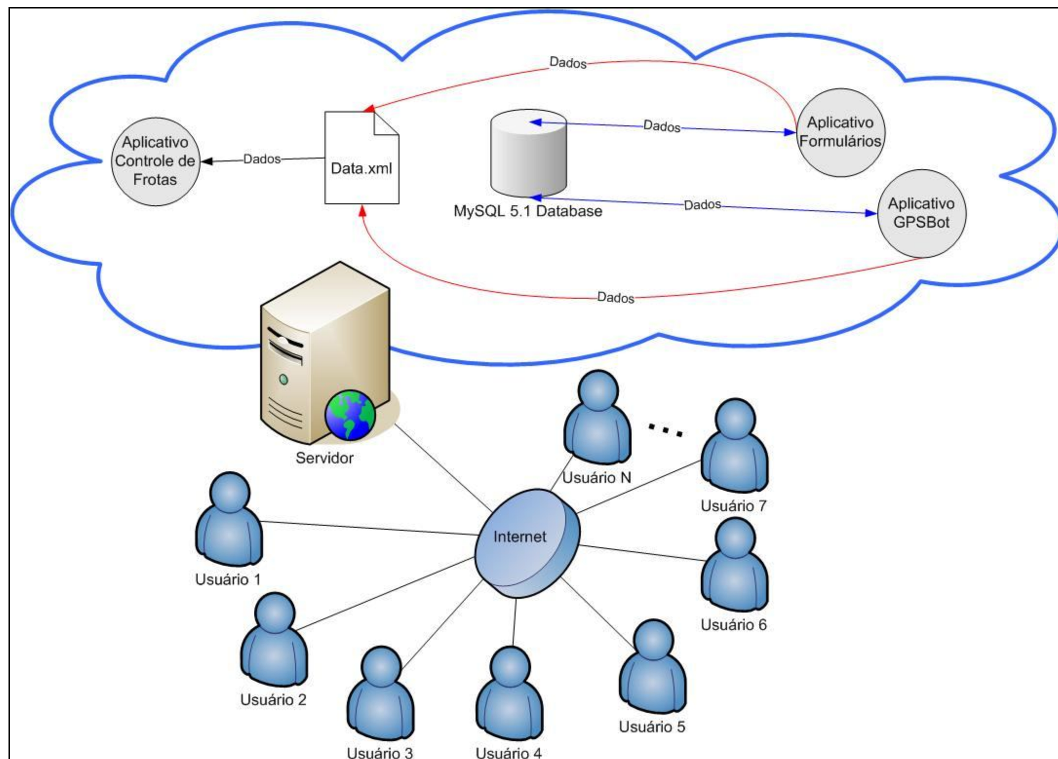


Figura 3.10 – Arquitetura do projeto

Durante a implementação das três aplicações procurou-se seguir o padrão MVC, com o intuito de isolarmos a camada de apresentação das camadas de manipulação e acesso a dados. Para as aplicações web, utilizamos as tecnologias de JSPs e Servlets que se aliam perfeitamente ao padrão MVC. Estas aplicações estão divididas da seguinte forma:

1) Classes Dao – são responsáveis pela realização de todas as transações necessárias com o banco de dados;

2) Classes Object – modelam os objetos necessários às aplicações;

3) Classes Controller – implementam os servlets que, por sua vez, controlam as views das aplicações;

4) Classes Business – realizam o intermédio entre as necessidades de dados das classes Controller e as classes Dao que realizam as transações com o banco. Caso ocorra alguma exceção durante as transações, estas classes registram essas exceções num arquivo de log para posteriores avaliações.

Analisando a aplicação GPSBot, temos uma estrutura semelhante às anteriores com as seguintes particularidades:

1) Classes Controller – implementam os métodos necessários à classe principal da aplicação;

2) Classes Connection – implementam as classes necessárias ao gerenciamento das conexões com o banco de dados.

Mais adiante, explicitaremos os diversos desafios conceituais e práticos enfrentados no decorrer da implementação deste sistema, explicando as soluções propostas e aquelas que se mostraram mais eficientes. Antes, no item 3.2, apresentaremos a modelagem de dados.

## 3.2 – Modelagem de dados

Durante a modelagem de dados buscamos conciliar a realidade do problema, que o nosso sistema pretendia solucionar, com as características das tecnologias que pretendíamos empregar. Sob esta ótica, apresentamos a modelagem mostrada na figura 3.11.

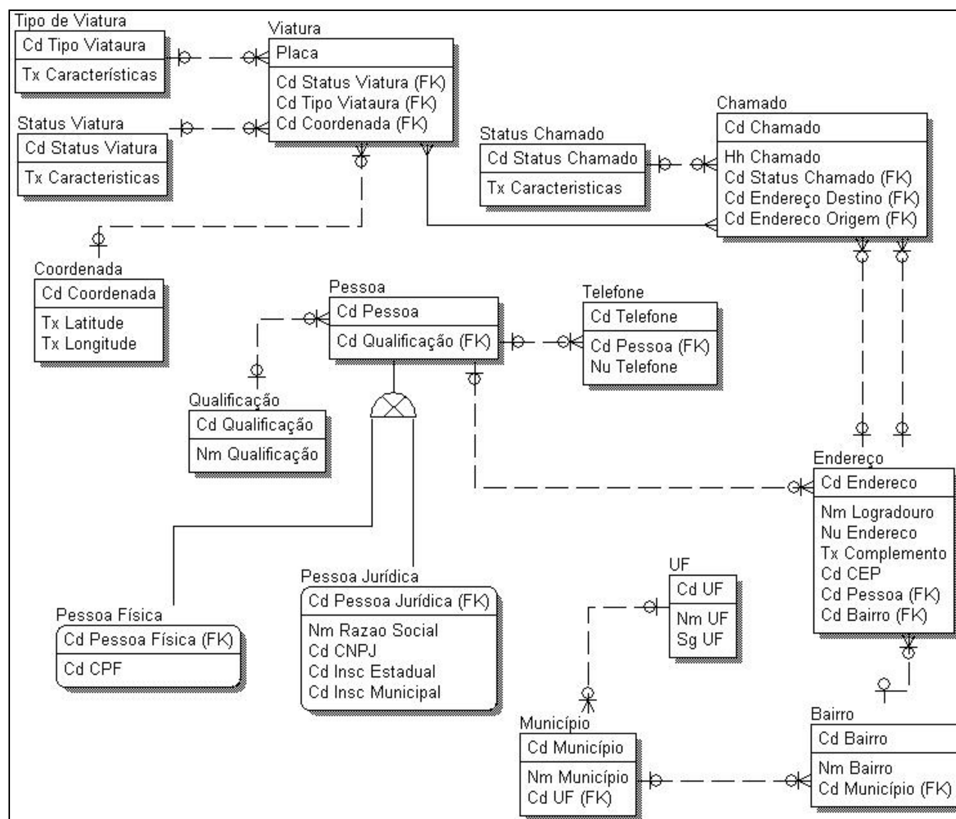


Figura 3.11 – Modelagem de dados

Como mostrado na figura 3.12, a entidade viatura possui sua placa como identificador, estando relacionada às entidades coordenada, tipo e status. Já a entidade chamado está relacionada às viaturas que o atendem assim como ao seus status, seu endereço de destino e seu endereço de origem.

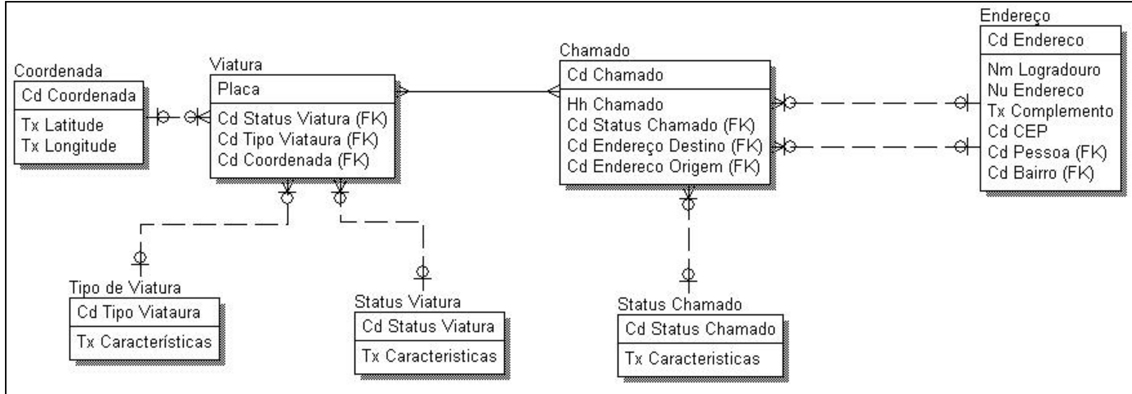


Figura 3.12 – Detalhes dos principais aspectos da modelagem

A entidade endereço além de possuir os atributos logradouro, número, complemento e CEP está relacionada a uma pessoa e a um bairro. Já a entidade telefone está relacionada única e exclusivamente à entidade pessoa. Estes detalhes são mostrados na figura 3.13.

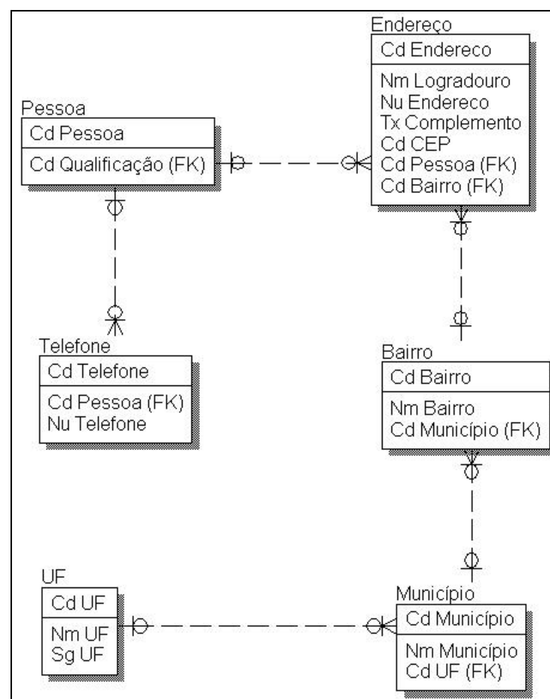


Figura 3.13 – Detalhes da modelagem do endereço

A entidade pessoa apresenta os subtipos pessoa jurídica e pessoa física, estando relacionada a estes exclusivamente. Além disso, está relacionada à entidade Qualificação. Estes detalhes são mostrados na Figura 3.14.

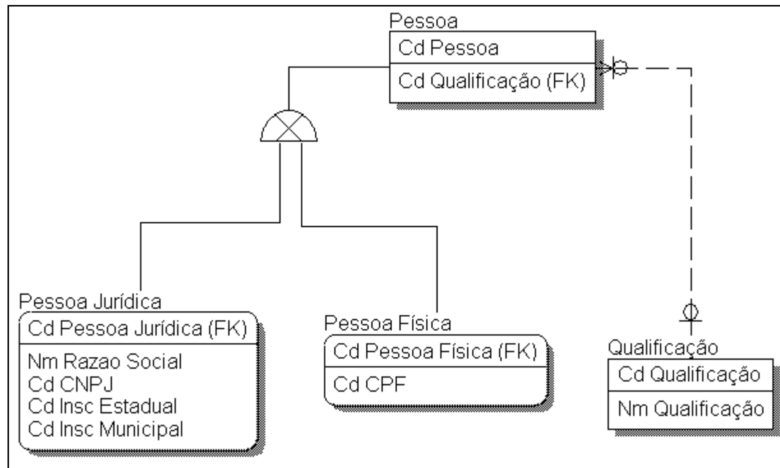


Figura 3.14 – Detalhes da modelagem de pessoa

## 3.3 – Gerenciamento de conexões ao SGBD

O SGBD utilizado foi o MySQL versão 5.1. Foram testadas diversas estratégias de integração entre o código Java e o SGBD até chegarmos àquela que melhor se aplicava ao sistema. Primeiramente, buscamos solicitar conexões diretamente das classes Dao, através de um conjunto de classes JDBC, como ilustrado na figura 3.15. Nossa opção foi a utilização da biblioteca MySQL Connector/J 5.1.

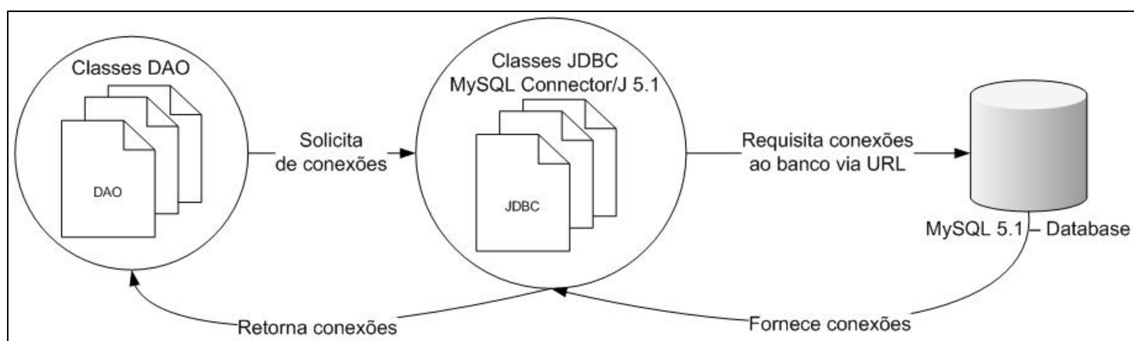


Figura 3.15 – Primeira alternativa de conexão ao SGBD

Apesar de a biblioteca ter funcionado perfeitamente, ainda tínhamos o desafio de otimizar as conexões ao banco. Por estarmos lidando com um sistema que se propõe a suportar vários usuários simultâneos, seria totalmente improdutivo gerarmos uma conexão a cada transação necessária. Daí surgiu a necessidade de elaborarmos um

sistema de gerenciamento das conexões ao banco. Este sistema teria o propósito de administrar estas, mediante a utilização de um “estoque” de conexões e uma administração deste. A figura 3.16 ilustra este gerenciamento.

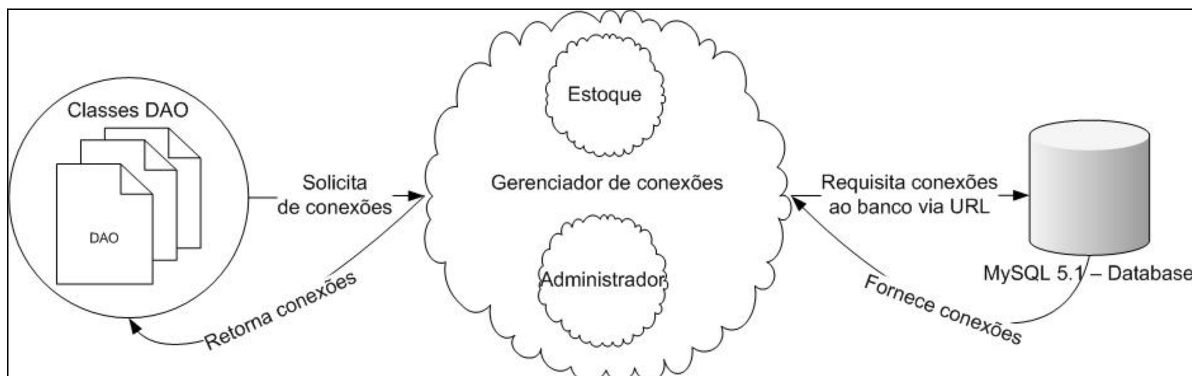


Figura 3.16 – Gerenciamento de conexões

A primeira estratégia foi a implementação de um pool de conexões via threads. A idéia era possuir um thread que gerasse conexões constantemente, com o intuito de manter um número fixo de conexões disponíveis. Quando algum método necessitasse de uma conexão, esse thread forneceria, criando imediatamente outra para substituir a utilizada. Atingimos a média de 9,4 segundos para as quinze primeiras conexões e 4,5 segundos para as vinte e cinco subseqüentes.

A segunda estratégia surgiu durante as pesquisas acerca da implementação da opção mencionada anteriormente. O Tomcat, gerenciador que escolhemos para administrar os servlets, possui uma funcionalidade que, através da tecnologia JNDI, administra as conexões a um banco de dados. A funcionalidade é totalmente transparente para o desenvolvedor, basta apenas que este configure este recurso no context.xml do Tomcat. Esta configuração consiste na especificação do número de conexões em espera, o driver de conexão utilizado, a string de conexão, o usuário e a senha do SGBD, o tempo que uma conexão deve perdurar após sua primeira utilização, dentre outros.

Esta estratégia se mostrou mais eficiente atingindo a média de 0,85 segundos para as quarenta primeiras conexões e 0,79 para as quarenta subseqüentes. Um comparativo simples encontra-se na tabela 3.1. Além de uma maior eficiência, passamos a possuir um pool de conexões, previamente estabelecidas de forma a antecipar a demanda, assim como uma administração eficiente deste recurso.

| Comparação de Conexões | | |
|---|---|---|
| | Pool via Contexto (s) | Pool via Threads (s) |
| Média Conexões 1 a 15 | 0,85 | 9,4 |
| Média Conexões 16 a 40 | 0,79 | 4,5 |

Tabela 3.1 – Comparação de Conexões

No caso da GPSBot, por termos um controle maior sobre o número de conexões e não possuirmos o Tomcat, utilizamos a estratégia de obtermos uma conexão cada vez que uma transação se fazia necessária.

## 3.4 – Gerenciamento das atualizações da aplicação principal

Para que o sistema atendesse seu propósito principal, otimizar o monitoramento e controle de frotas, havia a necessidade de que o mapa da aplicação principal fosse atualizado constantemente. Neste sentido, quando em produção, o sistema estará sujeito ao acesso de vários usuários simultâneos. Sendo assim, as requisições ao banco aumentam proporcionalmente a este número, dependendo da solução de atualização utilizada.

A solução imediata seria atualizar a página por meio de uma meta-tag de refresh que solicitaria ao navegador do cliente que atualizasse a página de tempos em tempos. Como mostrado na figura 3.17. Esta opção além de sobrecarregar o servidor, faria o mapa ficar piscando a cada atualização.

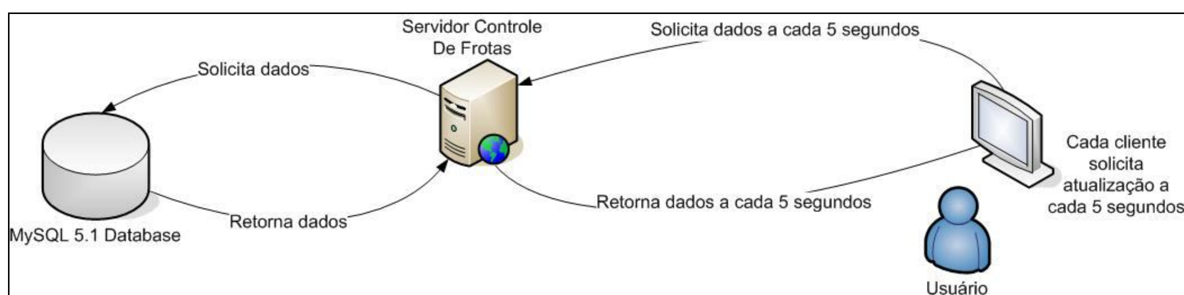


Figura 3.17 – Gerenciamento de atualizações, solução imediata

Para evitar as requisições subseqüentes ao banco foi criado um arquivo XML no servidor chamado refreshTime que continha a data da última atualização do banco. Do lado do servidor, toda vez que as aplicações Formulários ou GPSBot atualizassem o banco, as mesmas atualizariam esse arquivo com a data atual. Do lado do cliente, cada

página possuía um parâmetro com a data da última atualização e a meta-tag de refresh que solicitava atualização periodicamente. Porém, ao invés de recarregar todas as informações na ocasião de cada atualização os servlets passaram a fazer uma administração inteligente destes dados.

As listas de viaturas e chamados passaram a ser enviadas como atributos ao cliente e recuperadas do mesmo a cada atualização. Na ocasião de uma requisição, o servlet verificava o parâmetro (enviado pelo cliente) com a última atualização, se este fosse menor que aquele indicado no refreshTime (armazenado no servidor), o mesmo recarregava os dados do banco. Caso contrário, reenviava os dados anteriores. Assim, eliminou-se o acesso desnecessário ao banco, como mostrado na figura 3.18.

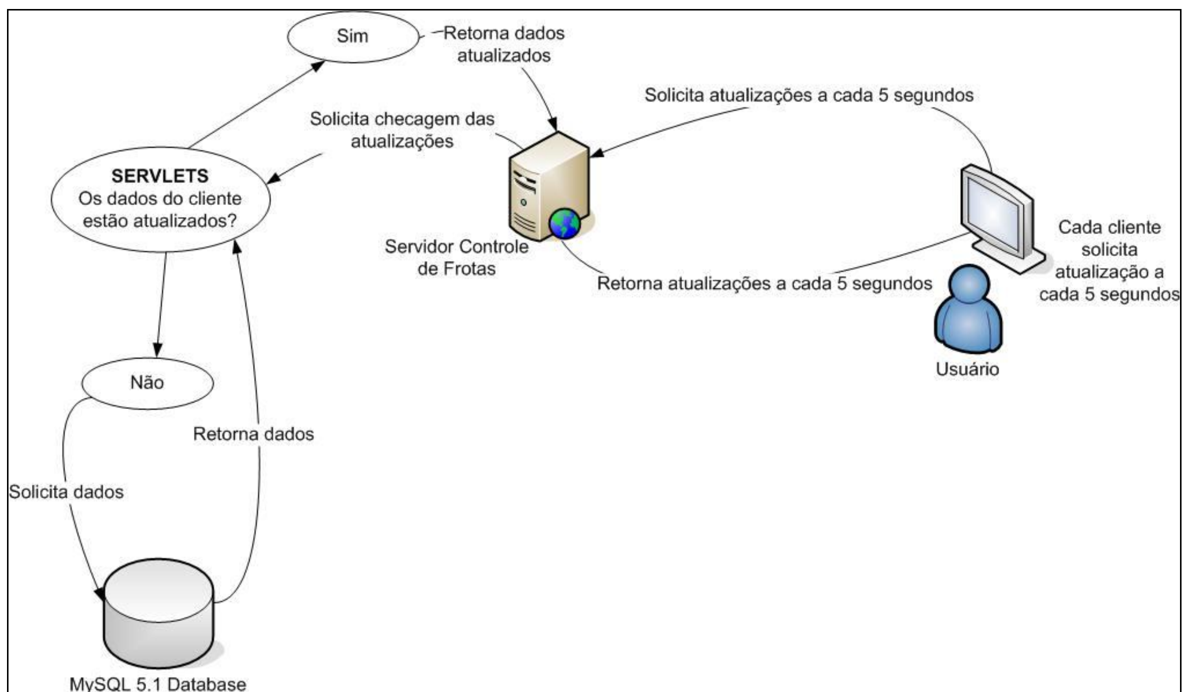


Figura 3.18 – Gerenciamento de atualizações, segunda alternativa

Porém essa solução apresentava dois problemas:

1) O mapa continuava a piscar a cada atualização, pois esta fazia com que a API do Google recarregasse os mapas; dentre outras coisas, essa recarga evitava que o usuário percebesse a movimentação das viaturas;

2) Para cada n usuários, uma atualização do banco gerava n novas requisições; logo, o número de requisições ainda era alto.

Para combater estes intempéries, recorreu-se a tecnologia javascript. Toda a API do Google Maps está estruturada nesta linguagem. Cada marcador, ícone que identifica

uma viatura ou chamado no mapa, corresponde a um objeto. A nova estratégia consistiu em atualizar apenas estes objetos, deixando o restante da página sem atualização. Esta atualização passou a ser alimentada por um XML geral denominado data.xml. Do lado do servidor, cada atualização passou a espelhar os dados do banco neste XML. O esquema do gerenciamento mencionado é mostrado na figura 3.19. Como duas aplicações (Formulários e GPSBot) tentarão acessar este arquivo intermitentemente, surgiu o problema de concorrência de acesso ao mesmo. A solução para este empecilho será descrita no item a seguir.

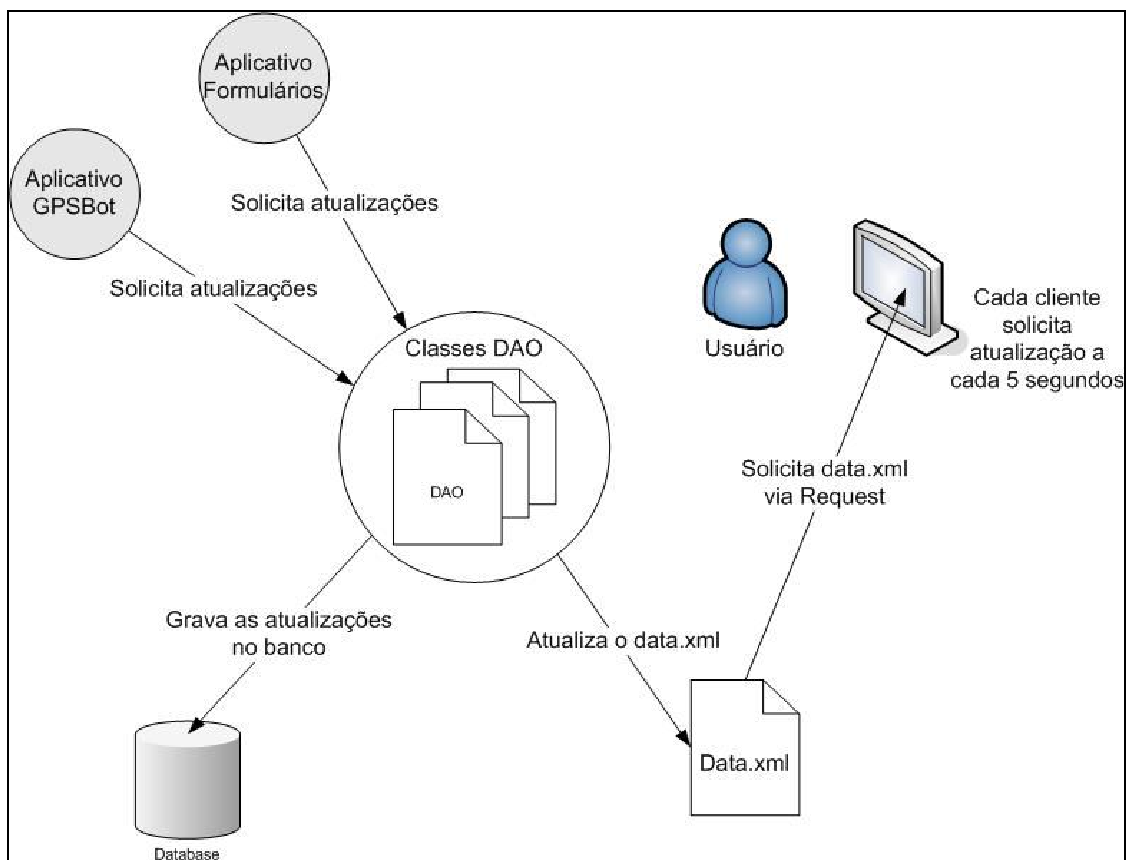


Figura 3.19 – Gerenciamento de atualizações, solução mais eficiente

A necessidade de envio de atributos a cada requisição, por parte dos servlets, desapareceu. Isso, devido à possibilidade de acesso ao data.xml através de uma requisição do tipo GET por parte do cliente. Do lado deste, o javascript passou a requisitar o data.xml do servidor a cada 5 segundos e parsear o mesmo alimentando os objetos de viaturas e chamados. Este processo passou a ser controlado pelo método javascript window.setInterval() que também dispara a atualização dos marcadores.

Com esta solução, a atualização passou a ser imperceptível ao usuário e o processo pôde ser aproveitado para a atualização das listas de viaturas e chamados. Para evitar o cache do data.xml por parte do navegador, a url de solicitação do mesmo passou a possuir um parâmetro que possuía a data e hora, em milissegundos, do momento da solicitação. Assim, o navegador passou a encarar cada solicitação como um novo recurso.

## 3.5 – Concorrência de acesso ao data.xml

Como mencionado anteriormente, duas aplicações precisam acessar o data.xml para atualização, a Formulários e a GPSBot. Esta necessidade exige o controle de concorrência em relação às tentativas de acesso a este arquivo por parte das aplicações.

Primeiramente, criamos um arquivo XML denominado flag, que possuía '0' quando o arquivo data.xml estava livre, ou '1' quando o mesmo estava sendo utilizado por outro processo. Esta solução era ineficaz, pois, dessa forma, transmitimos o problema de concorrência ao flag.xml.

A solução vigente se baseou na anterior. Criou-se uma proteção via algoritmo que só permite o acesso ao data.xml, caso os processos de acesso e modificação do flag.xml obtenham sucesso. Existe um thread que é disparado a cada atualização do banco. Este thread recebe em sua construção as listas de viaturas e chamados, assim como o momento da atualização em milissegundos, sendo iniciado.

Enquanto não obtém sucesso, este thread fica rodando indefinidamente. Para obter sucesso, o mesmo deve, na seguinte seqüência:

1) conseguir acessar o flag.xml e modificar seu status para ocupado, impedindo o acesso ao data.xml por parte de qualquer outro thread;

2) verificar se a última data de atualização do XML é anterior à data dos dados que ele pretende gravar no arquivo;

3) Caso os dados sejam mais atuais que aqueles armazenados no XML, procede com a atualização do data.mxl (chamados, viaturas e data de atualização); caso contrário, dispensa a atualização e realiza a próxima etapa;

4) conseguir acessar o flag.xml e modificar seu status para livre, liberando o acesso ao data.xml por parte de qualquer outro thread.

O processo mencionado é mostrado no diagrama de processo da figura 3.20.

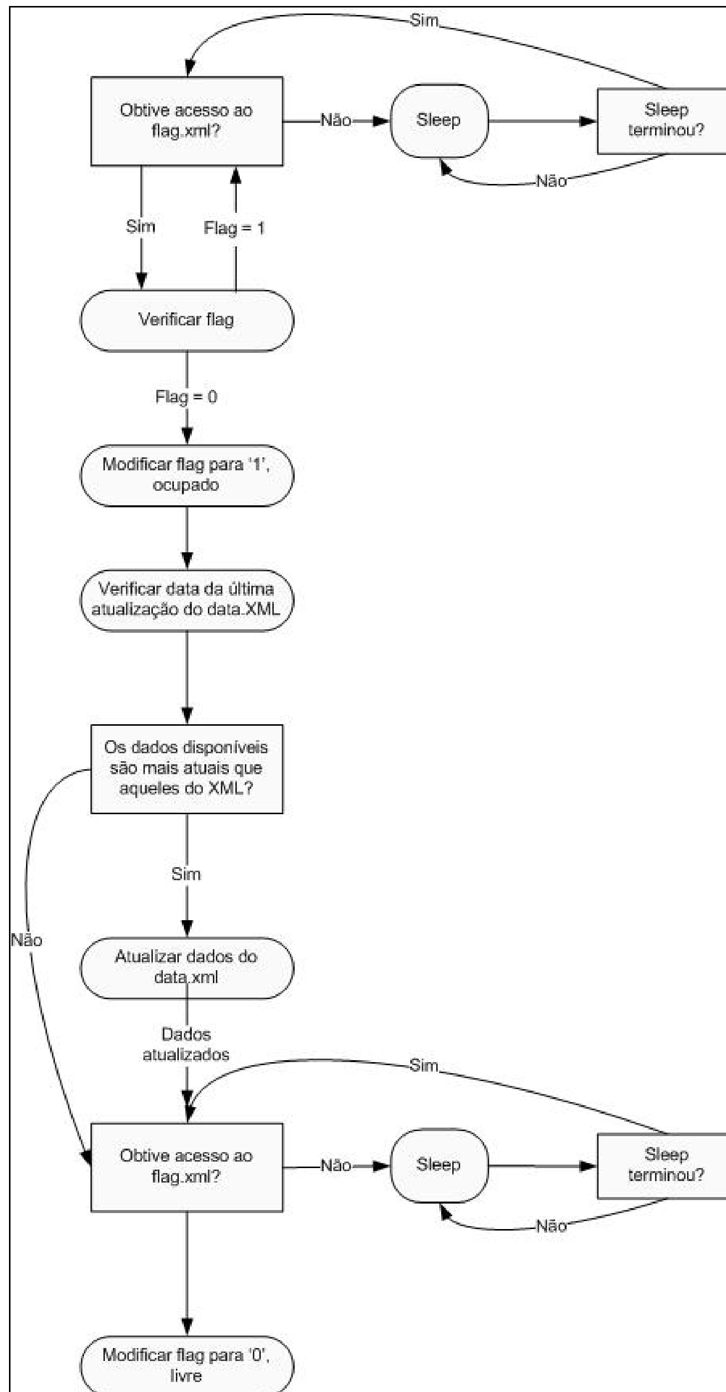


Figura 3.20 – Gerenciamento da concorrência de acesso ao data.xml

Para evitar consumo excessivo de CPU, nas ocasiões em que determinado thread não obtém sucesso, o thread de atualização fica em sleep por 200 milissegundos. Nas ocasiões em que ocorrer uma tentativa de atualização, por parte de um thread, após seu sleep, e os dados do mesmo estiverem desatualizados em relação ao XML, a verificação da data de atualização, mencionada nos passos acima, impedirá a atualização errônea do arquivo.

## 3.6 – Levantamento de custos

A escolha pela API do Google Maps reduziu sensivelmente o custo do projeto uma vez que as alternativas disponíveis no mercado são extremamente custosas. Algumas destas alternativas possuem licenças de uso, e outras, apesar de gratuitas demandariam muito desenvolvimento para integração. Sob esta ótica, as alternativas que serão apresentadas a seguir são as possíveis de serem implementadas em médio prazo, ou seja, não necessitariam de uma construção de mapas, mas sim uma integração com mapas já existentes. Na tabela abaixo, estão relacionadas as alternativas à API do Google Maps e seus respectivos preços.

| Produto | Licença |
|---|---|
| ESRI ArcGIS | $1500,00 por ano |
| Terraview | General Public License |
| TrackSource | General Public License |

Tabela 3.2 – Custo das alternativas á API do Google Maps.

A primeira alternativa é o produto ESRI Developer Network, desenvolvido pela empresa ESRI, que é uma empresa com soluções de sistemas de informações geográficas. Este produto é um kit de desenvolvimento de software para integração com o ArcGIS, o principal produto da ESRI.

O ArcGIS não provê o banco de dados com os mapas, portanto para substituir a API do Google Maps precisaríamos também dos mapas vetoriais e das imagens de satélites. Dentre outras funcionalidades, o sistema da ESRI sobrepõe as imagens de satélite com o mapa vetorial. Uma licença custa $1500,00 por ano e permite acesso à exemplos de aplicações integradas com o ArcGis, blogs e toda a documentação online [11].

As imagens de satélites podem ser fornecidas com diversas resoluções, sendo, geralmente, vendidas por quilômetro quadrado. O preço depende de alguns fatores como a data que o levantamento foi feito e a quantidade solicitada, custando em torno

de R$100,00 o quilômetro quadrado. Essas imagens podem ser obtidas nos sites dos satélites, como Landsat, CBERS e SPOT. A área do município do Rio de Janeiro é de 1.205,8 Km2 [13], portanto, seriam gastos com imagens de satélite aproximadamente 1.205,8 Km2 multiplicado por R$100,00/Km2, que resulta em R$120.580,00. O mapa vetorial do Município do Rio de Janeiro pode ser comprado através do Instituto Pereira Passos. No contexto deste projeto, as camadas de interesse são as camadas de transporte e localidades, totalizando um valor de R$2.400,00. Sob a perspectiva do desenvolvimento, se a alternativa escolhida fosse o ArcGis por exemplo, além do valor da licença e dos mapas, seriam gastos aproximadamente 300 homem hora a um custo aproximado de R$90,00 cada, totalizando um custo de R$27.000,00. No caso da alternativa do Terraview, o custo é similar, entretanto deveriam ser computadas apenas as despesas de desenvolvimento. Somando os custos de licença e homem hora com o custo das imagens de satélite e dos mapas vetoriais, teremos um total de R$152.980,00.

A terceira alternativa é o TrackSource, que é um projeto de mapas para o GPS Garmin. Este projeto é desenvolvido e atualizado pelos próprios usuários. Os mapas são distribuídos gratuitamente [12]. A utilização dos mapas do projeto TrackSource fora do GPS Garmin é possível pois todos os códigos são abertos e livres para quaisquer modificações, mas exigiria muito desenvolvimento. Para utilizar os mapas do projeto TrackSource seriam necessárias aproximadamente 500 homem hora a um custo de R$90,00 cada para realizar toda a integração necessária, totalizando R$45.000,00.

Podemos concluir que utilização da API do Google Maps é a alternativa mais econômica disponível. O desenvolvimento de aplicativos utilizando a API do Google Maps é bastante facilitado e incentivado. Toda a documentação necessária, incluindo exemplos, estão disponíveis ao público. Além disso, a sua utilização é gratuita.

# Capítulo 4

# Conclusão

## 4.1 – Conclusão

O desenvolvimento deste projeto provou que um sistema de gerenciamento de frotas, empregando informações georeferenciadas, pode ser implementado a um custo baixo, residindo os maiores custos nos sistemas de GPS a serem instalados na viatura.

Tecnologias de código aberto como a linguagem Java e o SGBD MySQL se mostraram eficazes ao desenvolvimento deste sistema complexo, viabilizando a solução de problemas que surgiram desde a manipulação de dados até a exibição destes aos usuários em tempo real. Além disso, as opções tecnológicas mencionadas garantem ao sistema portabilidade, no tocante a sistemas operacionais. Esta propriedade garante ao projeto vantagem estratégica no ambiente comercial.

A arquitetura web tornou o projeto escalável, isto é, sem quaisquer alterações no software desenvolvido podemos ampliar o número de usuários atendidos pelo sistema. Apenas eventuais investimentos na infra-estrutura do servidor viriam a ser necessários. Estes teriam o intuito de aumentar a capacidade conexões simultâneas ao mesmo, assim como a capacidade de processamento necessária ao processamento dos Servlets e o acesso crescente ao banco de dados. Enfim, ao final do projeto, se obteve um sistema de baixo custo, portável e com uma excelente capacidade de expansão.

Para futuras melhorias do sistema, sugerimos que o mecanismo de controle de concorrência ao data.xml seja melhorado. O atual é eficiente, pois impede o surgimento de um conjunto de dados inconsistente, porém, ainda não é eficaz para atender a um número grande de atualizações simultâneas. Nestas ocasiões, o mesmo gera um conjunto de atualizações em aguardo e, se a atualização da vez for mais antiga que a última realizada, ela não é efetuada, liberando o data.xml. Caso o sistema de concorrência, ou até mesmo o sistema de atualização de dados, permitissem uma atualização mais rápida, nenhuma informação seria dispensada.

# Bibliografia


[1] BASHAM, B., SIERRA, K., BATES, B. , Use a cabeça – Servlets e JSPs. Rio de Janeiro, Alta Books, 2007.

[2] JANDAL JUNIOR, P. J., Java – Guia do Programador. São Paulo, Novatec, 2007.

[3] GOOGLE CODE, Documentação da API do Google Maps. Disponível em: http://code.google.com/intl/pt-BR/apis/maps/. Acesso em: 20 de Maio de 2009.

[4] ECKSTEIN, ROBERT, Java SE Application Design With MVC. Disponível em: http://java.sun.com/developer/technicalArticles/javase/mvc/. Acesso em: 20 de Maio de 2009.

[5] SUN MICROSYSTEMS, JavaServer Pages[tm] Technology – Powering the Web Experience with Dynamic Content. Disponível em: http://java.sun.com/products/jsp/jsp_servlet.html. Acesso em: 20 de Maio de 2009.

[6] SUN MICROSYSTEMS, JNDI Overview. Disponível em: http://java.sun.com/products/jndi/tutorial/getStarted/overview/index.html. Acesso em: 20 de Maio de 2009.

[7] OBERRIGHT, JOHN E. Satellite, Artificial. World Book Online Reference Center. 2004. World Book, Inc. http://www.worldbookonline.com/wb/Article?id=ar492220.

[8] BETKE, KLAUS, The NMEA 0183 Protocol. Disponível em: http://www.tronico.fi/OH6NT/docs/NMEA0183.pdf. Acesso em: 21 de Maio de 2009.

[9] PFEIFER, HEINRICH, A long term fix with GPS and GARtrip. Disponível em: http://www.gartrip.de/long.htm#p6. Acesso em: 21 de Maio de 2009.

[10] EUROPEAN SPACE AGENT, What is EGNOS?. Disponível em: http://www.esa.int/esaNA/GGG63950NDC_egnos_0.html. Acesso em: 21 de Maio de 2009.

[11] ESRI, *ESRI Network Developer*. Disponível em: http://www.esri.com/software/arcgis/edn. Acesso em: 29 de Maio de 2009.

[12] TRACKSOURCE, *Projeto TrackSource*. Disponível em: http://www.tracksource.org.br. Acesso em: 29 de Maio de 2009.

[13] PORTAL DO CIDADÃO, Dados do Município do Rio de Janeiro. Disponível em: http://www.governo.rj.gov.br/municipal.asp?M=85. Acesso em: 2 de Junho de 2009.