HENRIQUE RIBEIRO REZENDE

ESTUDO E DESENVOLVIMENTO DE UM SISTEMA DE COLETA, ANÁLISE E VISUALIZAÇÃO DE DADOS GEOREFERENCIAIS APLICADOS AO SETOR DE TRANSPORTE PÚBLICO: MÓDULO GERENTE E MÓDULO WEB

LAVRAS - MG

2010

HENRIQUE RIBEIRO REZENDE

ESTUDO E DESENVOLVIMENTO DE UM SISTEMA DE COLETA, ANÁLISE E VISUALIZAÇÃO DE DADOS GEOREFERENCIAIS APLICADOS AO SETOR DE TRANSPORTE PÚBLICO: MÓDULO GERENTE E MÓDULO WEB

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientador:

Prof. Dr. Ahmed Ali Abdalla Esmin

LAVRAS - MG

2010

HENRIQUE RIBEIRO REZENDE

ESTUDO E DESENVOLVIMENTO DE UM SISTEMA DE COLETA, ANÁLISE E VISUALIZAÇÃO DE DADOS GEOREFERENCIAIS APLICADOS AO SETOR DE TRANSPORTE PÚBLICO: MÓDULO GERENTE E MÓDULO WEB

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

APROVADA em ____ de _____ de _____

_____

Drª. Marluce Rodrigues Pereira – UFLA

_____

Dr. Denilson Alves Pereira – UFLA

_____

Prof. Dr. Ahmed Ali Abdalla Esmin
Orientador

LAVRAS - MG

2010

A meus pais, Hilton e Lena,

A meu irmão e ídolo, Rafael

A minha eterna companheira, Fernanda

Dedico.

RESUMO

Um sistema de transporte público pode influenciar na qualidade de vida e até mesmo no desenvolvimento de uma cidade. Um dos maiores motivos para as pessoas abandonarem o transporte público para terem seus próprios veículos é a falta de informações e má qualidade do serviço oferecido pelas empresas. Toda essa sequência de eventos pode levar ao caos do trânsito nas grandes cidades e novamente contribuir para uma baixa qualidade de vida. O projeto e desenvolvimento de um sistema que vise sanar um pouco dessas deficiências encontradas atualmente no setor de transporte é importante. Este trabalho tem como objetivo planejar de um sistema capaz de lidar com informações geográficas de maneira simples e eficiente, com o objetivo de exibir informações relevantes aos usuários finais, como localização de veículos em tempo real, rotas, pontos de paradas, previsão de embarque, etc. Porém, por ser um sistema complexo, há também a necessidade de lidar com certos problemas como o suporte a acessos simultâneos em grande escala, além de critérios que todo sistema deve levar em consideração, como modularidade, manutenibilidade, portabilidade, entre outros. O foco deste trabalho está no projeto e desenvolvimento do Módulo Web e principalmente do Módulo Gerente.

Palavras-chave: sistemas de transporte inteligentes, GPS, banco de dados geográficos, mapeamento objeto-relacional, sistemas web, GIS, sistema de rastreamento.

ABSTRACT

A public transportation system can influence the quality of life and even the development of a city. One of the biggest reasons that make people leave the public transport to have their own vehicles is the lack of information and poor quality of service offered by companies. This entire sequence of events can lead to traffic chaos in major cities and again contribute to a low quality of life. The design and development a system that devise to redress some of these shortcomings currently found in the transportation sector is important. This work have as objective, plan a system capable of dealing with geographic information easily and efficiently, in order to display relevant information to end users, such as vehicle location in real time, routes, way stations, provision of boarding, etc. However, being a complex system, there is also the need to deal with certain issues such as support for simultaneous access in large scale, and some criteria that any system should take into consideration, such as modularity, maintainability, portability, among others. The focus of this work is on the design and development of a Web Module and mainly on a Manager Module.

Keywords: intelligent transportation systems, GPS, geographic databases, object-relational modeling, web systems, GIS, tracking system.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ACID	*Atomicidade, Consistência, Isolamento e Durabilidade*
AJAX	*Asynchronous JavaScript and XML*
APTS	*Advanced Public Transportation Systems*
ATIS	*Advanced Traveler Information Systems*
ATMS	*Advanced Transportation Management Systems*
AVL	*Automatic Vehicle Location*
CGI	*Common Gateway Interface*
EATS	*Events Activated Tracking System*
EJB	*Entity Java Beans*
ETC	*Electronic Toll Collection*
GEOS	*Geometry Engine – Open Source*
GIS	*Geographic Information System*
GLONASS	*Global Navigation Satellite System*
GNSS	*Global Navigation Satellite Systems*
GPRS	*General Packet Radio Service*
GPS	*Global Positioning System*
ITS	*Intelligent Transportation Systems*
JPA	*Java Persistence API*
JPQL	*Java Persistence Query Language*
MER	Modelo Entidade-Relacionamento
NAVSTAR-GPS	*Navigation System Using Time and Ranging* / *Global Positioning System*
OGC	*Open Geospatial Consortium*
OO	Orientado a Objetos
ORM	*Object-Relational Mapping*
POJO	*Plain Old Java Object*

SGBD	Sistema Gerenciador de Banco de Dados
SFS	*Simple Features Specification for SQL*
SQL	*Structured Query Language*
SRID	*Spatial Referencing System Identifier*
SVD	*Selective Vehicle Detectors*
V2V	*Vehicle-to-Vehicle*
V2V	*Vehicle-to-Vehicle Integration*
VII	*Vehicle-to-Infrastructure Integration*
VII	*Vehicle-to-Infrastructure Integration*
VMT	*Vehicle Miles Traveled*
WKB	*Well-Know Binary*
WKT	*Well-Know Text*
XML	*Extensible Markup Language*

SUMÁRIO

1. INTRODUÇÃO

1.1. Visão Geral

O transporte público exerce um papel fundamental para o desenvolvimento e qualidade de vida de uma cidade. À medida que uma cidade se desenvolve, a necessidade por deslocamentos aumenta. Pessoas precisam percorrer longas distâncias por motivos variados como trabalho, lazer ou estudos. Vários motivos como, status, divulgação da mídia ou até mesmo facilidade faz com que pessoas deixem de utilizar o transporte público para recorrerem a automóveis individuais, causando grandes danos ao ambiente, a circulação do trânsito, e consequentemente à cidade como um todo.

Sempre se deve pensar em meios de aprimorar o transporte público de maneira a convencer as pessoas que sua utilização causará menos danos às cidades. Um dos fatores que mais influenciam na qualidade de serviço do transporte público é a tecnologia empregada nos veículos, visando o conforto e facilidade para a vida do usuário (Travassos, 2000, citado por Azambuja 2002).

Para isso, foi proposto o projeto e desenvolvimento um sistema cujo objetivo geral é facilitar a vida do usuário e também das empresas de transporte público. A idéia principal consiste em um sistema AVL (*Automatic Vehicle Location*) (Amarante, 2007), que coletará as posições GPS (*Global Positioning System*) (Dana, 2010) dos ônibus cadastrados no sistema e utilizará esses dados para vários fins, explicados mais adiante nos módulos.

Este sistema está dividido em cinco módulos: Módulo Coletor, Módulo Gerente, Módulo Administrador, Módulo Web e Módulo Móvel. Alguns desses módulos estão distribuídos em outros dois trabalhos, Rocha (2010) e Castro (2010), e foram projetados e desenvolvidos em paralelo com os módulos

abordados nesse trabalho. Sua arquitetura, partindo de uma visão geral, pode ser vista na Figura 1.1.

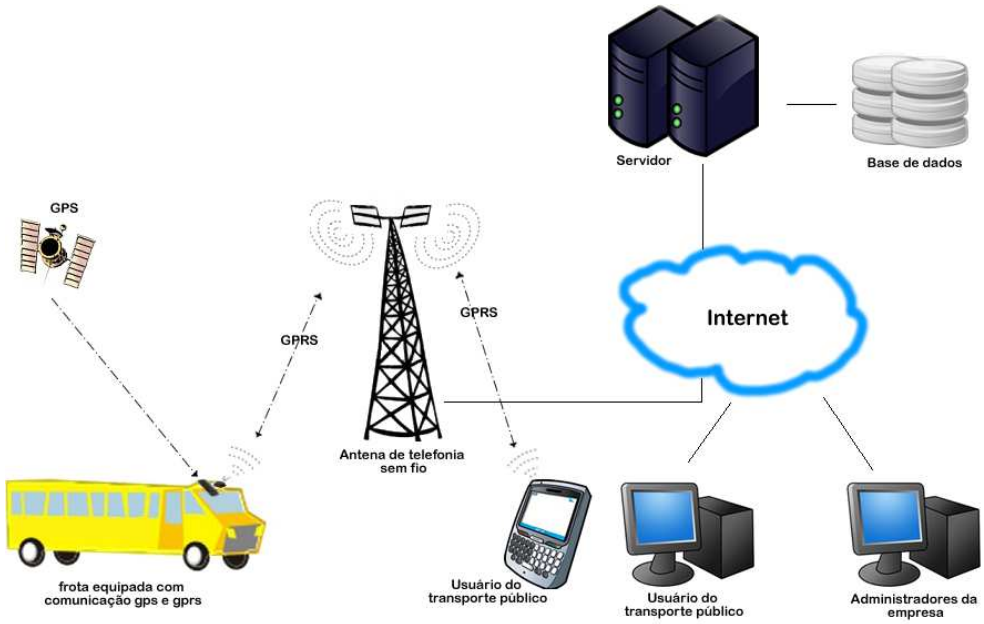

Figura 1.1: Arquitetura Geral do Sistema. Fonte: Rocha (2010).

1.2. Módulo Coletor

O Módulo Coletor é responsável por obter os dados GPS (*Global Positioning System*) e armazená-los em um servidor com banco de dados para análise em tempo real. Para isso, cada ônibus possuirá um módulo GPS, para captar a posição geográfica atual do ônibus via satélite, e um módulo GPRS (*General Packet Radio Service*), que enviará esses dados para o servidor utilizando os serviços da telefonia móvel.

Este módulo foi projetado e desenvolvido por Castro (2010) e seu foco maior está no desenvolvimento de um módulo de hardware com os objetivos citados anteriormente.

1.3. Módulo Gerente

O Módulo Gerente é responsável por toda a lógica de negócio do sistema. O maior fluxo de informações ocorre neste módulo, pois todos os outros módulos estão conectados a ele e só se comunicam através dele. Suas funções gerais incluem o acesso ao banco de dados, de modo a atender todas as necessidades de um banco de dados geográfico, à realização de toda regra de negócio que envolve o sistema no geral, e tradução de todas as informações coletadas no Módulo Coletor para os módulos que irão usufruir de tais dados. Por ser considerado o núcleo de todo o sistema, seu projeto e desenvolvimento foi elaborado de maneira a se manter o mais Orientado a Objetos possível, tornando-o assim manutenível, de mais fácil entendimento e modularizado.

1.4. Módulo Web e Módulo Administrador

O Módulo Web e Módulo Administrador foram colocados juntos devido às suas semelhanças, principalmente por estarem voltados a sistemas Web. Porém, sua diferença básica está no público-alvo, sendo que, enquanto o Módulo Administrador está voltado para os administradores do sistema, o Módulo Web está mais voltado para o usuário final, pois é através dele que os usuários irão usufruir de funções relativas à visualização, como por exemplo, posição atual de cada ônibus da empresa, suas rotas e tempo necessário para o ônibus chegar ao próximo ponto de parada. Os administradores podem visualizar tudo o que um usuário comum pode visualizar, e ainda contar com outras funções relativas ao gerenciamento da empresa, por exemplo, quantidade de quilômetros percorridos, dados do motorista e do ônibus e velocidade atual.

Parte deste trabalho está conectada ao Módulo Web, pois é através dele que podemos visualizar dados em tempo real.

1.5. Módulo Móvel

De maneira similar ao Módulo Web, o Módulo Móvel está voltado ao usuário final. Seu objetivo é facilitar a integração e visualização desses dados em dispositivos móveis. Seu objetivo principal está em uma aplicação para visualizar informações em tempo real providas pelo Módulo Gerente, como por exemplo, a posição atual do ônibus e tempo estimado para a chegada no ponto de parada. Rocha (2010) descreve em seu trabalho sobre a integração desse módulo com o Modulo Gerente, além do desenvolvimento do módulo em si, utilizando a plataforma *Android* do *Google*.

1.6. Organização deste documento

O trabalho está organizado da seguinte maneira:

No Capítulo 2 é apresentado o referencial teórico, onde serão abordados assuntos como o transporte público, alguns Sistemas Inteligentes de Transporte (ITS) (ITIF, 2010) como o AVL (*Automatic Vehicle Location*), os sistemas de posicionamento global, os sistemas de informação geográfica e alguns trabalhos relacionados como o *Transport Direct* no Reino Unido. Ainda no Capítulo 2 serão abordados temas mais técnicos utilizados na implementação da aplicação como, por exemplo, PostGIS (PostGIS, 2010), ORM (*Object-Relational Mapping*) (Fowler, 2010), *Servlets* (*Java Servlet Technology*, 2010), e *Google Maps* (Google Maps API, 2010).

No Capítulo 3 abordamos a metodologia utilizada, com a classificação da pesquisa e as atividades exercidas. No Capítulo 4 são apresentados os resultados obtidos e discussões acerca desses resultados, e no Capítulo 5 a conclusão do trabalho.

2. REFERENCIAL TEÓRICO

2.1. Transporte Público

O transporte público exerce papel fundamental na forma de interligação entre bairros, cidades e estados, possibilitando uma forte alternativa para redução de problemas graves que atingem as cidades atualmente, como por exemplo, poluição, congestionamentos e acidentes de trânsito.

O motivo para os deslocamentos variam entre trabalho, estudos, diversão, entre outros, abrangendo assim uma parte razoável da população. De acordo com a Associação Nacional de Transporte Público (ANTP, 2010), são realizadas aproximadamente 0.47 viagens de transporte coletivo por habitante por dia no Brasil, gerando uma demanda de 16.8 bilhões de viagens por ano.

Uma parcela razoável, aproximadamente 29.4% da população do Brasil, utiliza o transporte público, chegando a 36% em municípios com mais de um milhão de habitantes (ANTP, 2010). Tal variação fica clara se associarmos que à medida que o setor terciário evolui nas cidades, a necessidade por maiores deslocamentos cresce. Mantendo um transporte público bem estruturado, as cidades garantem uma melhor circulação do trânsito, que não afete tanto o ambiente e não aumente ainda mais os congestionamentos. Tais medidas influenciam desde a qualidade de vida dos usuários do transporte até a cidade, que aumenta em produtividade.

Segundo Travassos (2000), citado por Azambuja (2002), os itens que mais contribuem para a melhora na qualidade de serviço do transporte público estão relacionados à infra-estrutura viária e à tecnologia dos veículos, e será neste último que este trabalho irá focar.

2.2. Sistemas Inteligentes de Transporte (ITS)

Intelligent Transportation Systems (ITS) (ITIF, 2010) é um conceito relativamente novo na literatura, e tem sido usado para se referir aos esforços para adicionar tecnologias de informação e comunicação em sistemas de transporte.

Os problemas referentes ao transporte em geral e suas relações com a logística, segurança, gerenciamento, e outros itens importantes para o setor, já perduram há tempos. Bem como o fato de que a necessidade de transporte é crescente à medida que as cidades aumentam e mais atividades surgem. A novidade se dá no acompanhamento, cada vez mais próximo (muitas vezes em tempo real), no controle, e nas intervenções que se podem realizar nas diversas etapas do transporte, com impactos em logística e nas atividades relacionadas. Tais mudanças se devem aos avanços tecnológicos da humanidade, com sistemas de rastreamento em tempo real, posicionamento relativo por satélite, maior cobertura de satélites da superfície terrestre, e aplicações no geral, que visam obter melhores desempenhos e dinamismo na visualização de informações obtidas da Terra.

Podemos encontrar sistemas ITS relacionados a vários aspectos dentro do setor de transporte, como o transporte de carga ou passageiros, veículos inteligentes, monitoramento e gerenciamento de tráfegos e vias, viagens, entre outros. Esses sistemas podem trabalhar tanto com dados obtidos em tempo real quanto com dados passados, que são aproveitados para fazer projeções para informações futuras.

ITS é um sistema que abrange várias tecnologias de informação avançadas como, tecnologia de transmissão de dados, tecnologia de sensores, tecnologia de controle eletrônico, controle de computador, e tecnologia de processamento de sinal. Tudo isto com o objetivo geral de integrar todo o

gerenciamento do tráfego, e construir, em grande escala e tempo real, um sistema de gerenciamento do tráfego preciso e eficiente. Tem-se comprovado que o ITS realmente pode melhorar o ambiente de trânsito, e resolver uma boa parte dos problemas atuais no setor de transporte de maneira eficaz e dinâmica, como o congestionamento do tráfego (Gang Jing et al., 2006).

Segundo ITIF (2010), pode-se subdividir os sistemas ITS em cinco categorias primárias: Advanced Traveler Information Systems (ATIS), Advanced Transportation Management Systems (ATMS), Sistemas de Tarifação do Transporte, Vehicle-to-Infrastructure Integration (VII) e Vehicle-to-Vehicle Integration (V2V), e Advanced Public Transportation Systems (APTS). Cada categoria é explicada de forma breve nas seções seguintes.

2.2.1. Advanced Traveler Information Systems (ATIS)

O objetivo de um sistema ATIS é de fornecer aos condutores informações em tempo real, tais como rotas de trânsito e horários, instruções de navegação e informações sobre atrasos devido a congestionamentos, acidentes, condições meteorológicas, ou trabalhos de reparação da estrada. Um exemplo típico são os aparelhos que andam surgindo no mercado, comumente chamados de navegadores GPS. Os sistemas ATIS mais eficazes são capazes de informar aos condutores sua localização exata em tempo real, informá-los sobre a situação do tráfego atual ou as condições das estradas e rodovias a seu redor, e capacitá-los com a escolha da melhor rota possível e com instruções de navegação precisas. Esses sistemas compreendem as seguintes aplicações:

- Visualização de informações relativas ao tráfego em tempo real;
- Sistema de navegação e guias de rota;
- Informações relativas a estacionamentos disponíveis;
- Sistema de informações metereológicas nas estradas;

Conforme listado acima, outra aplicação de um sistema ATIS interessante é o de tornar a procura por estacionamentos mais fácil. Cidades como Singapura, Lisboa e São Francisco estão implantando sistemas que indicam aos motoristas onde podem ser encontrados estacionamentos vagos na cidade, e até mesmo permitir que motoristas reservem a vaga com antecedência (Figura 2.1). Estudos têm demonstrado que mais de 30% do congestionamento causado nas grandes cidades é devido a motoristas que ficam circulando, em busca de estacionamento (ITIF, 2010).

ɡura 2.1: Parking Guidance System, Singapura.
ıte: ITIF (2010)

2.2.2. Advanced Transportation Management Systems (ATMS)

Advanced Transportation Management Systems (ATMS), ou Sistema de Gerenciamento do Transporte, inclui aplicações ITS que focam em dispositivos de controle de tráfego, como semáforos dinâmicos, e painéis eletrônicos em rodovias que exibem mensagens dinâmicas informando aos motoristas sobre situações gerais na estrada, como o tráfego, suas condições, acidentes etc. Tudo isso em tempo real.

2.2.3. Sistemas de Tarifação do Transporte

Não existe uma sigla própria para indicar o conjunto de sistemas responsáveis por cobrar tarifas automaticamente, porém vários sistemas desse tipo são conhecidos. Dentre eles, o que mais se destaca é o *Electronic Toll Collection* (ETC), em que o usuário pode pagar pedágios automaticamente, bastando apenas ter um aparelho em seu automóvel que descontará um valor cada vez que for necessário. Um exemplo de um sistema ETC é o “Sem Parar/ViaFácil”, utilizado nos pedágios em algumas rodovias no Brasil, onde não é necessário parar o automóvel para pagar o pedágio, basta passar pelo corredor onde o sistema está instalado.

Os Sistemas de Tarifação do Transporte são muito utilizados para combater o congestionamento de tráfego. Um sistema relativamente novo é o de vias HOT (*High-Occupancy Toll*), que são vias reservadas para ônibus e outros veículos de alta ocupação, mas que podem ser disponibilizadas para outros veículos mediante o pagamento de uma taxa.

Outro Sistema de Tarifação do Transporte que está começando a ser implantado em alguns países é o VMT (*Vehicle Miles Traveled*), onde taxas são cobradas dos motoristas para cada quilometro rodado. Esses sistemas de taxas VMT representam uma alternativa ao imposto cobrado atualmente sobre os combustíveis e outras taxas que muitos países e estados usam a financiar seus sistemas de transporte. O programa "*Holanda Kilometerprijs*" (preço por quilometro) está previsto para ser o primeiro sistema mundial de VMT implementado tanto para veículos de passageiros quanto veículos pesados. O programa, que começará com o transporte de mercadorias em 2012, seguido por veículos de passageiros em 2016, usará a tecnologia avançadas de satélites juntamente com um sistema implantado no veículo para cobrar os usuários com base na quilometragem rodada. Alemanha já está cobrando o transporte de

mercadorias com base nesse sistema. Nos Estados Unidos, a Comissão Nacional de Financiamento em Infra-Estrutura de Transportes recomendou em fevereiro de 2009 a implantação de um sistema de cobrança de taxas baseados no VMT dentro de uma década (ITIF, 2010).

2.2.4. *Vehicle-to-Infrasctructure Integration* (VII) e *Vehicle-to-Vehicle* (V2V)

Os sistemas VII e V2V permitem a comunicação entre os vários elementos que compõem um sistema de transporte, por exemplo, a comunicação de veículos com sensores na estrada que podem informar se o carro está saindo da pista, ou mesmo a comunicação entre dois veículos.

Um exemplo de VII relativamente novo e muito interessante é o IntelliDrive^{SM}, um sistema que pode ser instalado em cada veículo, tornando-os capazes de trocar informações entre si. Com esse sistema podemos permitir que os veículos tenham consciência de tudo que acontece a sua volta, informando o condutor do veículo de perigos e situações que não podem ver. Aplicações de segurança IntelliDrive^{SM} têm o potencial para reduzir acidentes com alertas e avisos. Por exemplo, o motorista do veículo pode ser avisado ao entrar em uma zona escolar, ou ao aproximar de uma curva acentuada, ou de um pedaço da estrada escorregadio logo à frente. Poderíamos por exemplo, ativar serviços de emergência caso o automóvel pare de funcionar ou até mesmo sofra uma batida (IntelliDrive, 2010).

2.2.5. *Advanced Public Transportation Systems* (APTS)

Advanced Public Transportation Systems (APTS) inclui aplicações como localização automática de veículos (AVL), que permite relatar a localização atual de veículos em trânsito como ônibus ou metrôs, tornando

possível aos gestores do tráfego obter uma visão em tempo real do status de todo sistema de transporte público. O uso de sistemas APTS pode ajudar a tornar o transporte público uma opção mais atraente para os viajantes, dando-lhes maior visibilidade em relação ao status de ônibus e trens, como o horário de chegada e de partida. Esta categoria também inclui sistemas eletrônicos de pagamento de tarifa para os sistemas de transporte público, como o Suica, no Japão ou o T-Money, na Coreia do Sul, que permitem aos usuários pagarem as tarifas através do uso de seus cartões inteligentes (S*mart C*ards) ou celulares (ITIF, 2010).

Para esse tipo de sistema de pagamento de tarifa eletrônico, também chamado de SAAT (Sistemas Automatizados de Arrecadação Tarifária) por Silva (2000), não é preciso ir muito longe. Cidades como São Paulo, Goiânia, Recife, e várias outras já possuem esse sistema implantado desde antes de 2000. Com a implantação desse tipo de sistema de bilhetagem eletrônica é possível obter vários benefícios, como por exemplo, a redução dos problemas de troco e consequentemente a do tempo de embarque, a diminuição do risco de assaltos, devido à menor circulação de dinheiro dentro dos ônibus, a supressão do comércio paralelo de vales-transporte, assim como o fim da falsificação de vales-transporte e de carteiras de descontos/gratuidades, e por fim, a redução dos custos operacionais com o processo de prestação de contas dos cobradores (Teixeira, 2005).

2.3. Localização Automática de Veículo (AVL)

Automatic vehicle location (AVL), é o significado para determinar automaticamente a localização geográfica de um veículo e transmitir a informação para quem a requisitou. Seu uso é mais comumente associado ao sistema GPS, porém, o AVL também pode ser utilizado com outras tecnologias

de posicionamento, como rádio frequência, rede de telefonia móvel, híbridos, etc.

O AVL é constituído por subsistemas de aquisição dos dados e atuação, equipamentos embarcados, comunicação e gestão das informações. De maneira geral, o sistema AVL coleta as informações de posicionamento geográfica dos veículos pelo subsistema de aquisição de dados. Depois, usando o subsistema de comunicação, transmite as informações a uma central de controle, onde é realizado a análise e processamento das informações, por meio da integração dos dados de posicionamento com base de dados existente, ferramentas GIS (*Geographic Information System*) e softwares especializados. Esse subsistema de gestão da informação lida com informações relevantes associadas aos deslocamentos do veículo, fornecendo funcionalidades ao sistema, além de gerenciar o recebimento e o envio de informações entre o veículo e central que requisitou as informações (Cunha, 2008).

Segundo Amarante (2007), os produtos AVL normalmente são configurados com taxas de atualização que ficam entre 1 a 15 minutos, de maneira a fornecer uma boa margem de custo/benefício para as aplicações que necessitam desse tipo de serviço atualmente. Porém, a decisão de qual taxa de atualização escolher pode afetar profundamente todo o sistema AVL. Se aumentarmos muito a taxa de atualização, poderemos obter maiores gastos com a comunicação e armazenamento de dados. Em contrapartida, se diminuirmos muito a taxa de atualização, o sistema pode confundir dados e tirar conclusões errôneas, além de agravarmos a representação do trajeto, principalmente se tratando de áreas urbanas. Por isso é importante a escolha de uma taxa de atualização adequada dependendo do tipo de serviço prestado, do local e da tecnologia de posicionamento empregada.

Um último fator importante a se destacar em AVL, é que não devemos confundi-lo com o sistema EATS (*Events Activated Tracking System*). O sistema

EATS, mais utilizado em soluções de segurança, funciona de maneira que o sistema só envia a sua localização caso algum evento ocorra, como por exemplo, se um ladrão tentar roubar o veículo. No caso do AVL, mais utilizado para gerenciamento de frota, cada unidade é configurada para enviar a sua posição geográfica após um determinado tempo, por exemplo, a cada 5 minutos. Os objetivos de um sistema AVL não se limitam somente na localização de veículos, mas também na obtenção de informações importantes relativas ao veículo, como consumo de combustível, informações sobre o estado mecânico e elétrico do veículo, dados do motorista, e dados de sensores do veículo, isto é, travamento das portas, pressão do ar, entre outros. Todos esses dados dependem de quais equipamentos o veículo possui e o que ele pode nos oferecer com tais equipamentos.

2.4. Sistemas de Posicionamento

Segundo Silva (2006), podemos dividir os sistemas de posicionamentos empregados em sistemas AVL em quatro tecnologias: *dead-reckoning* (DR), postos fixos, triangulação de antenas de rádio e posicionamento por satélites.

2.4.1. *Dead-Reckoning*

Dead Reckoning (DR) é o processo de estimar a posição atual de um objeto baseando-se em uma posição previamente determinada, ou então prever e corrigir uma futura posição baseando-se em valores de velocidade conhecidos ou estimados durante o tempo decorrido, e curso. Embora os métodos tradicionais de *dead-reckoning* já não serem considerados os principais meios de navegação, sistemas modernos de navegação inercial ainda os utilizam.

Para fazer esses cálculos, normalmente são utilizados instrumentos como um giroscópio, para determinar a direção que o veículo se move, e um hodômetro, para registrar os quilômetros percorridos desde a última atualização. Uma desvantagem do *dead-reckoning* é que desde que as novas posições são calculadas apenas a partir de posições anteriores, fazendo com que os erros do processo sejam cumulativos, portanto, o erro na posição fixa cresce com o tempo.

Atualmente, o sistema de *dead-reckoning* costuma ser aplicado em alguns sistemas de navegação automotiva, a fim de superar as limitações da tecnologia GNSS (*Global Navigation Satellite Systems*) sozinha. Normalmente, sinais por satélite estão indisponíveis em parques de estacionamento fechados e túneis, e por vezes, seu sinal é severamente degradado em centros urbanos ou lugares com muitos obstáculos concentrados, impedindo a propagação de sinais. Em um sistema de navegação *dead-reckoning*, o carro é equipado com sensores que registram a rotação da roda e a direção do volante. Esses sensores muitas vezes já estão presentes nos carros para outros fins (sistema de controle de frenagem, sistemas para controle de estabilidade) e pode ser lido pelo sistema de navegação. Combinando o *dead-reckoning* com algum sistema de posicionamento por satélite, o dispositivo de navegação pode oferecer uma maior precisão sobre a posição atual do veículo.

Segundo Casey et al. (1998, citado por Silva, 2006), é possível notar alguns exemplos da aplicação de *dead-reckoning* no transporte público no começo dos anos 90. Em 1991, a *Street Railway Company*, na cidade de Hamilton (Ontario, Canadá), utilizou o *dead-reckoning* para obter a localização aproximada de seus 240 ônibus. Para corrigir os erros que poderiam se acumular utilizando o *dead-reckoning*, devido a superfícies irregulares ou pequenas falhas de precisão, a companhia adicionou dois postos fixos ao longo de todas as rotas para que o sistema seja calibrado periodicamente. Em 1992 as prefeituras de

Houston (Texas, EUA) e Santa Monica (California, EUA), também implantaram um sistema que utilizava *dead-reckoning* em toda a frota pública, como ônibus, carros, motos, etc., o objetivo desse sistema era de fornecer informações para identificar problemas no trânsito.

2.4.2. Postos fixos

Os Postos fixos, ou *Signpost*, são muito utilizados em sistemas ETC (*Electronic Toll Collection*), citados anteriormente em Sistemas de Tarifação do Transporte. Normalmente o *Signpost*, é composto por dois dispositivos: uma etiqueta inteligente, também conhecida como *smart-tag* ou *e-tag*, e uma antena transmissora. Essas *smart-tags* são compostas por um chip com uma pequena memória e uma antena para a comunicação com a antena transmissora. Um exemplo de aplicação ETC que utiliza *Signpost* é o sistema "Sem Parar/ViaFácil", utilizado nos pedágios de algumas rodovias do Brasil. O motorista coloca sua *smart-tag* no pára-brisas do veículo, essa *smart-tag* possui algumas informações como a quantidade de crédito que o motorista tem, assim que o veículo passa por um posto de pedágio a antena transmissora e a *smart-tag* iniciam uma conexão onde é debitado o valor do pedágio no chip de memória dentro da *smart-tag*. Assim o motorista não precisa parar para pagar o pedágio, evitando congestionamentos, e diminuindo o tempo de viagem do motorista.

O problema da utilização desta tecnologia é que o sistema fica dependente das antenas transmissoras, e em muitos casos é inviável a implantação de várias antenas transmissoras para cobrir a área desejada. Sua aplicação muitas vezes acaba ficando restrita à cobrança automática de pedágios, controle de acessos, rastreamento de produtos, entre outros.

2.4.3. Triangulação de antenas

Esse sistema de triangulação de antenas utiliza sinal de rádio, possibilitando assim localizar o objeto quando este estiver dentro da área de triangulação. Essa técnica também é conhecida como RDF (*Radio Direction Finder*). Os problemas na utilização de triangulação por antenas de rádio envolvem a abrangência, que depende da quantidade e da potência das antenas instaladas, problemas relativos a obstáculos, como prédios ou relevos, interferências, que podem prejudicar a qualidade do sinal, principalmente em grandes centros urbanos, entre outros.

2.4.4. Posicionamento por Satélites

GNSS (*Global Navigation Satellite Systems*), ou Sistemas de Posicionamento Geodésicos por Satélites, são sistemas capazes de descobrir a posição de um objeto ou indivíduo na superfície terrestre, usando coordenadas fornecidas por satélites. Dentre os sistemas GNSS, os mais famosos são o NAVSTAR-GPS (*Navigation System Using Time and Ranging* / *Global Positioning System*), de origem americana, o GLONASS (*Global Navigation Satellite System*), de origem russa, e o Galileu, de origem européia. As únicas desvantagens encontradas na utilização de posicionamento por satélites é a possibilidade de perda de sinal devido a obstáculos como prédios, montanhas, etc.

Para este sistema, basearemos no Sistema de Posicionamento GPS, por ser mais difundido.

2.4.4.1. Sistema de Posicionamento Global (GPS)

O GPS foi desenvolvido e é gerenciado pelo Departamento de Defesa dos EUA, foi arquitetado originalmente para aplicações militares, e depois disponibilizado para uso civil. Os aparelhos GPS fazem triangulações com os sinais recebidos via satélite e calculam sua posição na superfície terrestre, baseando nessas informações obtidas por cada satélite. Seu principal objetivo é auxiliar na navegação em três eixos com alta precisão, mesmo que o receptor esteja sujeito a condições dinâmicas variadas, sofrendo pouca influência das variações no relevo terrestre. Quanto maior a quantidade de satélites, e quanto mais afastados estiverem entre si, melhor será a precisão da coordenada solicitada.

Este sistema é constituído pelos segmentos espacial (a), de controle (b), e do usuário (c) (Silveira, 2004):

a) O segmento espacial (Figura 2.2) possui atualmente 24 satélites a aproximadamente 20.200 quilômetros acima da Terra, distribuídos em seis órbitas planas inclinadas em 55º em relação ao Equador, que funcionam de modo preciso, rápido e praticamente ininterrupto.

Com esta configuração em qualquer ponto sobre a superfície ou próximo a ela haverá um mínimo de quatro satélites acima do horizonte, fazendo com que cada satélite tenha um período de 12 horas siderais como duração de uma volta ao redor da Terra.

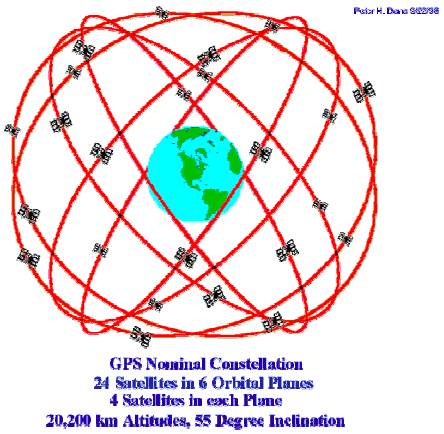

Figura 2.2: Segmento Espacial GPS. Fonte: (Dana, 2010)

b) O segmento de controle é composto por um conjunto de cinco estações terrestres que recebem os sinais dos satélites, calculam medidas meteorológicas e enviam as informações a uma estação principal, que calcula as informações sobre as órbitas dos satélites, bem como os coeficientes de correção dos relógios e os envia para as estações de transmissão.

c) O segmento do usuário abrange o conjunto de usuários militares e civis do sistema, tratando-se basicamente dos aparelhos receptores que possuem a funcionalidade de armazenar e analisar os sinais enviados pelos satélites. Podemos aproveitar esses receptores para diversos fins, como por exemplo, traçar rotas entre várias localidades, análise de tráfego, monitoração de frotas, aplicações militares, entre outras.

Segundo (Marques, 2006), o principio básico para o funcionamento de um sistema GPS se deve a medição da distância entre a antena do receptor e as antenas dos satélites da constelação GPS (segmento espacial). A partir da medição dessa distância entre a antena do receptor e de quatro satélites ao mesmo tempo, podemos determinar a posição geográfica (coordenada) da antena do receptor (Figura 2.3).

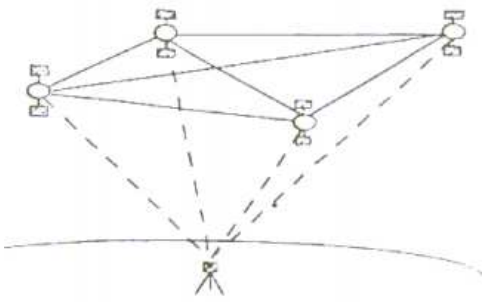

Figura 2.3: Posicionamento GPS. Fonte: Silveira (2004)

2.5. Transmissão de dados

Um dos maiores problemas enfrentados na implantação de um sistema para rastreamento está na comunicação entre as unidades móveis. Para obter a posição global atual de um objeto, basta utilizar um sistema de posicionamento de satélites como o GPS, pois seus custos se limitam somente ao dispositivo GPS. Porém se quisermos transmitir dados entre unidades móveis, o custo já começa a se tornar relevante.

2.6. Sistemas de Informação Geográfica (GIS)

GIS ou *Geographic Information System* é um sistema responsável por coletar e analisar informações geograficamente referenciadas, criando uma associação do mundo físico com um modelo determinado onde cada atributo desse modelo representa essas informações físicas. Ele permite aos usuários manipular os dados para que eles possam resolver os problemas geográficos através de cálculo de estatística espacial, processamento de consultas, entre outras coisas. Também é usado para gerenciar a visualização de informações

geográficas, não só informações físicas como o relevo, como também fenômenos geográficos, tais como os impactos do tráfego em certo local, surtos de doenças e inundações. Os resultados são freqüentemente visualizados na forma de um mapa.

Atualmente é possível notar que o termo Sistemas de Informações Geográficas está extremamente ligado com as áreas de tecnologia da informação que utilizam dados georeferenciados e objetos espaciais. Esses sistemas envolvem diversas aplicações como cadastro de dados geográficos, consultas, análises espaciais, rastreamento, SIG-WEB, tratamento de informações entre outras. Porém, conforme Silva (2009), a sigla SIG não pode ser usada somente com o intuito de referenciar informações geográficas relacionadas às tecnologias atuais, o termo já vem sendo utilizado desde 1960, onde as definições ainda não eram muito claras, porém alguns destaques no desenvolvimento da tecnologia eram obtidos através de esforços pessoais. A sigla SIG normalmente é encontrada em documentos com o intuito de mencionar sistemas que manipulam informações geográficas/espaciais, tais termos são amplos e devem possuir uma definição mais abrangente.

Segundo Toscano (2006), dentre as principais vantagens do uso de GIS na Internet estão, a redução do custo de investimento em software e hardware, pois com apenas um navegador Web é possível acessar o sistema, livrando o cliente da necessidade de instalação de software localmente, e a independência de plataforma, ou seja, arquiteturas de hardware e software utilizadas nos clientes não afetam diretamente o funcionamento do sistema. A maior desvantagem destacada por Toscano estava na necessidade de uso de banda larga, que atualmente já não um grande empecilho.

Quando utilizados de maneira correta e com objetivos claros, os aplicativos desenvolvidos utilizando as tecnologias SIG, contribuem para que os profissionais possam obter maior rapidez, segurança e economia na extração e

apresentação de dados e informações em mapas digitais georeferenciados, facilitando a análise de todo o universo de dados sem ter de se preocupar com pequenos detalhes físicos, e somente na informação que aqueles dados podem oferecer (Silva, 2009).

2.7. Trabalhos Relacionados

Neste tópico serão abordados alguns trabalhos similares ao proposto, dando certo enfoque em dois sistemas: *Transport Direct*, um sistema utilizado no Reino Unido e *Magic Bus*, utilizado na Universidade de Michigan, nos Estados Unidos. O motivo para o enfoque nesses sistemas é que ambos são portais Web voltados para o transporte público. Logo após será citado alguns outros projetos também voltados ao transporte público, porém utilizando tecnologias diferentes.

2.7.1. *Transport Direct* (Reino Unido)

Transport Direct é um portal Web que oferece serviços de planejamento de viagens dentro do Reino Unido. Ele foi desenvolvido pela companhia *Atos Origin*, que foi escolhida em um consórcio feito pelo Departamento de Transporte do Reino Unido (*Departament for Transport*) em Janeiro de 2003 (*Transport Direct*, 2010). O serviço inclui um sistema para planejar viagens, mapas, informações sobre as viagens, tarifas ferroviárias e serviços de venda de bilhetes. À medida que cresce, o portal espera também integrar outros serviços, como informações sobre hotéis, restaurantes e pontos de interesse.

Alguns fatores interessantes podem ser ressaltados nesse sistema, por exemplo:

- Caso disponível, o sistema leva em consideração o histórico de congestionamento em cada local para calcular o tempo total de uma viagem e qual o caminho mais adequado de acordo com o horário da viagem;
- É possível comprar tickets de ônibus e de metrô antecipadamente evitando a superlotação e facilitando a disponibilidade de transportes de acordo com a demanda;
- É compatível com celulares e PDA's provendo, assim, informações em praticamente qualquer localidade;
- Possui um serviço que calcula o quanto de CO2 será emitido caso a viagem seja feita de carro e também informa o quanto gastaria caso utilizasse outros serviços, como por exemplo, o transporte público;

Todas essas informações contribuem para o melhor planejamento de viagem do usuário de acordo com suas necessidades. Assim, é possível incentivar o usuário a utilizar mais os serviços de transporte público, diminuindo o congestionamento enfrentado atualmente em grandes cidades e influenciando em vários outros fatores já citados anteriormente.

A Figura 2.4 mostra um exemplo de uma viagem de Liverpool a Manchester feita no portal *Transport Direct*. Ela nos informa as possibilidades de viagem, qual o tipo de transporte terá que ser usado, quantas mudanças terão que ser feitas, e a duração de cada viagem. Pelo menu logo acima é possível ver mapas, preços de passagens, alterar a rota ou até mesmo verificar quanto de CO2 será gasto para cada tipo de viagem. Logo abaixo o sistema também informa passo-a-passo de como será a viagem, descrevendo quantos minutos levará cada passo. Por exemplo, "Pegue o metrô *London Underground/Hammersmith* rumo a Estação de metrô *King's Cross*", "Ande 13 minutos em direção a Estação de metrô *King's Cross*", etc.

Details | Summary | Maps | Tickets/Costs | Modify journey | Check CO2

Outward journeys for Thu 17 Jun 10 leaving after 20:30

Option	Transport	Changes	Leave	Arrive	Duration	Select
1	**Train, Tram/Light Rail, Underground, Walk**	**2**	**20:34**	**23:37**	**3hours, 03 mins**	◉
2	Metro, Train, Tram/Light Rail, Underground, Walk	4	23:04	07:01 (18/06)	7hours, 57 mins	○
3	Metro, Train, Tram/Light Rail, Underground, Walk	4	05:38 (18/06)	09:19 (18/06)	3hours, 41 mins	○
4	Car	0	20:30	00:26 (18/06)	3hours, 56 mins / 203.0miles	○

Figura 2.4: Exemplo de rota criada no sistema Transport Direct.
Fonte: Transport Direct (2010)

2.7.2. *Magic Bus* – Universidade de Michigan

O projeto *Magic Bus* foi iniciado em 2004 com o objetivo de melhorar a experiência dos usuários de ônibus na Universidade de Michigan, permitindo-lhes ver onde os ônibus estão em tempo-real e quanto tempo falta para o ônibus alcançar certo ponto.

Usando o Sistema de Posicionamento Global (GPS), o sistema é capaz de exibir dados de localização dos ônibus que transitam pela universidade praticamente em tempo real, com menos de quatro segundos de latência. Sua característica essencial para os usuários é a possibilidade de apresentar uma estimativa de quantos minutos faltam para o próximo ônibus chegar em um determinado ponto de parada, assim os usuários sabem qual o horário que devem se dirigir ao ponto de ônibus mais próximo.

No portal Web criado, é possível visualizar e selecionar rotas e paradas, obtendo assim quanto tempo vai levar para os dois próximos ônibus chegarem no ponto de parada (Figura 2.5). O portal também possui uma versão para dispositivos móveis, incluindo um aplicativo para o *iPhone*. Esse é um dentre os vários projetos desenvolvidos pela Universidade de Michigan que visam aprimorar toda a infra-estrutura de transporte dentro da universidade. A

universidade também incentiva a utilização de serviços como o *Zimride*, que é outro portal web criado com o objetivo de conectar pessoas que possam oferecer ou solicitar caronas, economizando gastos com o transporte, como combustível e manutenção, diminuindo o congestionamento, liberando mais vagas de estacionamento, e reduzindo a emissão de poluentes. Outro portal web também similar é o *GreenRide*, que possui praticamente os mesmo objetivos, sempre incentivando o compartilhamento de carros e o uso de vans para chegar ao trabalho (*University of Michigan Parking & Transpotation Services*, 2010).

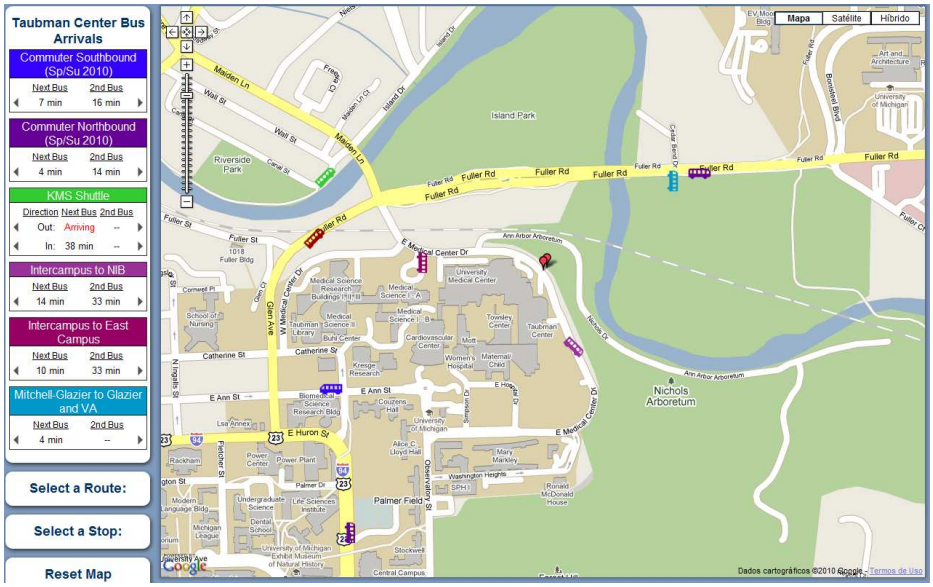

Figura 2.5: Portal web Magic Bus. Fonte: (http://mbus.pts.umich.edu)

2.7.3. Outros

O *Countdown* é um sistema de informação em tempo real utilizado em Londres, Inglaterra, que fornece aos passageiros o horário da chegada do ônibus ao ponto de parada. Conforme mostra a Figura 2.6 o sistema é composto de

painéis eletrônicos instalados nos pontos de parada de ônibus que atualizam a cada 30 segundos informando qual será o próximo ônibus a chegar ao ponto, qual seu destino, e quantos minutos faltam para chegar. Conforme o TFL (2010), o sistema cobre cerca de 10% da rede de pontos de ônibus da Inglaterra, porém a partir de 2011 serão introduzidos novos painéis eletrônicos com o objetivo de cobrir cada um dos 19.000 pontos de paradas de Londres, sendo que 2.500 estarão implantados até 2012. Também haverá melhoramentos em relação ao atual sistema, o novo sistema vai fornecer informações em tempo real sobre a chegada de ônibus através de uma série de canais, incluindo envio de mensagens de texto a um baixo custo e acesso por web de graça. Devido a essa e outras medidas que visam melhorar seu sistema de transporte público, Londres pode notar uma taxa de crescimento de 40 por cento em viagens de passageiros entre 2001 e 2010.

Figura 2.6: Sistema Countdown em Londres, UK. Fonte: TFL (2010)

Análogo ao portal web *Transport Direct temos* o SIT (*Sistema de Información de Transportes*) em Madri, Espanha. Criado em 1986, o SIT fornece aos usuários informações relativas a metrôs, ônibus municipais e intermunicipais, e trens, com possibilidade de criar rotas, verificar itinerários e se informar sobre problemas ou mudanças de rotas. O número de passageiros utilizando esses serviços tem crescido de 951.000.000 em 1986 para 1.598.200.000 em 2008, que é uma boa indicação do investimento e esforço de coordenação feito nos últimos anos (SIT, 2010).

Os Terminais Públicos Interativos, ou PIT (*Public Interative Terminals*) são sistemas voltados ao fornecimento de informações ao usuário antes do início da viagem, que permitem a tomada de decisões em relação ao horário da partida, o meio de transporte, a rota, entre outros. Para possibilitar o usuário a tomar tais decisões, o sistema pode fornecer informações relevantes como possíveis rotas, mapas, custos, tempo, postos de abastecimento, ou até informações que costumam serem chamadas de "pontos de interesse", representados por parques, museus, cachoeiras, e atrações turísticas em geral. Como exemplo de PIT temos o *TRIPlanner*, no Reino Unido, e o *FITs*, na Itália, que fornecem dados relativos ao transporte público, condições de tráfego e qualidade do ar (Teixeira et al., 2005).

Uma estratégia adotada em muitos projetos de ITS é a de controle de tráfego adaptativo. O SCOOT é um exemplo de sistema desse tipo, onde através de uma central é possível coordenar o funcionamento de todos os semáforos em uma área, oferecendo assim, uma boa progressão dos veículos através da rede. O sistema possui um detector de veículos que fica armazenando em um banco de dados informações a respeito do fluxo de automóveis de acordo com o horário do dia, assim, ao se aproximar de um cruzamento, o veículo aciona o detector e, baseando no histórico até o momento e nas prioridades, o semáforo mudará de estado, acionando a luz vermelha, amarela, ou verde (Figura 2.7). Assim o tráfego fluirá de maneira mais natural, podendo reduzir significativamente o tempo de espera de um semáforo. Essa estratégia foi utilizada em várias grandes cidades como São Paulo, Londres, Beijing, Toronto, e Santiago. Uma abordagem interessante seria a de utilizar do sistema de controle adaptativo para dar prioridade a certos veículos, como ônibus. Basta equipar um ônibus com um sistema AVL (*Automatic Vehicle Location*), para localizar o veículo, e um sistema SVD (*Selective Vehicle Detectors*), que fará com que o ônibus tenha prioridade em cima de outros veículos. Em 1995, um experimento feito na

cidade de Southampton, na Inglaterra, mostrou que a utilização desse sistema pode reduzir a jornada de um ônibus em até 60%, sendo que tal fator depende do horário (SCOOT, 2010).

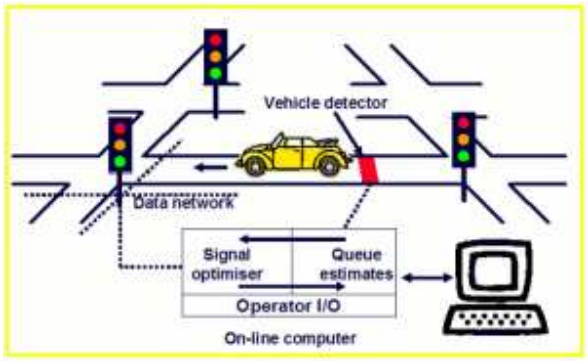

Figura 2.7: Sistema de controle de tráfego adaptativo. Fonte: SCOOT (2010)

2.8. Ferramentas Utilizadas

2.8.1. Banco de Dados Geográfico

Um Sistema de Informação Geográfica normalmente envolve a análise de uma grande quantidade de dados, e consequentemente a necessidade de registrar esses dados em um meio de persistência. Para facilitar a manipulação de dados geográficos precisamos de um banco de dados capaz de armazenar estruturas espaciais de maneira transparente ao usuário, assim como ler tais estruturas e traduzi-las de volta para uma forma que o usuário entenda. Bancos de dados geográficos podem nos fornecer tais funcionalidades, além de uma gama de funções pré-definidas capazes de efetuar operações trabalhosas de uma forma simples, como por exemplo, obter todos os pontos que estão contidos em um polígono, ou então informar qual o ponto mais próximo de certa linha.

2.8.1.1. PostgreSQL/PostGIS

O PostgreSQL (PostgreSQL, 2010) é um SGBD (Sistema Gerenciador de Banco de Dados) objeto-relacional de código aberto, tem mais de 15 anos de desenvolvimento, roda em quase todos os Sistemas Operacionais (GNU/Linux, Unix, MS Windows) e é totalmente compatível com o conceito ACID (Atomicidade, Consistência, Isolamento e Durabilidade) de banco de dados, onde cada sigla tem um significado:

- Atomicidade: A transação deve ter todas as suas operações executadas em caso de sucesso ou nenhuma, caso uma ou mais dessas operações falhem.
- Consistência: As regras do banco de dados não podem ser quebradas, tais regras podem envolver referências válidas ou inválidas, campos nulos, etc. Por exemplo, caso algum elemento que está sendo referenciado por outro for excluído, algumas providências devem ser tomadas para manter a consistência como marcar a chave que o referencia como nula, ou excluir o elemento que o referencia (exclusão em cascata), ou abortar a transação.
- Isolamento: Outras operações não podem acessar os dados que estão sendo modificados durante uma transação ainda não concluída, ou seja, duas transações só podem ser simultâneas caso elas não forem alterar os mesmos dados.
- Durabilidade: Após uma transação ter sido executada com sucesso, seus efeitos devem permanecer no banco, mesmo que ocorram falhas posteriores.

Uma das maiores vantagens do PostgreSQL é sua extensibilidade, que permite incorporar capacidades adicionais ao sistema, tornando-o mais simples

e flexível para gerenciar os dados dependendo da aplicação. No caso do GIS, isso torna possível o desenvolvimento de uma extensão geográfica capaz de armazenar, recuperar, alterar, excluir, e analisar dados espaciais em geral. Tal extensão geográfica é chamada de PostGIS (Silva, 2009).

O PostGIS é uma extensão/módulo do PostgreSQL que possui o padrão estabelecido na SFS (*Simple Features Specification for SQL*), do consórcio *Open Geospatial Consortium*, também chamado de *OpenGIS Consortium* (OGC, 2010). Juntamente com a biblioteca GEOS (*Geometry Engine – Open Source*), o PostGIS passa a oferecer mais de 130 funções e operadores para lidar com dados geográficos, além de contemplar o padrão SFS para objetos espaciais como ponto, linha, área, polígonos, entre outros (Ferreira, 2007).

A especificação OpenGIS da OGC, define dois meios padrões para expressar objetos espaciais. A forma WKT (*Well-Know Text*), que representa os dados geográficos em modo de texto, facilitando a compreensão humana, e a WKB (*Well-Know Binary*), que representa os dados em binário. A especificação OpenGIS também exige que o objeto espacial a ser armazenado possua um identificador conhecido como SRID (*Spatial Referencing System Identifier*), que é um identificador único usado para definir uma projeção, sistema de coordenadas, entre outros (PostGIS, 2010). Seguindo tais padrões é possível abstrair a forma como os dados são armazenados e facilitar a comunicação entre SGBDs com suporte a manipulação de dados geográficos.

2.8.2. Mapeamento Objeto-Relacional

Mapeamento Objeto-Relacional, ou ORM (*Object-Relational Mapping*), consiste em uma abordagem que permite a construção de sistemas OO (Orientado a Objetos), que utilizam banco de dados relacionais. Utilizando-se

das técnicas de ORM, é possível mapear um modelo relacional em classes com atributos e referências.

Enquanto no paradigma Orientado a Objetos, o acesso aos objetos é feito apenas navegando-se entre seus relacionamentos, no paradigma relacional é necessária uma junção entre dados das tabelas. Os problemas que podem ocorrer ao comunicar entre esses dois paradigmas diferentes é frequentemente chamado de *O/R impedance mismatch*, ou incompatibilidade da impedância objeto-relacional (Ambler, 2010).

Bauer (2007) cita alguns dos problemas que podemos ter ao mapear um modelo relacional para um modelo Orientado a Objetos, entre eles estão os problemas de mapeamento e navegação entre as entidades, por exemplo, no modelo OO, enquanto temos objetos que referenciam uma lista de outros objetos, no modelo relacional temos novas tabelas para lidar com relacionamentos $n_x n$. Ambler (2010) também cita algumas diferenças entre modelos OO e modelos relacionais que devem ser lidadas com cuidado, como a normalização de dados *versus* normalização de classes, e modelagem de dados *versus* modelagem de classes. É interessante notar que, para lidar com associações, os bancos de dados relacionais se utilizam de chaves estrangeiras, que são campos que simplesmente apontam para um campo de outra tabela, enquanto que os objetos, através de um único atributo, conseguem referenciar coleções de outros objetos (Fowler, 2006).

Devido aos vários motivos já citados, a utilização de frameworks para mapear entidades em objetos e vice-versa tem se tornado cada vez mais frequente e com essa crescente utilização dos frameworks ORM, a *Sun Microsystems* lançou uma especificação chamada JPA (*Java Persistence API*), que faz parte de outra especificação maior chamada EJB (*Entity Java Beans*), e está contida na especificação JEE (*Java Enterprise Edition*).

2.8.3. JPA (*Java Persistence API*)

Os produtos que implementam a especificação JPA adotam um modelo de persistência não intrusivo, fazendo com que os objetos de domínio não precisem ter conhecimento a respeito dos meios de persistência. Eles não têm de implementar nenhum tipo de interface ou estender qualquer classe especial. O desenvolvedor pode manipular o objeto a ser persistido como qualquer outro objeto Java, mapeá-lo para um framework de persistência, e depois usar a API para persistir o objeto. Por serem objetos Java regulares, este modelo de persistência passou a ser conhecido como persistência POJO (*Plain Old Java Object*) (Keith, 2006).

Entre os frameworks ORM que implementam a JPA mais conhecidos estão o Hibernate, o TopLink Essentials e o OpenJPA, e uma das maiores vantagens em se utilizar frameworks ORM que implementam esta especificação está na portabilidade fornecida. Para uma grande corporação, a utilização de uma tecnologia padrão reduz substancialmente seu risco e permite a empresa trocar seu framework de persistência caso seja necessário, bastando apenas modificar alguns arquivos de configuração. Além disso, a utilização de uma especificação pode facilitar a comunicação entre toda a comunidade de desenvolvedores.

Desde o início, um dos principais objetivos da JPA foi garantir que ela seja fácil de usar e de entender. Apesar do domínio do problema não poder ser banalizado, a tecnologia que permite lidar com ele pode ser simples e intuitiva. Para que esta utilização se torne intuitiva, devemos entender alguns conceitos básicos como entidades, gerenciadores de entidades e comandos SQL.

O conceito de entidade não é novo e tem sido utilizado a mais tempo do que muitas linguagens de programação existir, inclusive mais tempo que o Java. A primeira citação foi feita por Peter Chen, em seu artigo publicado em 1976

onde abordava o assunto sobre Modelo Entidade-Relacionamento (MER). Chen descrevia entidades como elementos que possuem atributos e relacionamentos. Em Orientação a Objetos é como se adicionássemos comportamento a essa entidade e chamarmos esse resultado de objeto (Keith, 2006).

Para a JPA, alguns critérios devem ser satisfeitos para que um objeto se torne uma entidade. Keith (2006) cita quatro características essenciais para que esta conversão seja feita com sucesso. A primeira e mais básica de todas as características de uma entidade é a de que elas devem ser persistíveis, ou seja, seu estado pode ser armazenado em um meio de persistência de tal maneira que posteriormente essas entidades podem ser acessadas e recuperadas exatamente como estavam no momento de sua persistência. A segunda característica é a de que, como qualquer outro objeto Java, a entidade deve possuir um identificador único, que, para o caso de um banco de dados relacional, é representado pela chave primária. O terceiro critério está relacionado à transação, pois para que uma entidade seja persistida, excluída ou atualizada, tudo deve ser feito dentro de uma transação que ocorrerá de forma atômica, contemplando assim um dos conceitos básicos de banco de dados relacionais. E por último a granularidade, onde entidades podem ser melhor explicas pelo que elas não são. Entidades não são tipos primitivos e normalmente só fazem sentido quando têm um significado semântico para uma aplicação. Por exemplo, não faz sentido uma String ser uma entidade, pois a String sozinha não possui nenhum significado semântico para a aplicação. Uma String se encaixa melhor como o tipo do atributo de uma entidade, onde este atributo possui algum sentido para a aplicação.

Conforme citado a anteriormente, a especificação JPA adota um modelo de persistência não intrusivo, e o elemento mais importante que permite esse baixo acoplamento é o *EntityManager*. Praticamente todas as operações básicas realizadas em um banco de dados podem ser efetuadas através desses gerenciadores de entidades, por exemplo, abrir e fechar uma transação com o

banco, inserir, recuperar, atualizar ou excluir uma entidade, criar *queries* personalizadas, entre outras. Quando um *EntityManager* possuir a referência a uma entidade, obtida explicitamente ou através da leitura de um banco de dados, podemos dizer que este objeto ou entidade está sendo "gerenciada" pelo *EntityManager*. O conjunto de instâncias gerenciadas pelo *EntityManager* em um determinado tempo é chamado de Contexto de Persistência, e enquanto este Contexto de Persistência existir, não poderão haver instâncias Java com identificadores iguais nesse mesmo contexto. Por exemplo, se um usuário com id 27 estiver no Contexto de Persistência, então nenhum outro objeto com id 27 pode existir dentro desse mesmo Contexto de Persistência. Este *EntityManager* é gerado por uma classe chamada *EntityManagerFactory*, que é configurada por uma Unidade de Persistência. Na Unidade de Persistência ficam as configurações necessárias para o correto funcionamento do framework, como a URL da conexão, usuário, senha, *drivers*, caminho para as classes de entidades, entre outras. A Figura 2.8 exemplifica melhor como estão relacionados todos esses conceitos da especificação JPA.

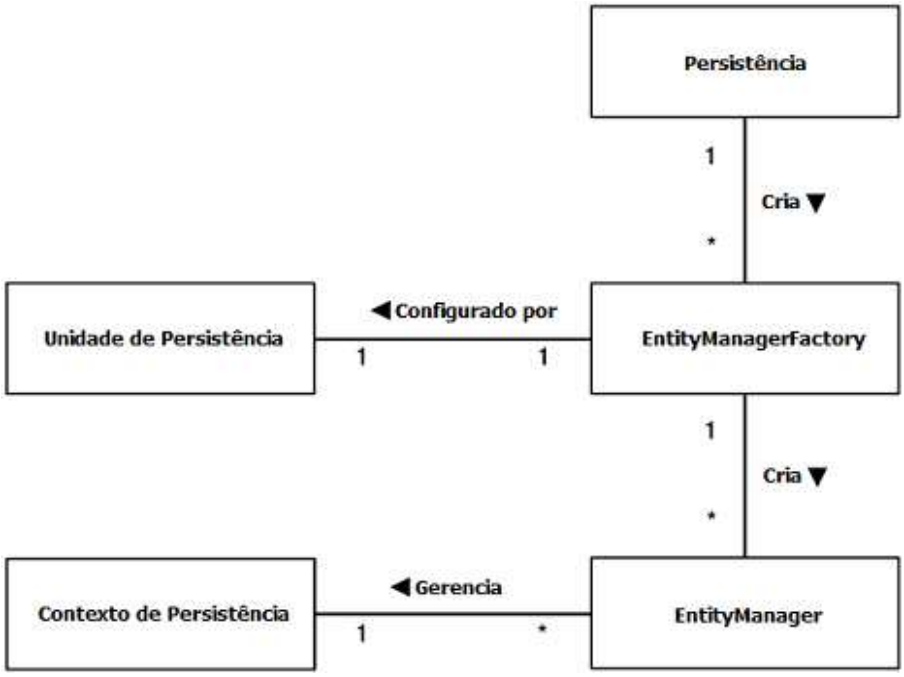

Figura 2.8: Relação entre os conceitos da JPA. Fonte: Adaptado de Keith (2006)

Para transformar uma classe Java regular (POJO) em uma entidade, a JPA oferece o recurso de Anotações. Normalmente criamos classes no estilo *JavaBean*, com um construtor sem argumentos, atributos privados e métodos de acesso *get* e *set*, e depois anotamos a classe com *@Entity* acima de sua declaração e *@Id* acima do atributo que será usado como identificador único no banco de dados. Essas são as anotações mínimas requeridas para mapear uma classe Java em uma entidade. O recurso de Anotações do JPA também oferece várias outras opções mais avançadas como, por exemplo, o *@Column*, que fica acima dos atributos que representarão as colunas da tabela e possuem informações como nome da coluna na tabela, tamanho do campo, se ele é único, se pode ser nulo, entre outras opções. Outra Anotação que também é muito

utilizada é a *@GeneratedValue*, que pode nos oferecer qual estratégia a ser empregada na geração de valores para as chaves-primárias.

Para lidar com o relacionamento entre as tabelas, o JPA oferece uma solução simples e extremamente intuitiva. Caso precisarmos mapear uma relação $n_x n$, por exemplo, que é o caso mais complexo de uma relação entre duas tabelas, podemos simplesmente adicionar a Anotação *@ManyToMany* a um atributo do tipo *Collection*, *List*, ou *Set*, e ao obtermos esse atributo, o framework já é capaz de preencher a entidade de acordo com a relação entre as tabelas no banco de dados. De modo similar, o JPA também oferece outras Anotações para mapear o relacionamento entre entidades, que são basicamente *@ManyToOne*, *@OneToMany* e *@OneToOne*.

As *queries* são implementadas em código no objeto *Query*, que é obtido através do *EntityManager*. Vários objetos *Query* podem ser gerados a partir do *EntityManager*, e são customizados de acordo com a necessidade da aplicação seguindo uma linguagem chamada *Java Persistence Query Language* (JPQL), que é similar ao SQL (*Structured Query Language*), porém que consulta por entidades e com enfoque mais Orientado a Objetos. Essas *queries* podem ser definidas estaticamente, também chamadas de *named query*, ou dinamicamente, permitindo que parâmetros possam ser adicionados posteriormente em tempo de execução. Obviamente as *queries* dinâmicas costumam ter um custo maior, pois elas não podem ser preparadas antes, mas sua sintaxe é tão simples quanto as *named queries*.

Uma das maiores vantagens da utilização de frameworks ORM é que podemos criar aplicações totalmente voltadas a Orientação a Objetos sem ter que nos preocupar em implementar meios de transformar as informações de um banco de dados relacional para objetos da aplicação. É importante notar que não precisamos nos preocupar com a implementação, mas nem por isso podemos deixar de lado as decisões de como vamos manipular essas informações do

banco. Tais frameworks oferecem apenas um meio mais simples e intuitivo de manipular essas informações, porém o desenvolvedor ainda deverá se preocupar com itens como *pool* de conexões, estratégia de geração de chaves-primárias, mapeamentos avançados, *queries*, controle de *lazy loading*, entre outros, e para lidar com esses itens mais complexos ou até mesmo com um simples mapeamento das relações entre as entidades, o desenvolvedor deverá ter certo nível de conhecimento em banco de dados.

2.8.4. *Google Maps*

O *Google Maps* é um aplicativo da *Google Inc.*, que tem como foco principal a disponibilização de serviços para pesquisa e visualização de mapas e imagens de satélite. Este ferramenta se diferencia das demais pela sua eficiência na busca e renderização das imagens obtidas por satélite, graças a um sistema de *tiles* que dividem o mapa e são carregados de maneira assíncrona e sob demanda (Pacheco, 2009).

O *Google Maps* compreende um conjunto de serviços web de geolocalização fornecidos pela *Google Inc*. Dentre esses serviços podemos calcular e visualizar rotas, escolher o meio de transporte (veículo, a pé e transporte público), localizar estabelecimentos, verificar a intensidade de tráfego em certos trechos, e até mesmo visualizar as ruas de algumas cidades como se estivesse em primeira pessoa (*Google Street View*). Similar ao *Google Maps*, temos o *Google Earth*, uma aplicação que roda no computador do usuário e, portanto, permite a utilização de recursos mais avançados para a visualização dos mapas, como por exemplo, a possibilidade de visualizar certos mapas em três dimensões.

Além do serviço web citados, a *Google Inc.* também fornece a API do *Google Maps*, que permite incorporar o *Google Maps* em suas próprias páginas

web com JavaScript. A API fornece diversos utilitários para manipular os mapas e adicionar conteúdo a eles através de uma variedade de serviços, permitindo a criação de aplicações geográficas robustas em um website (Google Maps API, 2010).

Várias ferramentas já incorporam o padrão utilizado pelo *Google Maps* em suas funcionalidades, facilitando a manipulação dessas informações. Por padrão, o *Google Earth* utiliza arquivos com extensão KML (*Keyhole Markup Language*) para manipular as informações geoespaciais, que nada mais é do que um arquivo XML (*Extensible Markup Language*), só que com *tags* próprias para expressar anotações geográficas (Pacheco, 2009). Manipulando tais informações utilizando arquivos nos padrões KML, podemos facilitar a comunicação entre aplicações, pois ambas reproduzem e interpretam a mesma linguagem, aumentando assim a portabilidade da aplicação.

Para trabalhar com a API do *Google Maps*, é necessário um pouco de conhecimento sobre a programação em *JavaScript* e os conceitos de programação orientada a objetos, além de, obviamente, ter uma certa familiaridade com o *Google Maps* do ponto de vista de um usuário.

JavaScript é a linguagem de *script* mais popular na Internet, e funciona em todos os principais navegadores, como Internet Explorer, Firefox, Chrome, Opera e Safari. Seu nome oficial é *ECMAScript* e sua função é a de trazer maior interatividade nas páginas Web, fazendo o uso de técnicas AJAX (*Asynchronous JavaScript and XML*) (W3Schools, 2010). Com isso podemos tornar as aplicações Web mais dinâmicas e mais leves, além de propiciar uma melhor comunicação entre a página Web e o usuário.

2.8.5. *Java Servlet*

Servlets são considerados componentes que estendem e aprimoram servidores web para oferecer conteúdo dinâmico a páginas web. Um *Servlet* é basicamente uma classe Java que processa requisições GET ou POST, por exemplo, e retorna um conteúdo HTML e XML criado dinamicamente para o cliente. Anteriormente, *scripts* CGI (*Common Gateway Interface*), eram utilizados para este mesmo propósito, porém devido a vários problemas, como falta de escalabilidade, a tecnologia *Servlet* tomou seu lugar na geração de conteúdo dinâmico, além de oferecer outros recursos como o uso de *Java Threads* (*Java Servlet Technology*, 2010).

Para o gerenciamento desses *Servlets*, é necessário o uso de um *Container*. Um exemplo é o Tomcat, um dos *Containers* mais utilizados atualmente. Quando uma aplicação de servidor web (*Apache*, por exemplo) recebe uma requisição para um *Servlet*, o servidor não envia esta requisição diretamente para o *Servlet*, e sim para o *Container* onde o *Servlet* está implantado. Este *Container* irá criar os objetos *request* e *response* e irá chamar o método *doGet()* ou *doPost()* do *Servlet* passando esses objetos de acordo com o tipo de requisição feita (Basham *et al.*, 2004). A Figura 2.9 exemplifica melhor o comportamento de uma requisição para um *Servlet*, e a resposta retornada.

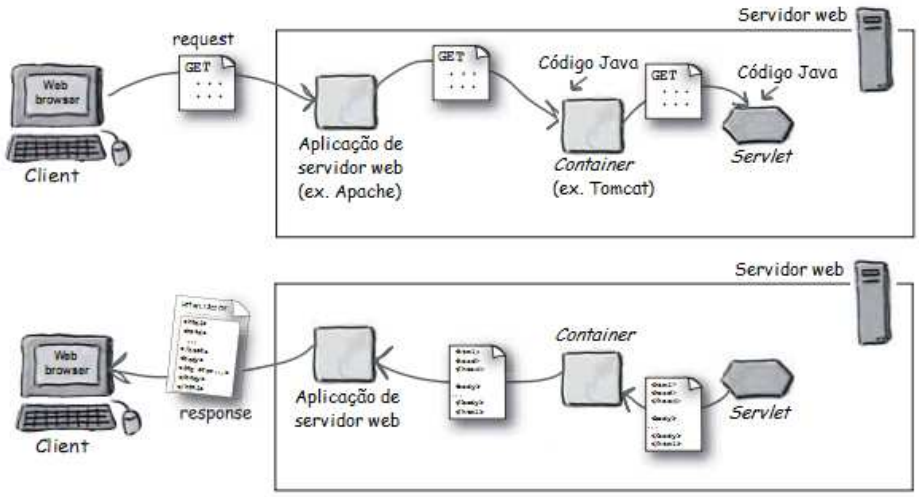

Figura 2.9: Comportamento de requisição e resposta a um *Servlet*. Fonte: Adaptado de (Basham *et al.*, 2004)

Dentre as vantagens de se utilizar *Containers* está a de gerenciamento do ciclo do vida de um *Servlet*, não sendo necessário o desenvolvedor se preocupar em instanciar e inicializar um Servlet, invocar seus métodos, ou tornar as instâncias visíveis ao *Garbage Collector*. Também não é necessário se preocupar em implementar métodos para a comunicação entre o servidor e o *Servlet*, pois o *Container* conhece todo o protocolo de comunicação do servidor. E outra grande vantagem é o suporte a *multithread*, pois a cada requisição o *Container* cria uma *Java Thread* para tratá-la, e assim que a requisição for respondida, esta *thread* morre (Figura 2.10). O *Servlet* fica vivo durante toda a aplicação (Basham *et al.*, 2004).

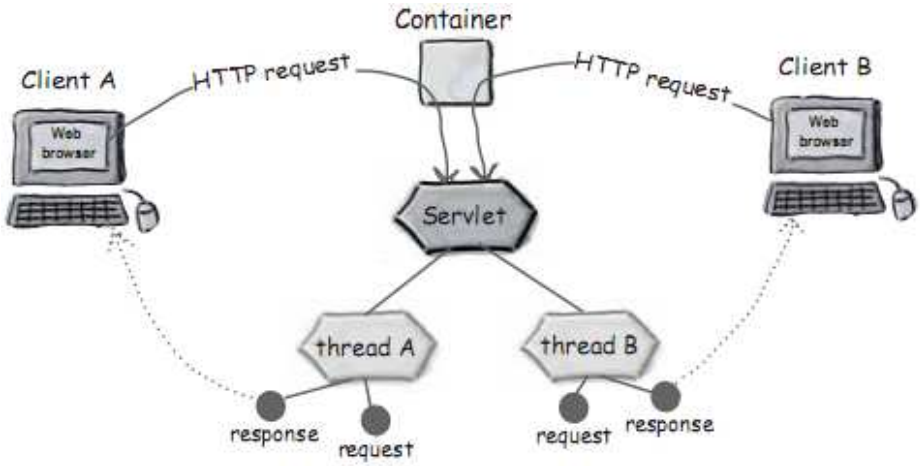

Figura 2.10: Cada requisição a um *Servlet* cria uma *Java Thread*. Fonte: (Basham *et al.*, 2004)

3. METODOLOGIA

Conforme Gil (1999), a ciência tem como meta principal abordar a veracidade dos fatos, e para que um conhecimento seja considerado como científico, devemos utilizar métodos já aceitos e empregados pela comunidade científica. Portanto, para atingirmos o conhecimento, é imprescindível a elaboração de métodos científicos baseados em um conjunto de procedimentos intelectuais e técnicos previamente adotados.

3.1. Classificação da pesquisa

Devido a sua natureza prática, este trabalho se enquadra como pesquisa aplicada. Através de estudos e experimentações, visa-se gerar conhecimentos para soluções de problemas relacionados à área de transportes, e principalmente para aplicações que utilizam posicionamento geográfico.

Em relação à forma de abordagem do problema, foi realizada uma pesquisa quantitativa. A pesquisa quantitativa requer o uso de recursos e técnicas estatísticas para transformar os dados coletados em informações relevantes para o problema (Silva e Menezes, 2001). Esta é a abordagem ideal ao problema, já que foi necessário lidar com dados obtidos através de GPS e GPRS, e transformando-os em informações úteis para o sistema.

Em relação aos objetivos da pesquisa, Moresi (2003), cita a pesquisa metodológica como um estudo que visa desenvolver ferramentas para a captação ou manipulação de dados relativos à nossa realidade. Portanto, ela está associada a caminhos, formas, métodos, e procedimentos para que seja alcançado um objetivo proposto. Um exemplo seria a construção de um instrumento para analisar o grau de descentralização decisória de uma organização. Como este projeto propõe a criação de um sistema com o objetivo geral de facilitar a vida

do usuário e das empresas de transporte público, esta seria a finalidade de pesquisa que mais se encaixa no contexto.

Quanto aos procedimentos técnicos, ou meios de investigação, o trabalho se adéqua melhor à pesquisa experimental, por seu caráter exato e por envolver muitas variáveis que podem ser medidas e quantificadas. Segundo Gil (1991), as vantagens da pesquisa experimental são praticamente indiscutíveis, pois foi a partir dela que foi possível obter os maiores avanços nas ciências físicas e biológicas. Ela nos permite obter uma maior nitidez, exatidão, e objetividade nos resultados, porém, exige um controle extremo das variáveis envolvidas.

3.2. Atividades da pesquisa

Neste tópico são abordadas as atividades que foram planejadas para esta pesquisa. A ordem das atividades listadas abaixo não reflete a ordem em que foram executadas durante o andamento do projeto. Em alguns casos tais atividades foram executadas em paralelo.

a) *Software* e linguagens de programação: Em todos os momentos deste projeto, houve a necessidade de analisar os vários *software* disponíveis no mercado e diversas linguagens de programação, principalmente voltadas para a Web, para a escolha das ferramentas que melhor se enquadram em cada situação. Foi priorizada a escolha de tecnologias baseadas em *software* livres, com a finalidade de suavizar os custos para implementação, já que a aquisição de módulos eletrônicos para coletar os dados via GPS e retransmiti-los por GPRS pode ser muito dispendiosa;

b) Arquitetura do sistema: No projeto e desenvolvimento do sistema, foram utilizadas estratégias e padrões para uma modelagem eficiente e mais orientada a objetos possível;

c) Processamento de informações: Foram criados métodos para processar as informações coletadas e persisti-las no banco de dados de maneira eficiente e intuitiva;

d) Visualização dos dados: Foram aplicadas técnicas para a transformação dos dados obtidos no Módulo de Coleta em informações relevantes para os usuários e fornecedores do serviço.

4. RESULTADOS

Neste capítulo são abordadas as decisões tomadas na elaboração de toda a aplicação. Tais decisões incluem decisões de banco de dados, padrão arquitetural, entre outros.

4.1. Arquitetura do sistema

O sistema está dividido em módulos de maneira a promover uma maior manutenibilidade, permitir um baixo acoplamento entre as partes, e facilitar sua compreensão. A Figura 4.1 exibe um esboço de como os módulos do sistema estão organizados.

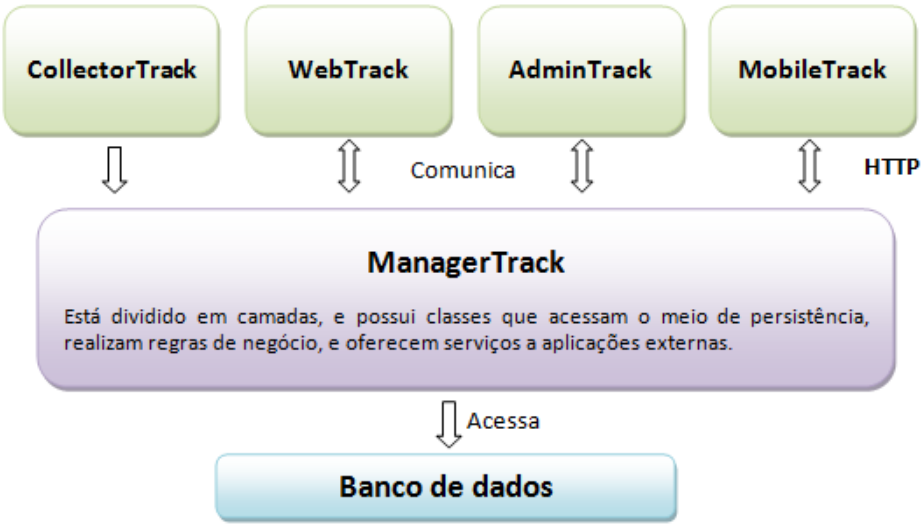

Figura 4.1: Arquitetura de comunicação entre os módulos do sistema.

Os módulos estão organizados em camadas que possuem contato somente com seus vizinhos e nunca com módulos dentro de sua própria camada.

O banco de dados está na camada mais baixa e representa o local onde as informações são persistidas e acessadas. A camada mais alta, com exceção do *CollectorTrack*, contém os módulos "usuários" do sistema, ou seja, os módulos que usufruem dos serviços oferecidos pelo *ManagerTrack.* Esses módulos "usuários" são o *WebTrack*, que representa a interface entre o sistema e o usuário final que acessa o serviço pelo navegador, o *AdminTrack*, que representa a interface onde os administradores podem modificar informações do sistema como cadastro de rotas, veículos e motoristas, e o *MobileTrack*, que possui as mesmas finalidades do *WebTrack*, como a visualização de rotas, porém em um dispositivo móvel. É interessante notar que a comunicação entre os módulos *CollectorTrack* e *ManagerTrack* é de apenas uma via, já que o *CollectorTrack* somente envia informações sobre a posição do ônibus para o *ManagerTrack* através de parâmetros passados por GET em uma URL. Porém nada impede que futuramente esta comunicação seja realizada em duas vias, por exemplo, caso seja necessário efetuar autenticações entre o módulo e o sistema, ou caso este módulo esteja integrado com um display onde o motorista poderá ter acesso a informações como paradas mais próximas ou quanto tempo falta para completar uma rota, entre outras. Os módulos dessa camada superior não podem se comunicar diretamente, evitando assim a descentralização das funcionalidades oferecidas, e aumentando a coerência do sistema como um todo. Nesta camada ainda é possível adicionar outros módulos com funções variadas, como por exemplo, um *WebService* que fornece serviços implementados pelo *ManagerTrack* para outras aplicações.

Na camada intermediária temos o *ManagerTrack*, que internamente também está dividido em camadas, e sua função principal é a de persistir informações coletadas através de módulos, realizar todas as regras de negócio necessárias para o bom funcionamento do sistema e oferecer serviços para os módulos acoplados a ele. O *ManagerTrack* está organizado para persistir os

dados do modo mais abstrato possível do meio de persistência utilizado. As decisões de como este módulo foi organizado estão descritos na próxima seção.

4.2. Decisões de projeto e diagramas do sistema

Neste tópico o sistema será melhor detalhado com a ajuda de diagramas e explicações a respeito de decisões tomadas no projeto para torná-lo mais robusto e manutenível.

A Figura 4.2 representa uma mistura de diagrama de classe com diagrama de pacotes. Desta maneira é possível visualizar o núcleo do sistema de maneira abrangente, e sem a necessidade de entrar em detalhes redundantes.

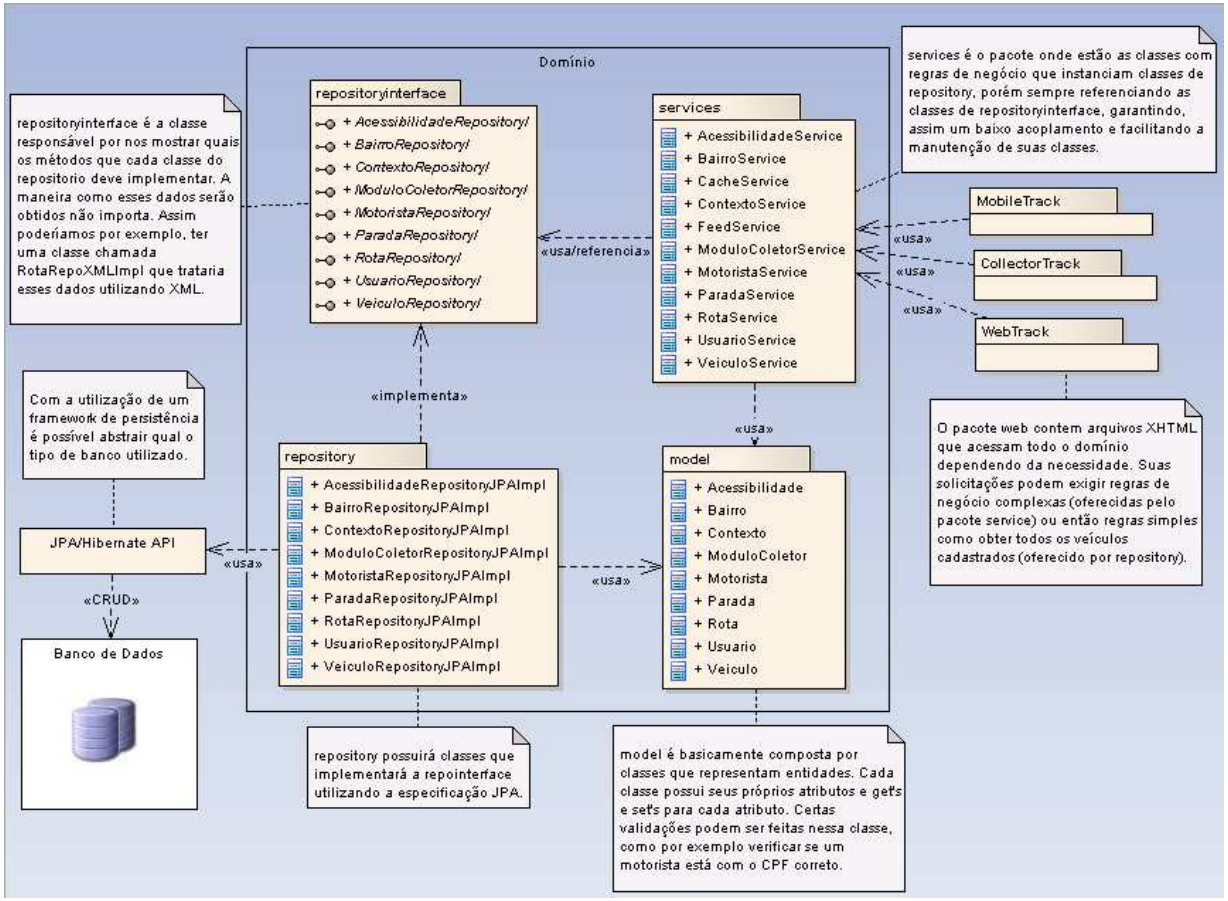

Figura 4.2: Diagrama de classe/pacotes do sistema.

Este diagrama representa uma boa parte do módulo *ManagerTrack*, que pode ser considerado o núcleo do sistema, onde ficam concentradas todas as regras de negócio, e onde são oferecidos serviços de acordo com a necessidade da aplicação. Além disso, é através desse módulo que acessamos o meio de persistência. Os pacotes que compõem a parte mais importante do sistema são o *model*, o *repositoryinterface*, e o *services*. Com o conhecimento destes pacotes, qualquer pessoa poderia entender o domínio da aplicação, ou seja, poderia entender o que o sistema oferece (*services*), utilizando quais modelos, ou conceitos, físicos ou abstratos (*model*), e de que maneira esses modelos são armazenados (*repositoryinterface*).

No pacote *model* estão inclusas as classes de entidades do sistema. Essas entidades são representações das nossas tabelas no banco de dados, onde cada atributo corresponde a uma coluna da tabela e cada um desses atributos possui Anotações indicando, por exemplo, o nome da coluna no banco, se esse valor pode ser nulo, ou até mesmo o modo de geração da chave primária no banco. Além disso, os atributos são acessados através de métodos *get* e *set*, onde pode-se até fazer algumas validações, por exemplo, não permitir que seja inserido um número negativo para o ano do veículo, ou até mesmo validações mais específicas como não permitir que este ano seja anterior a 1980. Desse modo, nossas entidades ficam restringidas conforme a necessidade sem alterar regras de negócio, que devem se preocupar mais com as funcionalidades do sistema do que com validações já pré-estabelecidas. O *model* possui tanto entidades concretas no mundo real, como veículo, quanto entidades mais abstratas, como rota.

Para gerenciar todo o meio de persistência da aplicação, foi criado o pacote *repositoryinterface*, que como o próprio nome já diz, é um pacote composto por *interfaces* que ditam ao nosso sistema quais funcionalidades o sistema de armazenamento de dados deve oferecer para a aplicação. Essas

interfaces possuem assinaturas de métodos que determinam a sua implementação quais funções ela deve implementar. Por exemplo, a *interface VeículoRepositoryI* determina que quem implementá-la, deve escrever métodos para inserção, exclusão e alteração de veículos, além de métodos para retornar todos os veículos, retornar veículos de acordo com sua rota, entre outros. Com a utilização dessas *interfaces* é possível garantir que nossas entidades, ou modelos, possuam meios de serem armazenadas, independente de como ocorrerá esse armazenamento. A idéia é obter um repositório de entidades, onde a única coisa que importa são as operações que podem existir entre a aplicação e o meio de persistência e não como essas operações serão realizadas. Caso queira-se persistir todos esses dados em arquivos XML, basta criar classes que implementam essas interfaces e o nosso sistema continuará funcionando sem a necessidade de alterar qualquer regra de negócio ou linha de código no domínio da aplicação, pois todas as classes do domínio acessarão o meio de persistência através dessas *interfaces*. Para esta aplicação foi implementado essas *interfaces* utilizando JPA, que permite mapear o modelo entidade relacional do banco de dados para o modelo de objetos utilizado na aplicação e representados no pacote *model*. Além da abstração fornecida pelo *repositoryinterface* de como essas entidades são persistidas (arquivos de texto, XML, banco de dados, etc.), com a utilização de JPA também é possível abstrair qual o tipo de banco de dados relacional utilizado (PostgreSQL, MySQL, Oracle etc.). Com a utilização do Hibernate, framework que implementa a especificação JPA, o mapeamento Objeto-Relacional se torna transparente, eliminando assim a necessidade de se preocupar com detalhes de implementação de código.

O pacote *service* possui as classes responsáveis por realizar todas as regras de negócio da aplicação. Essas classes sempre acessarão informações do meio de persistência utilizando as *interfaces* de *repositoryinterface*, garantindo assim um baixo acoplamento e facilitando a manutenção de suas classes. Seu

objetivo principal é o de oferecer serviços que serão utilizados por módulos acoplados a ele, por exemplo, a classe *RotaService* possui o método *findRotasAsKML(),* que retorna todas as rotas cadastradas no sistema no formato KML, que é um tipo de XML muito utilizado para exibir informações em mapas geográficos. Esses serviços também podem ser utilizados internamente, por exemplo, a classe *ContextoService* possui o método *persistContext*, que recebe como parâmetro a latitude, a longitude, a velocidade, e o id do módulo coletor que está acoplado no veículo, e através de algumas regras de negócio persiste essa informação na tabela *Contexto* para que seja analisada posteriormente. Quem utiliza esse serviço é o *servlet CollectorServlet* que é acessado pelo módulo *CollectorTrack*. Este módulo envia para o *servlet* algumas informações coletadas pelo GPS, e o *servlet* por sua vez, solicita ao *ContextoService* a inclusão dessas informações no sistema. As classes no pacote *service* foram criadas para diversas finalidades, pois inclui as regras de negócio do sistema. Essas classes podem ser acessadas por vários outros módulos como o *WebTrack* e o *MobileTrack*. Poderíamos por exemplo, criar um módulo de *Web Services* que acessa as informações oferecidas pelas classes de *service* e disponibiliza essas informações para aplicações externas independentemente da linguagem utilizada por tais aplicações.

O sistema possui três *Servlets* para atender suas necessidades. O *CollectorServlet* tem por finalidade atender o módulo *Collector*, sendo que esse módulo é composto basicamente pelos módulos GPS/GPRS que estão instalados em cada veículo. Futuramente o sistema poderia acoplar módulos mais sofisticados e com funcionalidades extras que exijam uma relação de requisição e resposta com o *ManagerTrack*. Por exemplo, podem-se instalar módulos com *display* nos veículos para permitir ao motorista alterar a sua rota ou exibir informações do sistema. Os outros dois *Servlets*, *WebServlet* e *MobileServlet* são utilizados para atender respectivamente os módulo, *WebTrack* e *MobileTrack*.

Essa separação foi feita apenas para manter a modularidade do sistema, porém suas funções são praticamente as mesmas. Os módulos clientes, tanto *MobileTrack* quanto *WebTrack*, solicitam informações ao *ManagerTrack* através de GET o qual possui o parâmetro *action* que nos diz qual a informação desejada. Por exemplo, ao acessar a URL com final /ManagerTrack/WebServlet?action=get_stops, estaremos solicitando ao *WebServlet* todos os pontos de paradas cadastrado no sistema. Essa requisição irá retornar um arquivo KML contendo todas as coordenadas de acordo com o padrão KML, com exceção de uma ou outra *tag* a mais utilizada para aprimorar a interação entre a aplicação cliente e a aplicação no servidor, como por exemplo, a *tag* id, com a identificação do ponto de parada do banco de dados. Outro motivo para divisão entre dois *Servlets* se deu pelo fato do módulo *MobileTrack* implementado estar utilizando um formato diferente para interpretar os dados (um XML feito pelo próprio autor do módulo), e como não há nada que poderia ser comum entre esses, a criação de um *Servlet* separado para o *MobileTrack* foi a alternativa que melhor se adequou a situação.

No decorrer do planejamento e desenvolvimento do sistema, foi necessário levar em consideração a quantidade de dados trafegados entre os módulos, pois, o sistema deverá suportar uma grande quantidade de informações sendo recebidas e persistidas. Para cada módulo coletor será coletado informações como latitude e longitude via GPS e retransmitidas via GPRS o tempo todo para o módulo principal, que persistirá todas essas informações. Além disso, todas essas informações coletadas deverão ser fornecidas para os módulos de visualização sem atrasos significativos. Levando em consideração que temos vários usuários acessando essas informações praticamente em tempo real, o sistema deverá suportar todas essas conexões simultâneas enviando e solicitando dados a todo tempo. Um mal planejamento do sistema pode deixar toda a aplicação indisponível.

Para suportar esse acesso concorrente de usuários e diminuir a carga de informações trafegadas pela aplicação, inicialmente foi planejada uma estratégia parecida com a técnica de *caching* e, posteriormente outra técnica, que chamada de *feed*, foi analisada e escolhida para ser implementada e aplicada no sistema. Porém nada impede que ambos os casos sejam estudados e analisados cuidadosamente para que se chegue a uma melhor conclusão. Nos próximos dois tópicos são demonstradas as duas técnicas, seus diagramas de sequência, e logo após, algumas observações relativas a elas e ao sistema como um todo.

4.2.1. *Caching* em *Servlet*

Caching consiste em manter informações utilizadas com frequência em uma memória de acesso mais rápido para que seja utilizada posteriormente pelo solicitante da informação, ou por outros solicitantes. Como o *Servlet* possui a característica de ficar instanciado durante toda a aplicação, ou seja, enquanto a aplicação estiver rodando, a mesma instância do *Servlet* estará disponível, foram adicionados atributos estáticos no *Servlet*. Esses atributos representam o *cache* do sistema. Cada vez que o usuário solicitar uma informação para o sistema, como por exemplo, a lista de rotas, o *Servlet* que receber a solicitação irá retornar este atributo contendo todas as rotas do sistema, nesse caso, um atributo estático do tipo String que chamamos de *rotaCache*. Dessa maneira, o sistema evita efetuar operações no banco de dados para cada solicitação de cada usuário e, além disso, como todos os usuários acessam o mesmo *Servlet*, para cada solicitação, este *Servlet* irá instanciar um *Thread* para lidar com cada solicitação, permitindo um acesso paralelo a essas informações do *cache*.

Quanto à atualização dessas informações no *cache*, deve-se pensar de maneira isolada à solicitação das informações *cache* em si, pois a atualização deste não depende da solicitação e, do mesmo modo, a solicitação dessas

informações não depende diretamente do *cache* estar atualizado ou não. Sabe-se que o *cache* só precisará ser atualizado caso alguma informação do banco seja alterada, excluída ou inserida, portanto, como o atributo *cache* no *Servlet* é estático, assim que algo for inserido, atualizado ou excluído com sucesso no banco de dados, um método da classe *CacheService* será chamado, e este método atualizará tais informações no atributo *cache* do *Servlet*. Não foi preciso preocupar em fazer sincronizações na leitura e escrita do atributo, pois o tipo String em Java é imutável, ou seja, quando dizemos que uma String com valor "abc" recebe o valor "def", o que acontece na verdade é que outra String com valor "def" é instanciada, e a referência que antes apontava para a String "abc" a partir de agora passará a apontar para a String "def".

Para exemplificar melhor a relação entre o *CollectorTrack* e o *ManagerTrack*, e o que o sistema faz ao receber a posição de um veículo, a Figura 4.3 exibe um diagrama de sequência do momento em que o Módulo Coletor envia as informações de id, latitude, longitude e velocidade, até o momento em que essas informações são persistidas e armazenadas no *cache*, tornando-as disponíveis aos módulos clientes. É importante notar que a instância do *CollectorServlet* permanece viva durante toda a aplicação.

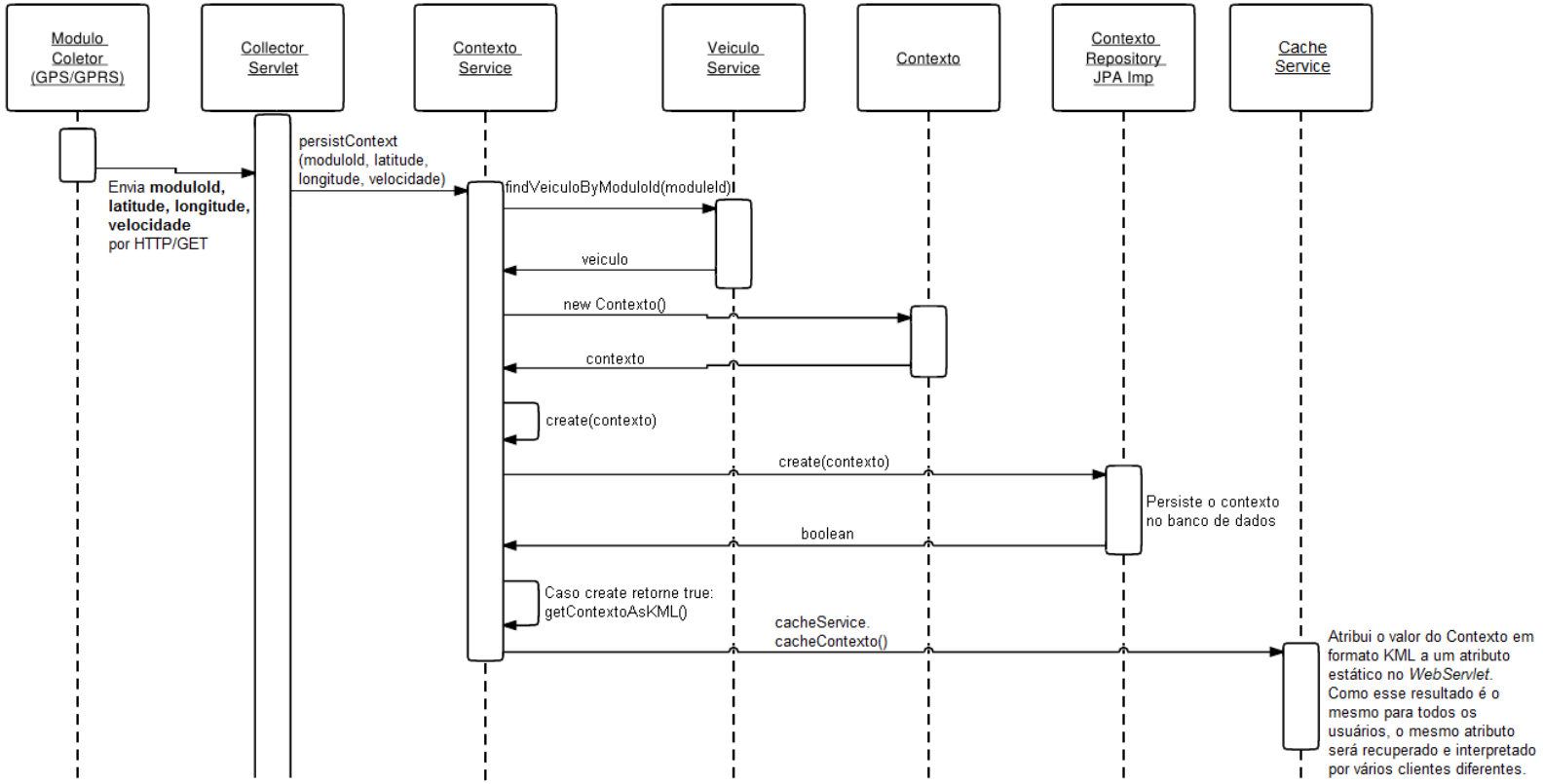

Figura 4.3: Diagrama de sequência para a técnica de *cache*, exibindo desde o Módulo Coletor até o momento da persistência e disponibilidade da informação coletada.

4.2.2. Arquivos de *feed*

A idéia de *feed* vem de *web feed*, ou *news feed*, que é um formato de dados utilizado para fornecer aos usuários conteúdo que é atualizado com freqüência, como por exemplo, *blogs* e sites de notícia. Basicamente temos uma aplicação, no caso a *ManagerTrack*, que, assim que ocorrer algo no banco de dados, como alteração, exclusão, ou inserção, um método é chamado para atualizar um arquivo KML hospedado no servidor que possui informações a respeito do dado que foi atualizado. Por exemplo, caso uma rota seja inserida no nosso banco de dados, o arquivo *rotaFeed*, localizado no nosso servidor, será atualizado com essa nova informação. Para que essa operação ocorra com segurança, foram implementados métodos para que nenhum arquivo seja lido enquanto houver escrita.

A Figura 4.4 exemplifica, com um diagrama de sequência, qual o comportamento do sistema utilizando essa técnica.

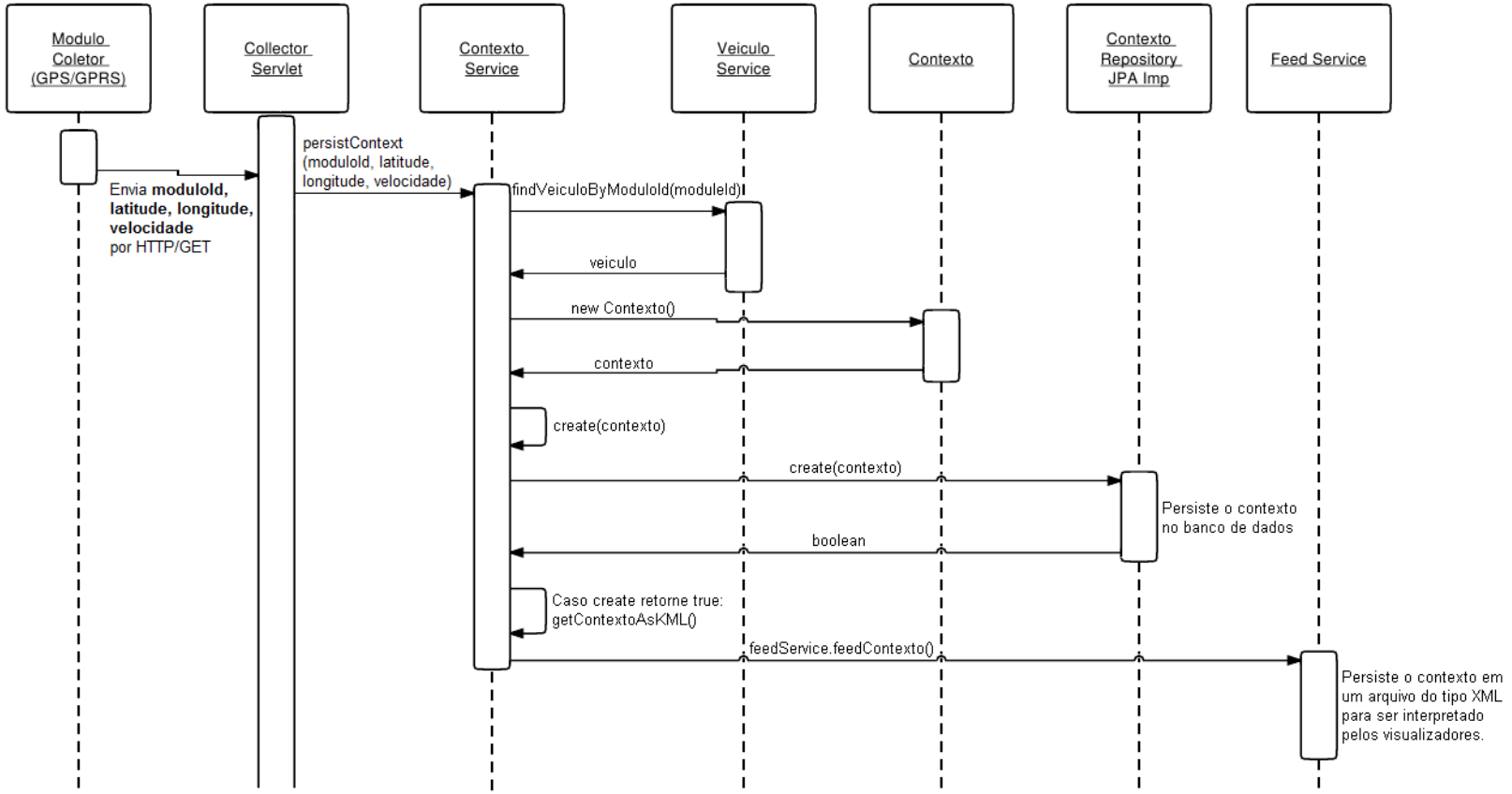

Figura 4.4: Diagrama de sequência para a técnica de *feed*, exibindo desde o Módulo Coletor até o momento da persistência e disponibilidade da informação coletada.

Utilizando essas técnicas citadas anteriormente, deixamos o sistema mais preparado para suportar acessos simultâneos, pois em ambos os casos, o próprio *Servlet* da aplicação cuida para que a cada requisição uma *Thread* seja criada, fazendo com que os acessos dos usuários possam ocorrem simultaneamente. Também é possível evitar sobrecarga de acessos ao banco de dados do sistema já que, no caso do *feed*, todos os usuários acessariam somente um arquivo criado e alterado conforme a necessidade, e no caso do *cache*, todas essas informações estariam disponíveis na memória do servidor. E a partir dessa última afirmação já é possível citar um diferencial entre as duas técnicas, pois o acesso ao arquivo exige que seja feita uma operação de E/S (Entrada/Saída) e, portanto, poderíamos estar sobrecarregando o acesso ao disco rígido do servidor, além de que esse acesso em disco poderia atrasar as requisições. Já em certos casos, talvez a disponibilidade de memória no servidor seja insuficiente para suprir a quantidade de memória que será exigida para que esses arquivos fiquem na memória, ou até mesmo por questões de segurança, seja mais interessante manter esses dados em arquivo. Portanto, cada caso exigirá uma solução mais adequada.

É importante ressaltar que, conforme pode ser notado nos diagramas de sequência, independente da técnica aplicada, o sistema permanece praticamente idêntico, sendo que, caso haja necessidade de alguma mudança, poucas partes do sistema precisarão ser modificadas. Isto se deve a uma arquitetura mais modularizada e voltada à orientação a objetos.

4.3. Banco de dados geográfico - PostgreSQL/PostGIS

Algumas decisões foram tomadas ao se utilizar o banco de dados PostgreSQL e sua extensão PostGIS. Essas decisões envolvem a arquitetura do banco de dados, como tabelas e relacionamentos, as estratégias para tratar esses

dados, e como lidar com os dados espaciais utilizando a extensão do PostgreSQL, PostGIS.

A Figura 4.5 representa o diagrama relacional do sistema. Através de uma rápida análise do diagrama fica claro quais as entidades utilizadas pelo sistema e como elas se relacionam. Por exemplo, a relação entre Rota e Veículo é $m_x n$, pois uma Rota pode possuir mais de um Veículo e um Veículo pode estar cadastrado em mais de uma Rota, e por isso houve a necessidade de criar uma tabela intermediária (*veiculo_tem_rota*) para lidar com esse relacionamento. Para saber qual a Rota atual que o Veículo está percorrendo, existe mais um campo na tabela Veículo chamado *rota_atual_id*, que identifica essa Rota, e como esta relação é $1_x n$, ou seja, cada Veículo só possui até uma Rota atual, a tabela Veículo recebeu esse atributo que é uma chave estrangeira e aponta para o campo de Id da Rota que está sendo percorrida pelo Veículo no momento.

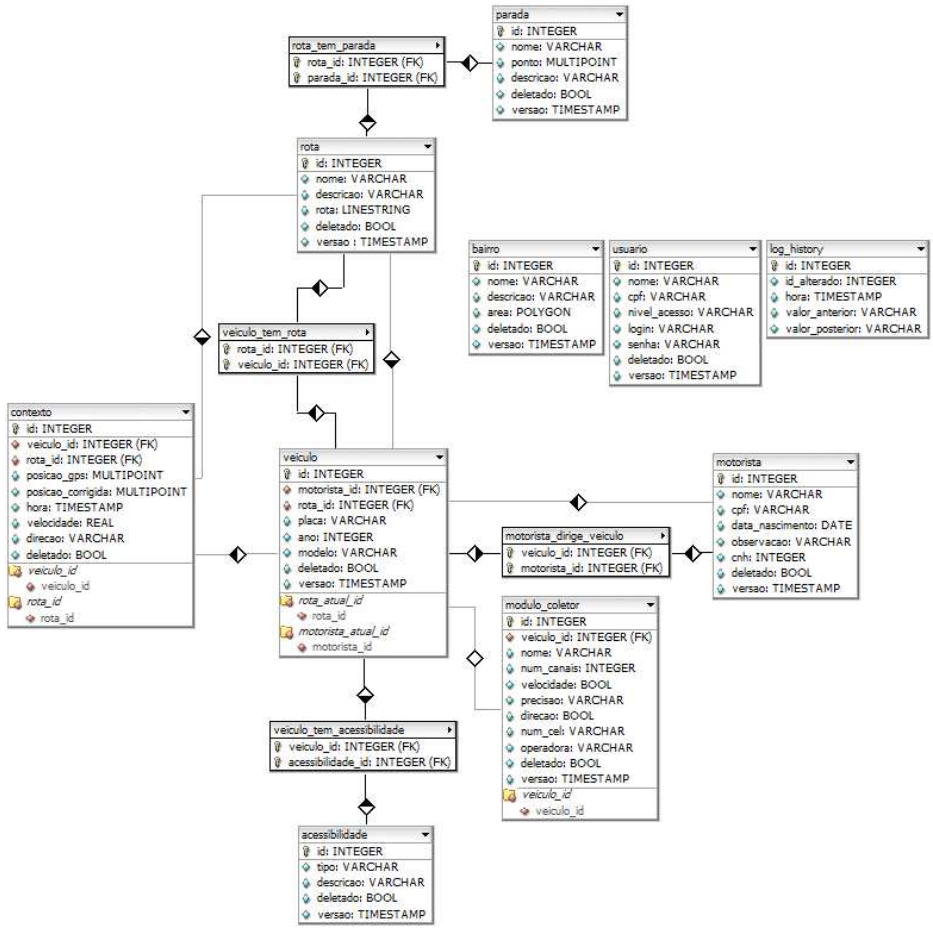

Figura 4.5: Diagrama Relacional.

Apesar de não possuírem relacionamentos, as tabelas de Bairro e de Usuário podem ser úteis ao sistema, onde a tabela de Bairro pode ser utilizada para melhor organizar a estrutura do serviço de transporte, exibindo informações sobre as áreas percorridas pelo transporte, e a tabela Usuário guarda informações a respeito do usuário do sistema, como login, senha, e nível de acesso do usuário no sistema. Já a tabela Contexto possui relacionamento com outras duas tabelas, Rota e Veículo, onde esses campos não devem ser nulos. A tabela Contexto representa as informações recebidas do Módulo Coletor, como a posição

geográfica do veículo no momento da coleta, o identificador do veículo e em qual rota este veículo se encontra.

Para persistir as informações geográficas no banco, como linhas, pontos e polígonos, utilizamos a extensão do PostgreSQL, PostGIS. As tabelas Parada, Rota e Bairro utilizam dados espaciais para representar certos atributos, como por exemplo, o atributo Ponto da tabela Parada, que representa a posição geográfica dos ponto de ônibus. Esse ponto de parada é do tipo *Point*, que nada mais é do que a latitude e longitude daquele ponto no mapa. De forma similar temos a *LineString* para marcar as rotas cadastradas no sistema, e *Polygon* que representa a área de um bairro. A única condição para a manipulação desses dados é sempre informarmos o SRID, explicado anteriormente, e que, por ser o mais utilizado, definimos seu valor padrão como 4326.

Para acessar essas informações geográficas de maneira transparente na aplicação, foi utilizada uma extensão do *Hibernate* chamada *Hibernate Spatial*, e para manipular esses dados em Java, uma biblioteca da *Vivid Solutions* chamada *JTS Topology Suit*. Assim, utilizando-se apenas da *Annotation @Type* do *Hibernate*, é possível declarar uma rota como um objeto do tipo *LineString* e, além de não precisar cuidar de qualquer tipo de conversão, podemos utilizar métodos apropriados para dados geográficos, como interseção entre duas *LineString* ou distância entre a *LineString* e outro objeto geográfico, entre outros.

Algumas estratégias foram adotadas para manter o banco de dados mais robusto e consistente. Uma estratégia muito adotada em bancos de dados que precisam guardar informações importantes, é chamada de “exclusão lógica”. A exclusão lógica consiste em adicionar uma nova coluna de valor booleano, verdadeiro ou falso, a cada tabela do banco de dados, assim, ao invés de excluir um item de uma tabela, este item é marcado como excluído, porém continua no banco de dados. Utilizando essa técnica, é possível ver quais itens foram

excluídos da base e evitar a perda completa das informações no caso de uma exclusão equivocada. Com algumas alterações é possível até manter o histórico de exclusões e alterações na base de dados. Essas informações relativas ao histórico de exclusões e alterações podem ser muito úteis para a segurança das informações. Por exemplo, caso o sistema sofresse algum tipo de invasão no banco de dados ou até mesmo tenha suas informações alteradas ou excluídas equivocadamente, com o histórico de alterações seria possível reverter a situação até o ponto onde o problema se iniciou, pois as informações continuam no banco e cada alteração ou exclusão possui uma ordem cronológica que auxilia no rastreamento de tais alterações.

Para implementar a exclusão lógica, todas as tabelas receberam uma coluna de tipo booleano chamada *deletado*. Ao excluir um item, a coluna desse item receberá o valor *true*, e para que o banco não perca sua consistência, esse item que será excluído não poderá referenciar itens que não estão marcados como excluídos. De maneira similar, todos os itens que referenciam esse registro também serão marcados como excluídos, que é o que chamamos de *on delete cascate*.

Em relação à alteração dos itens, para que seja mantido um histórico de alterações e exclusões ocorridas na tabela, foi criada uma tabela chamada *log_history*, onde serão armazenadas informações a respeito de todas as exclusões e alterações realizadas no banco, com data e hora da alteração ou exclusão. Outra estratégia utilizada é a adição do campo *versão* em todas as tabelas que podem ser editadas, ou seja, todas as tabelas do modelo com exceção das tabelas *contexto* e *log_history*, que não devem ser editadas por motivos de segurança. O campo *versão* é do tipo *timestamp*, e a cada alteração ou até mesmo a exclusão de um item, este campo será atualizado com a data e hora atual do sistema, assim é possível saber o momento da alteração ou exclusão e principalmente poderá ser usado para fins de sincronização entre outros bancos.

4.4. Mapeamento Objeto-Relacional utilizando *Hibernate/JPA*

É importante destacar alguns detalhes na implementação do mapeamento objeto-relacional realizado utilizando o framework *Hibernate* e a especificação JPA. Todas as classes do pacote *model* foram marcadas com a *Annotation @Entity* logo acima da declaração da classe, por representarem todas as nossas entidades do banco de dados. Também foi utilizada a *Annotation @SequenceGenerator* que indica a sequência utilizada para gerar o número das chaves primárias da tabela e a *Annotation @GeneratedValue* que faz com que o *Hibernate* gere os valores dessas chaves primárias automaticamente conforme a sequência informada anteriormente. Algumas colunas receberam o parâmetro *nullable=false* que informa ao framework que tal coluna não pode receber valores nulos, assim, antes mesmo de tentar persistir uma entidade no banco de dados, o framework pode, no momento da inserção, verificar se a entidade não possui nenhuma irregularidade e interceder com mensagens de erro caso necessário.

Para casos de chaves estrangeiras e relações $m_x n$, algumas *Annotations* apropriadas a cada caso foram utilizadas, por exemplo, na relação $m_x n$ entre Rota e Parada cada lado da relação possui uma coleção de objetos do outro lado da relação. Do lado da Rota a relação que ficou foi a seguinte:

```
@ManyToMany(fetch=FetchType.LAZY)
@JoinTable(name="rota_tem_parada",
    joinColumns=@JoinColumn(name="rota_id", nullable=false),
    inverseJoinColumns=@JoinColumn(name="parada_id", nullable=false))
private Collection<Parada> paradas;
```

A partir de um simples mapeamento, muitas conclusões podem ser tiradas. Cada rota possui uma coleção de paradas e a *Annotation @JoinTable*

nos informa que o nome da tabela que mapeia esta relação no banco de dados relacional é a tabela *rota_tem_parada*, suas colunas são *rota_id* e *parada_id*, e essas não podem ser nulas. Dentro da *Annotation @ManyToMany* o atributo *fetch* está como *LAZY*, informando ao *Hibernate* que, ao carregar qualquer objeto Rota do banco de dados, a coleção de objetos Parada associada a esta rota não deverá ser carregada. Tal decisão, que pode ser aplicada a qualquer coluna da entidade, alivia o carregamento excessivo de objetos do banco de dados, que possivelmente não serão utilizados pela aplicação, ficando a cargo do desenvolvedor decidir o que será melhor para um bom desempenho na aplicação. Um caso em que seria interessante o carregamento "prematuro" das informações no banco de dados é o da relação entre o veículo e a rota atual deste veículo, mapeados da seguinte maneira:

```
@ManyToOne(fetch=FetchType.EAGER)
@JoinColumn(name="rota_atual_id")
private Rota rotaAtual;
```

A *Annotation @ManyToOne* neste caso nos mostra que uma rota pode pertencer a vários veículos diferentes, porém cada veículo tem a sua rota atual, e a coluna que faz o *join* entre o veículo e a rota atual é a *rota_atual_id*.

Para entidades como Contexto, que devem possuir um campo espacial, a *Annotation @Type* foi utilizada. Essa *Annotation* pertence somente ao *Hibernate* e não está na especificação do JPA. Com ela declaramos campos espaciais como um ponto ou uma linha utilizando a API *Hibernate Spatial*. O exemplo abaixo mostra o mapeamento da coluna *posicaoGPS*, que é do tipo *Point*, e pertence a tabela Contexto.

```
@Lob
@Type(type = "org.hibernatespatial.GeometryUserType")
@Column(name = "posicao_gps", nullable=false)
private Point posicaoGps;
```

4.5. Módulo para visualização dos dados - *WebTrack*

Para visualizar os dados foi desenvolvido um aplicativo que faz o papel do módulo *WebTrack*. Por não exigir recursos muito avançados além do JavaScript, o Módulo Web só possui componentes HTML e códigos em JavaScript.

A API do *Google Maps* possui muitos recursos avançados e úteis aos desenvolvedores, desde recursos para visualização dos dados, como ícones e balões de informações que suportam HTML, até recursos para a criação e manipulação de objetos como pontos, retas e polígonos.

O objetivo principal deste módulo é o de acessar as informações do *ManagerTrack* através do *WebServlet*, e exibir essas informações de acordo com o que foi solicitado no parâmetro *action* passado na URL. Entre as funções solicitadas estão a de exibir todas as rotas, exibir todos os pontos de parada, e a principal que é a de exibir a posição de todos os veículos e atualizá-las em tempo real. Para realizar esta última tarefa e testá-la antes de colocá-la em prática junto com o Módulo Coletor, foram necessárias algumas simulações.

Na aplicação *ManagerTrack*, foi criada a classe *CollectorTrackSimulator* que, como o próprio nome já diz, simula o comportamento do Módulo Coletor. Assim que executada, esta classe cria algumas *Java Threads*, sendo que cada uma dessas *threads* lê, a cada *x* segundos, uma linha de seu arquivo correspondente. Cada linha deste arquivo contém informações básicas que um módulo GPS/GPRS poderia fornecer como id, latitude, longitude e velocidade. Fazendo uma analogia, a classe *CollectorTrackSimulator* estaria representando todo o módulo *CollectorTrack*, cada *thread* representaria um módulo GPS/GPRS, e cada arquivo correspondente seria todas as informações obtidas naquele momento, naquele

veículo. Com isso pode-se testar a exibição dos veículos em tempo real. Primeiro implanta-se as aplicações *ManagerTrack* e *WebTrack* no Tomcat, e depois executa-se a classe *CollectorTrackSimulator* e abre-se a URL do *WebTrack* para visualizar no *Google Maps* a atualização em tempo real dos veículos simulados.

Outra decisão tomada durante o desenvolvimento do projeto foi em relação aos resultados retornados aos módulos de visualização. É importante ressaltar que, apesar de guardarmos em um mesmo arquivo todas as informações relativas a uma entidade do sistema, isso não significa que todas elas devam ser utilizadas. Por exemplo, no arquivo *rotaFeed*, possuímos informações de todas as rotas que estão cadastradas no sistema, porém caso seja interessante mostrar somente a rota "UFLA – Rodoviária", então o módulo de visualização deve cuidar para filtrar somente por esta rota. A decisão de colocar todas as rotas em um arquivo só é justamente para que este arquivo seja acessado por todos que solicitarem rotas, independente de qual rota, além de facilitar na organização e visualização do sistema. Obviamente, é possível obter soluções mais eficientes dependendo da complexidade e principalmente da abrangência do sistema. Caso o sistema seja aplicado a nível mundial, por exemplo, para evitar que os arquivos fiquem grandes demais, poderíamos distribuí-los em diretórios. Por exemplo, caso o usuário solicite todas as rotas de "Lavras – MG – Brasil", o *Servlet* procuraria pelo diretório "Brasil", dentro deste procuraria por "MG", e depois por "Lavras", retornando somente os resultados para a cidade de Lavras.

4.6. Testes e visualização do sistema

Apesar de até o momento da conclusão deste trabalho a integração entre os módulos não tenha sido completamente testada, o sistema se comportou conforme o planejado. Através da ferramenta de programação (*IDE*) e de

aplicações como *Firebug*, foi possível acompanhar as informações sendo geradas no Módulo Coletor, simulado pela aplicação, sendo processadas e disponibilizadas pelo Módulo Gerente, e depois sendo exibidas no Módulo Web. Todas as conversões e tratamentos ocorrerão conforme o previsto. A Figura 4.6 exibe a tela de visualização do Módulo Web.

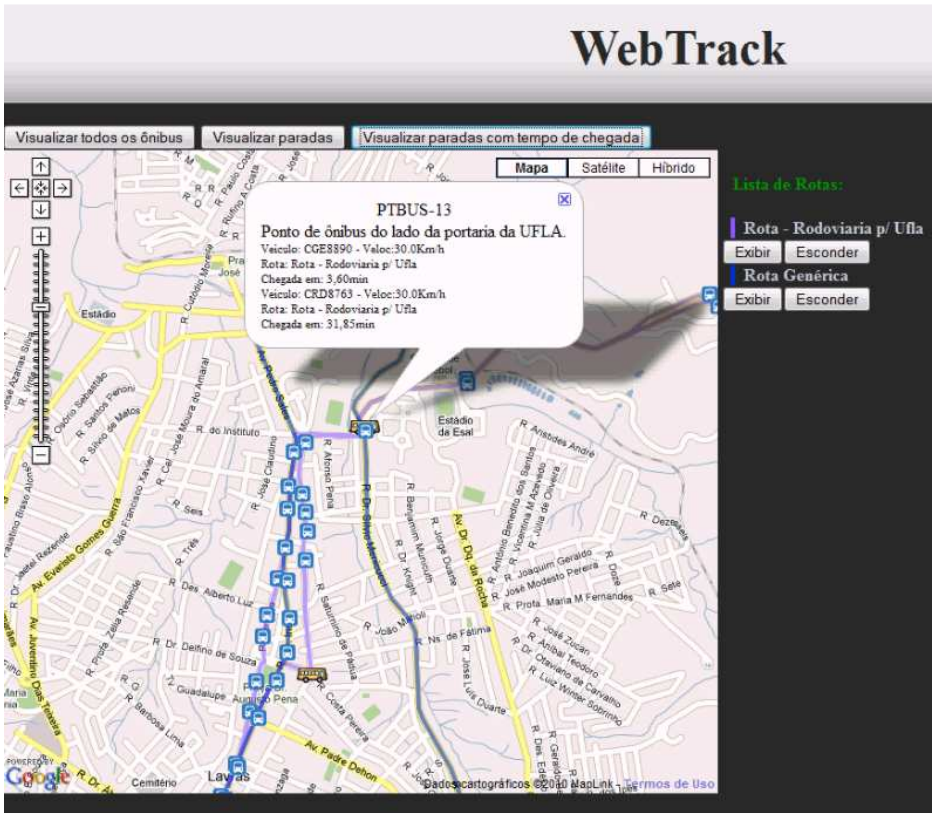

Figura 4.6: Tela de visualização do Módulo Web.

Cabe ressaltar que este trabalho manteve sua prioridade no projeto e desenvolvimento do Módulo Gerente, e a criação do Módulo Web teve o intuito de somente exibir as informações processadas pelo Módulo Gerente, portanto a interface pode e deve sofrer ajustes futuramente.

5. CONCLUSÕES

Este trabalho se propôs a apresentar um sistema que utiliza dados georeferenciais para proporcionar informações relevantes aplicadas ao setor de transporte público. Seu foco maior esteve no Módulo Web e principalmente no Módulo Gerente, que é responsável por conectar todos os módulos funcionais do sistema.

Analisando as tecnologias empregadas atualmente no setor de transporte público, fica evidente a carência que este setor tem perante as tecnologias disponíveis no mercado. A única solução para essa situação é que seja dado o devido valor a esse setor, que só tende a crescer ao longo dos anos. As próprias companhias ligadas ao transporte público devem investir mais em tecnologia, para atrair mais público, e consequentemente, trazer mais desenvolvimento para esta área.

O objetivo geral desse sistema foi fornecer uma base sólida para a criação de serviços em cima de uma aplicação. Sua arquitetura foi planejada de maneira a permitir ao máximo a extensão desse sistema para outros sistemas mais complexos sem que seu núcleo seja afetado por tais modificações. A partir dessa base podemos agregar serviços variados, com foco sempre voltado ao transporte público. Tais serviços que podem ser agregados vão desde funcionalidades simples, como a de visualização de rotas e paradas, até sistemas mais complexos como, por exemplo, um sistema de mineração de dados para exibir informações em um display no ônibus de acordo com sua localização atual.

5.1. Trabalhos Futuros

Como qualquer sistema complexo, sempre há a necessidade de pensarmos em situações extremas e planejarmos mais a respeito de falhas que o sistema poderia sofrer, além de melhorias funcionais e não funcionais.

Apesar de ser um assunto que estava fora do escopo deste trabalho, dentre alguns dos tópicos que mais se discutem atualmente em aplicações *web*, está o de segurança. A segurança é um item de grande importância para qualquer sistema, porém se tratando de sistemas *web*, tal atenção deve ser redobrada, pois tais sistemas estão sujeitos a ataques de todos os tipos. Devem ser analisadas e implementadas técnicas para evitar qualquer tipo de uso indevido do sistema.

Dentre as melhorias, funcionais e não funcionais, devem ser estudados e comparados métodos para aumentar a precisão das informações coletadas, assim como a sua exibição nos módulos de visualização.

Mesmo não sendo o foco principal deste trabalho, será interessante realizar análises da interface (Módulo Web). Essas análises podem incluir avaliações feitas por usuários ou até mesmo através estudos aprofundados sobre maneiras de como criar interfaces simples, porém intuitivas. Todos esses aperfeiçoamentos atingem somente o Módulo Web, porém o sistema inteiro é beneficiado.

6. BIBLIOGRAFIA

AMARANTE, R. R. **Desenvolvimento de sistema AVL com regras para atualização de posição inteligente que melhora a representação dos trajetos**. Campinas, SP, 2007. Dissertação (Mestrado em Engenharia Civil) – Universidade Estadual de Campinas. Faculdade de Engenharia Civil, Arquitetura e Urbanismo – Programa de Pós-Graduação em Engenharia Civil.

AMBLER, S. Mapping Objects to Relational Databases. Disponível em: <http://www.agiledata.org/essays/mappingObjects.html>. Acessado em: 31 jul. de 2010.

ANTP – Associação Nacional de Transporte Público – Sistema de Informações da Mobilidade Urbana, Relatório Geral 2008 – Dezembro 2009. Disponível em: <http://portal1.antp.net>. Acessado em: 03 ago. de 2010.

AZAMBUJA, A. M. V. de. **Análise de Eficiência na Gestão do Transporte Urbano por Ônibus em Municípios Brasileiros**. Florianópolis, SC, 2002. Tese (Doutorado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, UFSC.

BASHAM, B.; Sierra K.; Bates, B. **Head First Servlets and JSP**. O' Reilly Media, 2004.

BAUER, C.; King G. **Java Persistence with Hibernate**. Nova Iorque: Manning Publication Co, 2007.

CASTRO, E. P. de. **Estudo e Desenvolvimento de uma Plataforma de Coleta, Análise e Visualização de Dados Georeferenciais Aplicados Ao Setor de Transporte Público: Módulo de Coleta de Dados**. Lavras, MG, 2010. Monografia (Graduação em Ciência da Computação) – Universidade Federal de Lavras, Lavras, MG.

CUNHA, J. N. **Metodologia de geração dinâmica de padrões de viagens rodoviárias para monitoramentos inteligentes de veículos de carga em sistemas AVL**. São Paulo, SP, 2008. Dissertação (Mestrado em Engenharia de Transportes) – Escola Politécnica da Universidade de São Paulo, São Paulo, SP.

DANA, P. H. **Global Positioning System Overview**. Disponível em: <http://www.colorado.edu/geography/gcraft/notes/gps/gps_f.html>. Acessado em: 20 mai. de 2010.

FERREIRA, P. R. **Metodologia Para Geração de Mapas de Transporte Público em Ambiente SIG Livre Via Web**. Rio de Janeiro, RJ, 2007. Dissertação (Mestrado em Engenharia de Transporte) – Universidade Federal do Rio de Janeiro, UFRJ.

FOWLER, M. **Padrões de Arquiteturas de Aplicações Corporativas**. Porto Alegre: Bookman, 2006.

GANG JING *et al*. Design of an Intelligent Transportation System Based on GPS and GPRS, 2006. **In:** ICWMMN2006 Proceedings - College of Information and Electrical Engineering, Shandong University of Science and Technology, Shandong, China.

GIL, A. C. **Como elaborar projetos de pesquisa**. São Paulo: Atlas, 1991.

GIL, A. C. **Métodos e Técnicas de Pesquisa Social**. São Paulo: Atlas, 1999.

GOOGLE MAPS - API Family. Disponível em: <http://code.google.com/apis/maps/index.html>. Acessado em: 05 set. de 2010.

ITIF – The Information Technology & Innovation Fundation – Intelligent Transportation Systems, January 2010, Stephen Ezell. Disponível em: <http://www.itif.org/>. Acessado em: 18 jun. de 2010.

INTELLIDRIVE – *About*. Disponível em: <http://www.intellidriveusa.org/>. Acessado em: 17 jun. de 2010.

JAVA SERVLET TECHNOLOGY. Disponível em: <http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html>. Acessado em: 05 set. de 2010.

KEITH, M.; Schincariol M. **Pro EJB 3 – Java Persistence API**. Nova Iorque: Apress, 2006.

MARQUES, A. P. **Proposta de um Programa de Gestão da Qualidade para uma Empresa Genérica de Posicionamentos com GPS**. São Carlos, SP, 2006. Tese (Doutorado em Engenharia de Transportes) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos.

MORESI, E. **Metodologia da Pesquisa**. Brasília, DF: Universidade Católica de Brasília – Pró-reitoria de Pós-Graduação – Programa de Pós-Graduação Stricto Sensu em Gestão do Conhecimento e Tecnologia da Informação, 2003.

OGC - Open Geospatial Consortium, Inc. Disponível em: <http://www.opengeospatial.org/>. Acessado em: 12 jul. de 2010.

PACHECO, M. Z. **Metodologia para o aproveitamento de dados de fontes heterogêneas na criação de informações geo-espaciais**. Ponta Grossa, PR, 2009. Dissertação (Mestrado em Gestão do Território) – Universidade Estadual de Ponta Grossa, Ponta Grossa.

POSTGIS. Disponível em: <http://postgis.refractions.net/documentation/>. Acessado em: 12 jul. de 2010.

POSTGRESQL. Disponível em: <http://www.postgresql.org/>. Acessado em: 12 jul. de 2010.

ROCHA, R. F. F. **Estudo e Desenvolvimento de um Sistema de Coleta, Análise e Visualização de Dados Georeferenciais Aplicado Ao Setor de Transporte Público: Módulo Móvel**. Lavras, MG, 2010. Monografia (Graduação em Ciência da Computação) – Universidade Federal de Lavras, Lavras, MG.

SCOOT – *Split Cycle and Offset Optimisation Technique*. Disponível em: <http://www.scoot-utc.com/>. Acessado em: 17 jun. de 2010.

SILVA, A. P. da. **Gestão de acidentes na cidade universitária de "Zeferino Vaz" com o uso de geotecnologias e softwares livres.** Campinas, SP, 2009. Dissertação (Mestrado em Engenharia Civil) – Faculdade de Engenharia Civil, Arquitetura e Urbanismo – Universidade Estadual de Campinas, UNICAMP.

SILVA, C. M. de P. e. **Utilização do Sistema de Posicionamento Global para Monitoramento de Transporte Fretado**. Campinas, SP, 2006. Dissertação (Mestrado em Engenharia Civil) – Faculdade de Engenharia Civil, Arquitetura e Urbanismo – Universidade Estadual de Campinas, UNICAMP.

SILVA, D. M. da. **Sistemas Inteligentes no Transporte Público Coletivo por Ônibus**. Porto Alegre, RS, 2000. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-Graduação em Engenharia de Produção, UFRGS.

SILVA, E. L. da; Menezes, E. M. **Metodologia da pesquisa e elaboração de dissertação** 3. ed. rev. atual. – Florianópolis, SC: Laboratório de Ensino a Distância da UFSC, 2001.

SILVEIRA, A. C. da. **Avaliação de desempenho de aparelhos receptores GPS**. Campinas, SP, 2004. Dissertação (Mestrado em Engenharia Agrícola) – Universidade Estadual de Campinas, Faculdade de Engenharia Agrícola.

SIT – *Sistema de Información de Transportes* – What's CRTM. Disponível em: <http://www.ctm-madrid.es>. Acessado em: 16 jun. de 2010.

TFL – *Transport for London*. Disponível em: <http://www.tfl.gov.uk>. Acessado em: 16 jun. de 2010.

TEIXEIRA, E. H. de S de B; Barth, J. V; Barros, P. L. **O Aperfeiçoamento do Transporte Público Brasileiro Através da Aplicação de Sistemas Inteligentes**. Rio de Janeiro, RJ, 2005. Instituto Alberto Luiz Coimbra de Pós-graduação e Pesquisa de Engenharia – Universidade Federal do Rio de Janeiro – Programa de Engenharia de Transportes, COPPE/UFRJ.

TOSCANO, A. B. A. **Um ambiente na web para otimização do transporte de passageiros sob regime de fretamento.** Fortaleza, CE, 2006. Dissertação (Mestrado em Informática Aplicada) – Fundação Edson Queiroz – Universidade de Fortaleza, UNIFOR.

TRANSPORT DIRECT – About Us. Disponível em: <http://www.transportdirect.info>. Acessado em: 16 jun. de 2010.

UNIVERSITY OF MICHIGAN PARKING & TRANSPORTATION SERVICES. Disponível em: <http://pts.umich.edu/>. Acessado em: 19 jun. de 2010.

W3SCHOOLS. Disponível em: <http://w3schools.com/js/js_intro.asp>. Acessado em: 05 set. de 2010.