UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA POLITÉCNICA

DEPARTAMENTO DE ELETRÔNICA

Sistema de Informação e Monitoramento do Tráfego

Autor: _____

João Leal Noya

Orientador: _____

Antônio Cláudio Gómez de Sousa

Examinador : _____

Aloysio de Castro Pinto Pedroza

Examinador : _____

Albino dos Anjos Aveleda

DEL

Abril de 2008

“A esperança não é a convicção de que as coisas vão dar certo,
mas a certeza de que as coisas têm sentido,
como quer que venham a terminar.”
Václav Havel

AGRADECIMENTOS

Agradeço este trabalho inicialmente à minha mãe Marlene Aparecida Leal, ao meu pai Roberto Noia de Miranda e à minha irmã Roberta por terem me apoiado sempre em todos esses anos de faculdade e em todos os anos da minha vida. Ao meu tio Célio, à minha tia Conceição e ao meu tio Pedro pelos conselhos e incentivos que me fizeram seguir em frente. À minha avó Dona Zizinha por sua fé infinita. Aos meus primos, tios e tias que gosto tanto. Mesmo distantes sempre me deram uma força.

Ao professor Ronaldo Balassiano pela oportunidade de trabalhar em um grande projeto acadêmico. Ao professor Antônio Cláudio Gómez de Sousa por realizar a orientação deste trabalho e pelos conhecimentos em engenharia de software. Aos professores Aloysio Pedroza e Albino Aveleda pela avaliação deste trabalho, muito importante na busca por melhorias.

Ao Programa de Engenharia de Transportes por permitir a utilização da infra-estrutura necessária para o desenvolvimento do projeto. À toda equipe do laboratório *Planet* pelo apoio e pelo ótimo ambiente de trabalho.

À Universidade Federal do Rio de Janeiro por ter me proporcionado um ensino de qualidade e oportunidades de crescimento profissional.

RESUMO

O Sistema de Informação e Monitoramento do Tráfego visa ser uma ferramenta que permita consultas e análises sobre horários e condições do tráfego de uma rede de transportes, através de uma interface simples e de fácil acesso aos seus usuários. Este projeto se restringe a monitorar a operação dos ônibus do campus da Ilha do Fundão, da Universidade Federal do Rio de Janeiro, e serve de protótipo para o desenvolvimento de um sistema mais abrangente.

Sua construção foi motivada principalmente pela crescente necessidade de organização do sistema de transporte público urbano. A tecnologia *GPS* de rastreamento possibilita a obtenção de dados sobre a operação dos veículos de transporte. Este monitoramento permite o levantamento de informações importantes para seus usuários e administradores.

A engenharia de software orientada a objetos foi a metodologia utilizada. Sua importância se mostra na organização e controle do processo de desenvolvimento, contribuindo para que o produto final tivesse uma estrutura confiável e bem documentada.

A escolha das tecnologias foi influenciada pelo desejo em aprender algumas das tecnologias mais utilizadas no mercado, principalmente o emprego da plataforma *J2EE*, que oferece uma variedade de recursos que facilitam a implementação de sistemas.

Os resultados obtidos foram considerados satisfatórios. Apesar das dificuldades encontradas, a metodologia e as tecnologias utilizadas contribuíram para que o esforço de desenvolvimento fosse atenuado. Além disso, este projeto permitiu adquirir experiência na aplicação da metodologia e da tecnologia *J2EE*.

PALAVRAS-CHAVE

- Engenharia de Software
- *J2EE – Java 2 Enterprise Edition*
- Padrão de Projeto
- *Framework*
- SIMT – Sistema de Informação e Monitoramento do Tráfego

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

GLOSSÁRIO

Container : Interface entre um componente e as funções de baixo nível da plataforma onde roda. È uma espécie de sistema operacional para objetos.

JDBC (Java Database Connectivity) : Conjunto de classes e interfaces que permitem acesso a determinado banco de dados a partir de código Java.

JNDI (Java Naming and Directory Interface) : Serviço que permite associar um nome a um recurso computacional e localizar um recurso a partir do seu nome.

JSP (JavaServer Page) : Arquivo contendo código *HTML* para apresentação de uma resposta *HTTP* gerada por um *servlet*. Os dados de resposta são inseridos no código HTML, separando-o do código Java existente nos *servlets*.

HTML (Hyper Text Markup Language) : Linguagem para formatação de páginas *web*, interpretada pelos navegadores *web*.

HTTP (HyperText Transfer Protocol) : Protocolo de comunicação para transferência de arquivos na Internet, através do tratamento de requisições / respostas entre cliente e servidor *web*.

MVC (Model-View-Controller) : Padrão arquitetural para separar dados ou lógica de negócios da interface do usuário e do fluxo da aplicação.

RMI-IIOP (Remote Method Invocation - Internet Inter-ORB Protocol): Protocolo de comunicação que permite a um objeto chamar métodos em objetos remotos.

Servlet : Programa escrito em Java, executado num servidor *web*, que obtém dados de uma requisição *HTTP* e gera uma resposta *HTTP*.

TCP / IP (Transmission Control Protocol / Internet Protocol) : Protocolos de comunicação entre computadores em rede.

UML (Unified Modeling Language) : Linguagem para especificação e documentação de projetos orientados a objeto.

URL (Uniform Resource Locator) : É o endereço de um recurso disponível em uma rede de computadores.

WML (Wireless Markup Language) : Linguagem interpretada pelos navegadores *wap*, semelhante à linguagem *HTML*.

WAP (Wireless Application Protocol) : Protocolo de comunicação para aplicações sem fio. Permite acesso à Internet pelos celulares.

XML (Extensible Markup Language) : Linguagem para descrição de tipos de dados em documentos de texto, separando o conteúdo dos dados da formatação do texto.

1. INTRODUÇÃO

Este documento apresenta o resultado do processo de desenvolvimento do produto **Sistema de Informação e Monitoramento do Tráfego (SIMT),** realizado a título de Projeto Final do curso de Engenharia Eletrônica e Computação da Universidade Federal do Rio de Janeiro. A elaboração e o desenvolvimento do projeto de software foram acompanhados pelo professor Antônio Cláudio Gómez de Sousa, do Departamento de Engenharia Eletrônica (DEL) da UFRJ. O professor Ronaldo Balassiano, do Programa de Engenharia de Transportes (PET) da COPPE/UFRJ, orientou a definição dos conceitos relacionados ao transporte público e a definição dos requisitos do sistema.

1.1. Motivação

A idéia de desenvolver este software surgiu com a participação em projeto do laboratório *Planet*, do PET, coordenado pelo professor Ronaldo Balassiano.

No contexto atual do planejamento de transportes no Brasil, pode ser observado que a tecnologia de rastreamento de veículos tem um potencial amplo de utilização, ainda não explorado. Durante pesquisas em busca de soluções de software desenvolvidas no Brasil para o setor de transporte público, notou-se que já existem alguns sistemas com informações de horários de transporte coletivo, mas sem suporte à anàlise de dados sobre a operação.

A possibilidade de monitoração em tempo real da operação de veículos para transporte de passageiros ou para prestação de outros serviços aumenta consideravelmente as chances de operação adequada e eficiente dos sistemas de transportes que trafegam, em especial, em áreas urbanas.

Empresas prestadoras de serviços de transporte de passageiros necessitam de práticas operacionais diferenciadas. Órgãos de concessão de transporte público precisam se valer de uma ferramenta eficiente para fiscalização dos serviços. Informações mais precisas e a melhoria da qualidade dos serviços poderão atrair um maior número de usuários, atualmente optando por outras formas de locomoção, em especial, o carro privado.

Outro fator de motivação foi a possibilidade de aprendizado e utilização de tecnologias de ampla aceitação pelo mercado de desenvolvimento de sistemas, além da prática da engenharia de software orientada a objetos como metodologia escolhida.

1.2. Metodologia

Para desenvolvimento do sistema, adotou-se a engenharia de software orientada a objetos, uma metodologia eficaz e muito utilizada por desenvolvedores de sistemas. Entre os motivos dessa escolha, temos:

- Oportunidade de aprofundar conhecimentos sobre as técnicas de engenharia de software orientada a objeto;
- Aprendizado das tecnologias que compõem a plataforma *J2EE* [7], através da implementação dos modelos teóricos.

Seguindo os conceitos dessa metodologia, o processo de desenvolvimento foi realizado em várias iterações, cada uma passando pelas atividades de análise, projeto, implementação e testes. O resultado final foi uma versão do sistema que procurou atender a todos os requisitos especificados.

1.3. Problema

No Brasil os transportes coletivos são a opção de deslocamento da maioria da população urbana. Entre os transportes coletivos, o ônibus é o meio mais utilizado, pela sua acessibilidade, flexibilidade e pelo atendimento amplo aos desejos de destino da população. Esta enorme demanda de usuários faz com que informações sobre horários e localização dos veículos sejam muito importantes. Como exemplo, muitas vezes um passageiro prefere entrar num ônibus lotado por não saber o tempo que irá esperar pelo ônibus seguinte. A falta de informação pode induzir os usuários a optarem pelo uso do carro, contribuindo ainda mais para o congestionamento urbano e tornando o transporte público ineficiente.

Por outro lado, é necessário fiscalizar e analisar a operação do transporte coletivo. Seu impacto sobre o congestionamento urbano deve ser medido de maneira prática e rápida, tornando sua administração mais eficiente.

Nesse contexto, o problema é projetar um sistema cujo objetivo é permitir a consulta e a análise de informações sobre horários e condições do tráfego. A ferramenta deve disponibilizar meios de acesso de maneira simples, seja através da *web* ou por aplicações *desktop*. Serão utilizadas tecnologias tais como sistema de posicionamento global (*GPS*), banco de dados e Internet .

1.4. Definições, Abreviaturas e Acronismos

A seguir, são apresentadas algumas abreviaturas e acronismos encontrados neste documento. Outros detalhes poderão ser esclarecidos no decorrer deste documento.

GPS – Global Positioning System. È a tecnologia utilizada pelos equipamentos de rastreamento para medir as coordenadas geográficas de uma localidade.

API – Application Programming Interface. É um conjunto de funções utilizado por desenvolvedores para criar programas aplicativos.

IDE – Integrated Development Environment. É um programa que serve de apoio ao desenvolvimento de sistemas com o objetivo de agilizar este processo.

J2EE – Java 2 Enterprise Edition. É uma especificação para servidores de aplicação, juntamente com um pacote de *API´*s e ferramentas para desenvolver componentes que rodam nesses servidores.

EJB – Enterprise JavaBean. São componentes *J2EE* que rodam num servidor de aplicação e atendem a um formato padrão de construção definido na especificação.

Servidor de Aplicação – Servidor que oferece ambiente para operação de componentes J2EE e diversos serviços de infra-estrutura.

Clientes de Aplicação – Consistem de aplicações gráficas ou de linha de comando, que permitem uma interface de usuário mais sofisticada.

Padrões – São soluções para problemas que ocorrem repetidas vezes no desenvolvimento de sistemas, permitindo a reutilização de idéias.

Framework – Conjunto de vários padrões relacionados em um nível de abstração mais alto, representando uma solução completa para um problema genérico.

SIMT – Sistema de Informação e Monitoramento do Tráfego de uma rede de transporte.

1.5. Sumário

O documento que se segue está dividido em quatro partes: “Concepção”, “Especificação”, “Desenvolvimento” e “Conclusões”.

Na etapa de “Concepção” procurou-se obter uma estimativa da ordem de grandeza do sistema, através de uma exploração rápida dos principais atributos, requisitos e restrições. A idéia foi realizar uma investigação para formar uma opinião clara e justificável da finalidade geral e da viabilidade do produto.

A parte de “Especificação” reúne as informações obtidas durante as interações de análise e projeto do sistema. Toda a funcionalidade do sistema é descrita através do modelo de casos de uso, identificando atores e seus objetivos. Os principais conceitos são descritos através de classes que irão modelar o comportamento do sistema. O modelo de projeto descreve a realização dos casos de uso pelos diagramas de interação, em termos de classes de projeto que colaboram entre si. Os componentes são projetados segundo sua ditribuição física em camadas.

Em “Desenvolvimento” descrevem-se as interações ocorridas durante a implementação do sistema, assim como as características da metodologia que foram aplicadas. As escolhas de projeto são justificadas e os resultados são apresentados, além de relatar os principais problemas encontrados.

O documento apresenta ao final as “Conclusões”, com a discussão sobre o sistema final, comparando o que era esperado e o que foi implementado, além da eficácia da metodologia e da tecnologia utilizadas. No fim, são apresentadas propostas de melhoria e uma análise das competências adquiridas.

2. CONCEPÇÃO

2.1. Perspectivas do Produto

O sistema foi especificado para disponibilizar informações sobre horários e condições do trafégo em tempo real, de maneira a aumentar a satisfação dos usuários de transporte urbano, além de realizar análises quantitativas sobre a operação das linhas na rede de transporte.

Este projeto está restrito às linhas de transporte que operam no campus da Ilha do Fundão, da UFRJ, mas pode ser encarado como um protótipo para a construção de um ambiente de software mais abrangente no setor de transporte público, capaz de atender a uma crescente demanda de utilização.

Podemos observar, pelo diagrama abaixo, o contexto no qual o sistema está inserido e os atores que interagem.

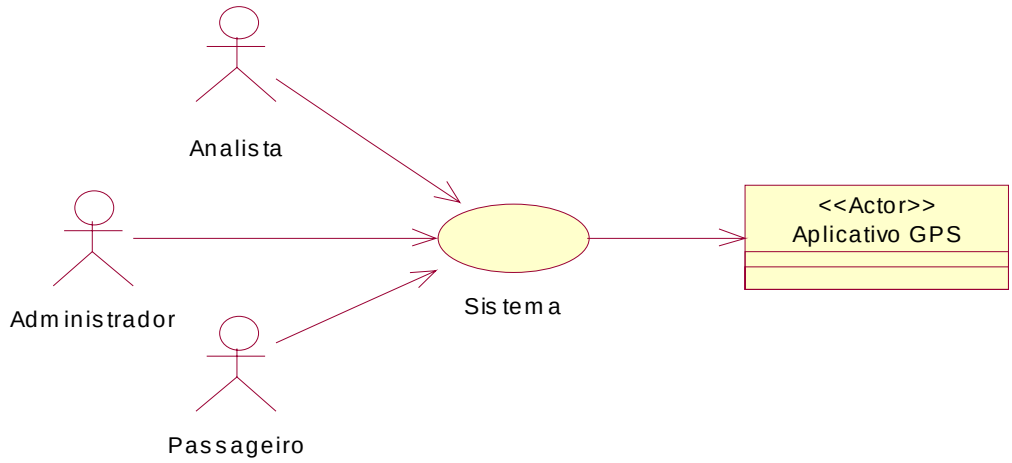

Figura 1 : Diagrama de Contexto do Sistema

O Passageiro é o usuário de uma rede de transporte. O Analista estuda a operação da rede de transporte. O Administrador é o responsável por gerenciar o sistema. O Aplicativo *GPS* fornece os dados que o sistema necessita para realizar suas funções.

Do ponto de vista operacional, um sistema abrangente poderá aumentar a confiabilidade na rede de transporte, permitindo o acompanhamento de horários pré-fixados para operação. Poderá também facilitar a diversificação dos serviços oferecidos, atendendo nichos específicos de mercado (serviços sob demanda) e a integração com outros modos de

transporte. Além disso, um sistema de transportes coletivo, operando de forma eficiente, poderá no médio prazo, desde que adequadamente gerenciado, atrair um maior número de usuários atualmente optando por outras formas de locomoção, em especial, o carro privado.

Alguns benefícios econômicos são obtidos quando se consegue um sistema de transporte eficiente. Com a utilização em maior escala dos equipamentos necessários ao rastreamento e controle de veículos, os custos para implantação desses serviços tendem a se reduzir, aumentando dessa forma a possibilidade de difusão tecnológica em maior escala. A redução no uso de carros privados é um ponto muito importante, pois contribui para a redução de congestionamentos e consumo de combustível, diminuindo o custo de "tempo de viagem" que existe em muitos serviços em nossa sociedade. Reduzem-se também impactos ambientais produzidos por sistemas de transporte como a poluição atmosférica e o ruído.

2.2. Características do Usuário

O conteudo do sistema foi especificado para usuários de uma rede de transporte público, insatisfeitos com o congestionamento urbano e a falta de informação sobre os horários das linhas. São usuários de todas as idades e de todos os níveis de escolaridade, que esperam por informações rápidas e confiáveis para garantir uma locomoção sem transtornos. Devem possuir conhecimentos básicos de informática e acesso a Internet, seja por um computador ou por um celular.

Considerando a administração e o planejamento das empresas de transporte e orgãos públicos, gerentes e analistas podem usar o sistema para fiscalizar a operação dos veículos e obter informações quantitativas e qualitativas sobre a operação da rede de transporte.

2.3. Pressupostos e Dependências

O sistema desenvolvido apresenta dependência de um segundo sistema. É o aplicativo *GPS*, responsável pela aquisição e armazenamento dos dados enviados pelos equipamentos de rastreamento. Seu correto funcionamento é vital para a operação adequada do sistema.

As requisições dos usuários são feitas a partir de máquinas remotas e de aparelhos celulares. Pressupõe-se que tanto a rede de computadores quanto a rede de telefonia móvel estejam em condições normais de operação, para o correto funcionamento do sistema.

Os programas desenvolvidos com a tecnologia Java, utilizada pelo sistema, necessitam que a *Java Virtual Machine (JVM)* esteja instalada em qualquer máquina onde algum componente do sistema irá rodar.

2.4. Interfaces de Software

Duas interfaces de software são utilizadas pelo sistema. Uma delas é o aplicativo *GPS*, desenvolvido pela empresa *Geocontrol*, que recebe e armazena os dados enviados pelos equipamentos de rastreamento, através da rede de telefonia móvel e da Internet. Este aplicativo roda em uma das máquinas da rede interna do laboratório *Planet* e utiliza o banco de dados *FirebirdSql* [10] para armazenar os dados recebidos. Uma das tabelas neste banco contém os dados de posicionamento geográfico num formato pré-definido, sendo a entrada para o sistema realizar suas pesquisas. A outra interface é o banco de dados *MySQL* [9], com a função de armazenar as informações de cadastro necessárias ao funcionamento do sistema.

O sistema utiliza interfaces de programação, específicas à cada um dos bancos citados, para adaptar o acesso e as operações sobre os dados.

2.5. Interfaces de Comunicação

O sistema opera sobre a rede mundial de computadores (Internet), baseando-se na arquitetura cliente-servidor, além de utilizar a rede de telefonia móvel. A comunicação entre o servidor e os clientes será feita utilizando-se os protocolos *GPRS*, *HTTP*, *WAP* e *RMI-IIOP*. A rede interna do laboratorio *Planet* possui capacidade para realizar a comunicação entre diferentes máquinas clientes e o servidor de aplicação. Não haverá interfaces de hardware.

2.6. Requisitos de Desempenho

Os usuários querem saber as condições do tráfego o mais rápido possível. O objetivo é conceber um tempo de resposta a uma consulta pela Internet menor que 10 segundos. Este tempo leva em conta a velocidade de acesso aos bancos de dados e a velocidade do algoritmo de pesquisa.

Prevê-se uma média de acesso muito alta. O desempenho não pode ser reduzido com um aumento no número de acessos. O sistema deve estar disponível nos horários de maior

fluxo de veículos. Um gargalo em potencial é o acesso ao aplicativo externo, além do número de requisições simultâneas que o servidor de aplicação pode atender. Tal situação pode exigir a distribuição do sistema em várias máquinas para dividir a carga de acessos em vários servidores de aplicação.

2.7. Restrições de Projeto

Por influenciar no projeto, alguns fatores restritivos como simplificações, limitações e preferências foram considerados:

- O sistema é desenvolvido na plataforma Java *J2EE*, utilizando-se de suas *API´*s e *frameworks* de desenvolvimento;
- O banco de dados do sistema é o *MySQL*;
- A *IDE* usada para programação é o *Eclipse*;
- 13 equipamentos de rastreamento são utilizados para implementação do sistema. São aparelhos da empresa *Geocontrol*, cujo aplicativo utiliza o banco de dados *FirebirdSql*;
- Os veículos rastreados são apenas os ônibus que operam no interior da Ilha do Fundão, na UFRJ;
- A interface com o usuário é realizada via navegadores *web* e *wap*, além de uma aplicação desktop com a *API Swing*.

Apenas a estrutura de rede e de hardware presente no laboratorio *Planet* – PET foi usada para apoio ao processo de desenvolvimewnto, incluindo:

- Rede Ethernet de 100 Mbs/s;
- Servidor de Aplicação: Processador Intel Pentium Dual Core 3 GHz, 2 GB de memória RAM, rodando Windows Server 2003 SP1;
- Máquinas Clientes: Processador AMD Sempron 1600 MHz, 1 GB de memória RAM, rodando Windows XP SP2.

Os sistema necessita, como recurso especial, do aluguel de uma linha de celular para realizar a transmissão de dados de posicionamento geográfico entre o equipamento de rastreamento e o software aplicativo.

2.8. Atributos

O principal atributo do sistema deve ser a simplicidade no acesso às informações, com uma interface objetiva e de fácil navegação, sendo assim fácil de aprender e usar. A informação deve estar num formato facilmente visível, independente do tipo de interface usada.

A confiabilidade do sistema é muito importante pois servirá de base para seus usuários decidirem quando chegar no ponto de parada ou qual via escolher para não enfrentar congestionamentos. As informações de horário devem ter uma margem de erro menor que 5 minutos.

Além disso, o sistema deve avisar quando for incapaz de obter a informação desejada, devido a condições anormais, como no caso de um veículo parar de enviar dados ao aplicativo *GPS*, por alguma falha no dispositivo de rastreamento.

Cada aplicativo de rastreamento pode utilizar um tipo de banco de dados. Portanto, o sistema deve facilitar a integração com diferentes tipos de banco de dados. Além disso, deve ser capaz de rodar em diferentes plataformas e permitir manutenbilidade a longo prazo.

Diversos ambientes podem necessitar diferentes configurações de rede, de arquitetura do sistema, de interface com o usuário, entre outras. Portanto, o sistema deve permitir que sua configuração seja alterada sem muito esforço.

3. ESPECIFICAÇÃO

3.1. Análise de Requisitos Formais

A identificação dos requisitos começa com uma lista de definições no contexto do problema, oferecendo uma percepção melhor do domínio:

- Um equipamento de rastreamento é um dispositivo eletrônico capaz de registrar a localização geográfica do lugar onde se encontra, junto com a hora em que foi feito o registro. Também é identificado como módulo GPS, aparelho de rastreamento ou rastreador;
- A localização geográfica é caracterizada pela latitude e pela longitude de um lugar;
- O aplicativo de rastreamento é um software capaz de armazenar as informações enviadas pelo equipamento de rastreamento em um banco de dados;
- Uma linha de transporte é um itinerário realizado por uma frota de veículos de transporte de passageiros para satisfazer as necessidades de deslocamento entre dois pontos, atendendo diferentes localidades;
- Um corredor de uma linha de transporte é parte de um percurso que integra duas paradas, uma de origem e outra de destino;
- Uma parada é o local onde ocorrem embarque e desembarque de passageiros, caracterizado por um nome e sua localização geográfica. Também é identificada como ponto de parada;
- O índice de congestionamento é uma medida do nível de congestionamento de um fluxo de veículos trafegando em um corredor, tendo como referência o tempo médio de viagem no mesmo, sem congestionamento;
- O *headway* é uma medida do tempo que se leva entre a passagem de dois veículos de uma mesma linha numa determinada parada ou entre dois pontos quaisquer do percusro.

A listagem a seguir apresenta as regras de negócio, descrevendo as restrições e comportamentos do domínio de interesse:

- Uma empresa possui um nome;
- Uma empresa pode operar mais de uma linha;
- Uma linha é composta por uma seqüência de corredores;

- Cada linha é operada por uma empresa;
- Uma linha possui vários veículos em operação;
- Uma linha possui um nome, uma via e um tempo médio de deslocamento;
- Mesmo que duas empresas operem um mesmo itinerário, serão consideradas linhas diferentes;
- Cada veículo estará equipado com um único equipamento de rastreamento;
- Cada veículo em operação pertence a uma empresa e pode operar uma ou mais linhas;
- Um veículo possui um identificador, uma placa, uma tarifa e uma capacidade de transporte de passageiros;
- Um veículo pode estar operando ou fora de rota;
- O equipamento de rastreamento envia os dados ao aplicativo com um intervalo de tempo que pode ser configurável;
- O equipamento de rastreamento possui um identificador, um modelo e um fabricante;
- O equipamento de rastreamento pode estar operando ou fora de uso;
- Um corredor tem uma parada de origem e outra de destino;
- Cada corredor possui um nome e um tempo médio de deslocamento;
- Dois corredores podem ter as mesmas paradas de origem e de destino, mas pertencerem a linhas diferentes;
- Os corredores de uma linha são sequenciais, sendo a parada de destino de um corredor equivalente à parada de origem do corredor seguinte;
- Se a linha não for circular, o último corredor não tem corredor seguinte;
- Um itinerário é uma seqüência de corredores;
- Um mesmo itinerário pode ser percorrido em tempos diferentes, o que caracteriza linhas diferentes;
- O *headway* só é medido para as paradas que são destino de um corredor;
- O índice de congestionamento é uma medida específica para cada corredor;
- Uma parada pode ser origem ou destino de mais de um corredor;
- Uma parada possui um nome, uma região retangular caracterizada pelas latitudes e longitudes máximas e mínimas, além de latitude e longitude central desta região;
- Para um usuário realizar uma análise da operação de uma determinada linha, ele necessita de um nome e uma senha que permitam acessar o sistema;
- Para consultar as condições do tráfego, o usuário não necessita ser autenticado pelo sistema.

Conforme apresentado abaixo, os requisitos levantados foram agrupados para facilitar a análise. Essa listagem exprime as características do sistema num formato compacto, para resumir toda a funcionalidade:

Condições do Tráfego

- Disponibilizar as condições do tráfego pela Internet, seja pelo computador ou por aparelho celular;
- Informar a posição geográfica dos veículos de transporte operando sobre uma determinada linha;
- Informar o índice de congestionamento dos corredores de uma determinada linha;
- Estimar o tempo de chegada a uma parada do próximo veículo operando em uma determinada linha.

Análise de operação

- Gerar relatório sobre um corredor, listando todas as medidas do índice de congestionamento obtidas ao longo do dia, destacando os valores máximo, mínimo e o número de vezes que foi completado;
- Gerar relatório sobre o *headway* de uma parada, listando todas as medidas obtidas ao longo do dia, destacando os valores máximo e mínimo;
- Gerar relatório sobre uma linha, listando a seqüência de corredores de todos os veículos que a percorreram;
- Gerar relatórios sobre os veículos, listando a seqüência dos corredores percorridos ao longo do dia;
- Gerar relatório sobre os aparelhos de rastreamento, listando os intervalos de transmissão que ocorreram ao longo do dia.

Segurança

- Permitir o login no sistema;
- Permitir o logout do sistema.

Administração

- Permitir o cadastro, consulta e edição de empresas;
- Permitir o cadastro, consulta e edição de linhas;
- Permitir o cadastro, consulta e edição de corredores;
- Permitir o cadastro, consulta e edição de paradas;
- Permitir o cadastro, consulta e edição de itinerário;
- Permitir o cadastro, consulta e edição de veículos;
- Permitir o cadastro, consulta e edição de equipamentos de rastreamento;
- Permitir o cadastro, consulta e edição da configuração veículo-equipamento;
- Permitir o cadastro, consulta e edição de usuários.

3.2. Análise de Casos de Uso

Os casos de uso permitem detalhar as funcionalidades que foram apresentadas anteriormente pela listagem de requisitos. O foco desta análise está em como o sistema pode satisfazer os objetivos pretendidos pelos usuários.

A tabela a seguir apresenta os casos de uso elaborados, separados em pacotes de acordo com suas funcionalidades. Cada caso de uso é descrito em seguida, apresentando os fluxos normais e alternativos de interação com o sistema.

Tabela 1 : Pacotes dos Casos de Uso

PACOTES	**CASOS DE USO**
Tráfego	- Consultar Condições do Tráfego. - Localizar Veículos.
Análise	- Gerar Relatório de Linha. - Gerar Relatório de Corredor. - Gerar Relatório de Headway. - Gerar Relatório de Veículo - Gerar Relatório de Equipamento de Rastreamento
Usuários	- Adicionar Usuário. - Listar Usuários. - Editar ou Excluir Usuário.

Empresas	- Adicionar Empresa. - Listar Empresas. - Editar ou Excluir Empresa.
Linhas	- Adicionar Linha. - Listar Linhas. - Editar ou Excluir Linha.
Corredores	- Adicionar Corredor. - Listar Corredores. - Editar ou Excluir Corredor.
Veículos	- Adicionar Veículo. - Listar Veículos. - Editar ou Excluir Veículo.
Paradas	- Adicionar Parada. - Listar Paradas. - Editar ou Excluir Paradas.
Equipamentos de Rastreamento	- Adicionar Equipamento. - Listar Equipamentos. - Editar ou Excluir Equipamento.
Itinerários	- Adicionar Itinerário. - Listar Itinerários. - Editar ou Excluir Itinerário.
Configurações Veículo-Equipamento	- Adicionar Configuração. - Listar Configurações. - Editar ou Excluir Configuração.
Acesso ao Sistema	- Login no Sistema. - Logout do Sistema.

PACOTE: Tráfego

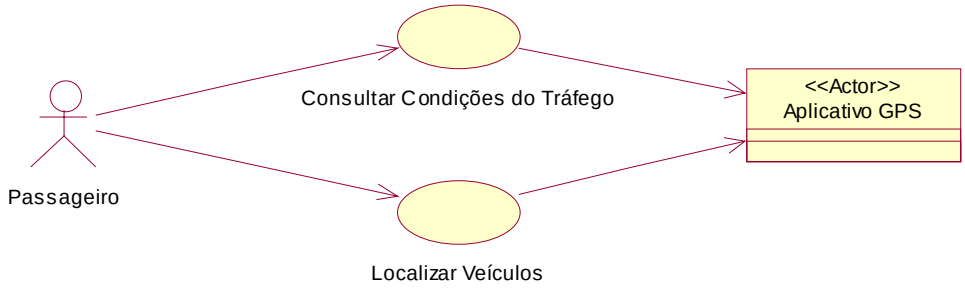

Figura 2 : Casos de Uso - Pacote Tráfego.

CONSULTAR CONDIÇÕES DO TRÁFEGO

Ator Principal: Passageiro.
Pré-Condição: Sistema acessível pela Internet.
Pós-Condição: Passageiro foi informado sobre o tempo estimado para chegar nas paradas dos corredores da linha e seus índices de congestionamento.

Cenários:
Fluxo Principal
1. Passageiro inicia consulta acessando a interface do sistema via Internet;
2. Sistema exibe formulário para seleção da linha de transporte;
3. Passageiro seleciona uma linha;
4. Sistema processa os dados do aplicativo GPS e informa o tempo estimado para o veículo chegar nas paradas dos corredores e os índices de congestionamento.

Fluxos de Exceção
(4) - Sistema não consegue estimar tempo de chegada nas paradas dos corredores
1. Nenhum veículo operando na linha foi encontrado;
2. Passageiro solicita nova consulta;
3. Sistema exibe novamente formulário de consulta.

(4) - Sistema não consegue estimar índice de congestionamento

1. Nenhum veículo pertencente à linha completou o corredor no qual se encontra um dos veículos;
2. Sistema informa o tempo para chegar ao ponto final do corredor sem o índice de congestionamento.

LOCALIZAR VEÍCULOS

Ator Principal: Passageiro.
Pré-Condição: Sistema acessível pela Internet.
Pós-Condição: Passageiro foi informado sobre a localização de todos os veículos que operam na linha selecionada.

Cenários:

Fluxo Principal

1. Passageiro inicia consulta acessando a interface do sistema via Internet;
2. Sistema exibe formulário para seleção da linha de transporte;
3. Passageiro seleciona uma linha;
4. Sistema processa os dados do aplicativo GPS e informa a localização de todos os veículos que operam na linha selecionada.

Fluxos de Exceção

(4) - Sistema não consegue localizar veículos operando na linha

1. Nenhum veículo operando na linha foi encontrado;
2. Passageiro solicita nova consulta;
3. Sistema exibe novamente formulário de consulta.

PACOTE: Análise

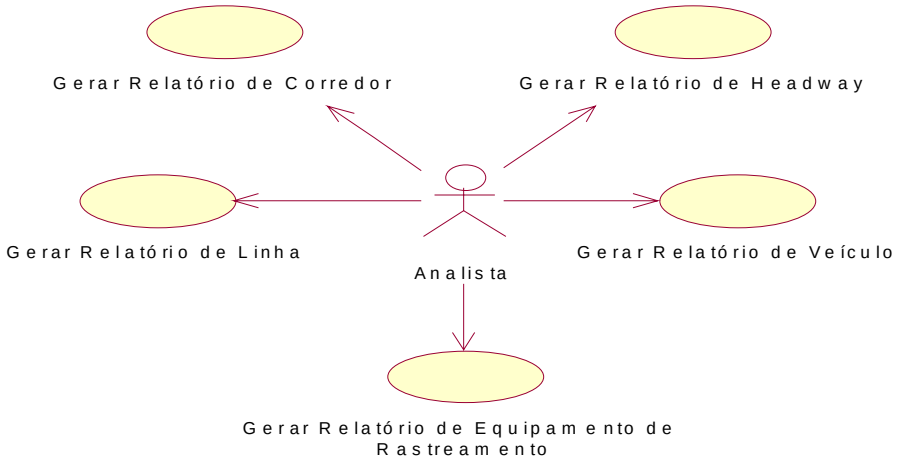

Figura 3 : Casos de Uso - Pacote Análise.

GERAR RELATÓRIO DE LINHA

Ator Principal: Analista.

Pré-Condição: Analista autenticado.

Pós-Condição: Relatório sobre operação em uma linha foi gerado.

Cenários:

Fluxo Principal

1. Analista acessa função de analisar linha no menu de operações;
2. Sistema exibe uma lista com todas as linhas;
3. Analista seleciona uma linha e uma data;
4. Sistema realiza análise e exibe relatório listando a seqüência de corredores de todos os veículos que a percorreram.

GERAR RELATÓRIO DE CORREDOR

Ator Principal: Analista.
Pré-Condição: Analista autenticado.
Pós-Condição: Relatório sobre operação em um corredor foi gerado.

Cenários:
Fluxo Principal
1. Analista acessa função de analisar corredor no menu de operações;
2. Sistema exibe uma lista com todos os corredores;
3. Analista seleciona um corredor e uma data;
4. Sistema realiza análise e exibe relatório com uma lista de todas as medidas do índice de congestionamento obtidas ao longo do dia, destacando os valores máximo, mínimo e o número de vezes que o corredor foi completado.

GERAR RELATÓRIO DE HEADWAY

Ator Principal: Analista.
Pré-Condição: Analista autenticado.
Pós-Condição: Relatório sobre *headway* de uma parada foi gerado.

Cenários:
Fluxo Principal
1. Analista acessa função de analisar *headway* no menu de operações;
2. Sistema exibe uma lista com todas as paradas origem ou destino dos corredores;
3. Analista seleciona uma parada e uma data;
4. Sistema realiza análise e exibe relatório com uma lista de todas as medidas de *headway* da parada, obtidas ao longo do dia, destacando os valores máximo e mínimo.

GERAR RELATÓRIO DE VEÍCULO

Ator Principal: Analista.
Pré-Condição: Analista autenticado.
Pós-Condição: Relatório sobre operação de um veículo foi gerado.

Cenários:
Fluxo Principal
1. Analista acessa função de analisar veículo no menu de operações;
2. Sistema exibe uma lista com todos os veículos;
3. Analista seleciona uma data e um veículo;
4. Sistema realiza análise e exibe relatório listando a seqüência dos corredores percorridos ao longo do dia.

GERAR RELATÓRIO DE EQUIPAMENTO DE RASTREAMENTO

Ator Principal: Analista.
Pré-Condição: Analista autenticado.
Pós-Condição: Relatório sobre operação dos equipamentos foi gerado.

Cenários:
Fluxo Principal
1. Analista acessa função de analisar equipamento no menu de operações;
2. Analista seleciona uma data;
3. Sistema realiza análise e exibe relatório listando os intervalos de transmissão que ocorreram ao longo do dia.

PACOTE: Usuários

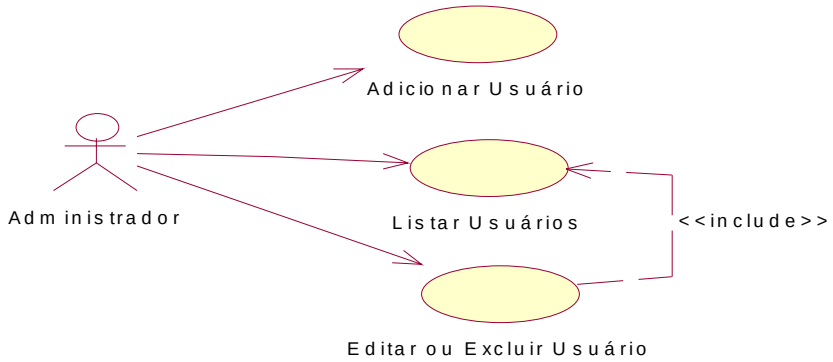

Figura 4 : Casos de Uso - Pacote Usuários.

LISTAR USUÁRIOS

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todos os usuários são listados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar usuário no menu de operações;
2. Sistema exibe uma lista com todos os usuários.

ADICIONAR USUÁRIO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Usuário foi cadastrado.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar usuário no menu de operações;
2. Administrador solicita cadastrar usuário;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona o novo usuário na lista de usuários na tela.

EDITAR OU EXCLUIR USUÁRIO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Usuário foi excluído ou seus dados foram editados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar usuário no menu de operações;
2. Sistema exibe uma lista com todos os usuários;
3. Administrador seleciona um usuário a ser editado ou excluído;
4. Sistema exibe formulário com os dados do usuário selecionado;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui;
7. Sistema remove ou atualiza o usuário na lista de usuários na tela.

PACOTE: Empresas

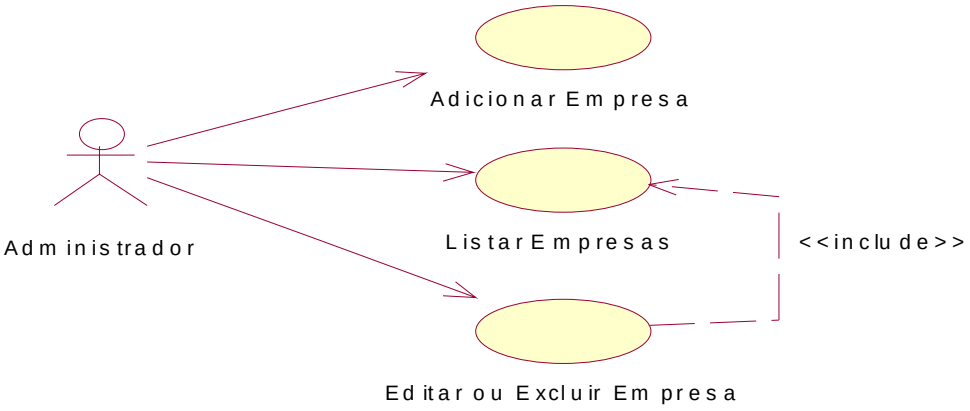

Figura 5 : Casos de Uso - Pacote Empresas.

LISTAR EMPRESAS

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todas as empresas são listadas.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar empresa no menu de operações;
2. Sistema exibe uma lista com todas as empresas.

ADICIONAR EMPRESA

Ator Principal: Administrador.
Pré-Condição: Administrador autenticado.
Pós-Condição: Empresa foi cadastrada.

Cenários:
Fluxo Principal
1. Administrador acessa função de administrar empresa no menu de operações;
2. Administrador solicita cadastrar empresa;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona a nova empresa na lista de empresas na tela.

EDITAR OU EXCLUIR EMPRESA

Ator Principal: Administrador.
Pré-Condição: Administrador autenticado.
Pós-Condição: Empresa foi excluída ou seus dados foram editados.

Cenários:
Fluxo Principal
1. Administrador acessa função de administrar empresa no menu de operações;
2. Sistema exibe uma lista com todas as empresas;
3. Administrador seleciona uma empresa a ser editada ou excluída;
4. Sistema exibe formulário com os dados da empresa selecionada;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui a empresa, incluindo as linhas, veículos, corredores, itinerários e configurações veículo-equipamento referentes à empresa;
7. Sistema remove ou atualiza a empresa na lista de empresas na tela.

PACOTE: Linhas

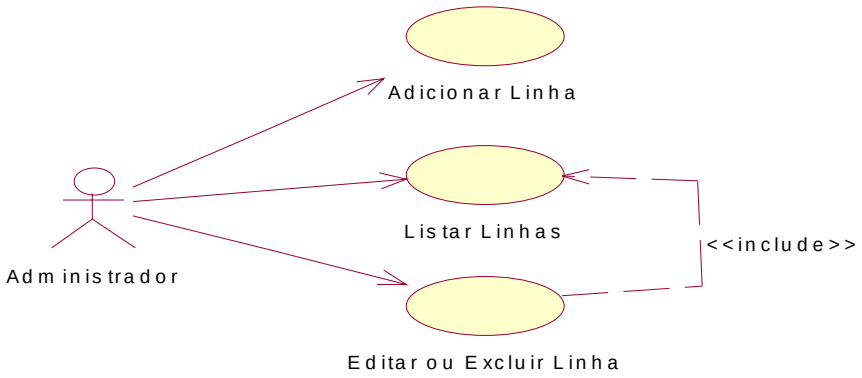

Figura 6 : Casos de Uso - Pacote Linhas.

LISTAR LINHAS

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todas as linhas são listadas.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar linha no menu de operações;
2. Sistema exibe uma lista com todas as linhas.

ADICIONAR LINHA

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Linha foi cadastrada.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar linha no menu de operações;
2. Administrador solicita cadastrar linha;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona a nova linha na lista de linhas na tela.

EDITAR OU EXCLUIR LINHA

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Linha foi excluída ou seus dados foram editados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar linha no menu de operações;
2. Sistema exibe uma lista com todas as linhas;
3. Administrador seleciona uma linha a ser editada ou excluída;
4. Sistema exibe formulário com os dados da linha selecionada;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui a linha, incluindo os corredores e itinerários referentes à linha;
7. Sistema remove ou atualiza a linha na lista de linhas na tela.

PACOTE: Corredores

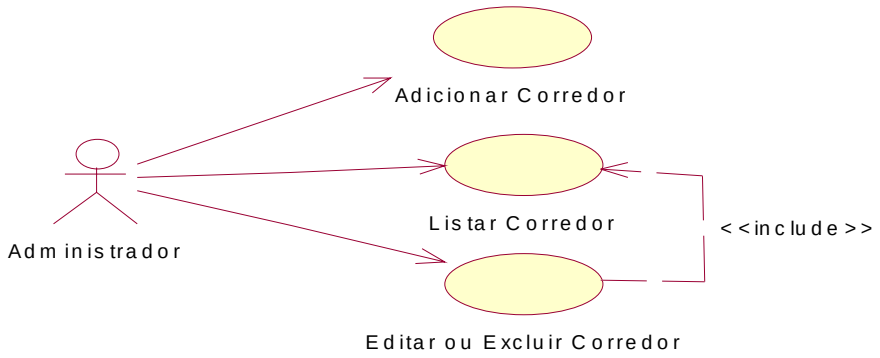

Figura 7 : Casos de Uso - Pacote Corredores.

LISTAR CORREDORES

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todos os corredores são listados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar corredor no menu de operações;
2. Sistema exibe uma lista com todos os corredores.

ADICIONAR CORREDOR

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Corredor foi cadastrado.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar corredor no menu de operações;
2. Administrador solicita cadastrar corredor;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona o novo corredor na lista de corredores na tela.

EDITAR OU EXCLUIR CORREDOR

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Corredor foi excluído ou seus dados foram editados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar corredor no menu de operações;
2. Sistema exibe uma lista com todos os corredores;
3. Administrador seleciona um corredor a ser editado ou excluído;
4. Sistema exibe formulário com os dados do corredor selecionado;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui o corredor, incluindo os itinerários referentes ao corredor;
7. Sistema remove ou atualiza o corredor na lista de corredores na tela.

PACOTE: Itinerários

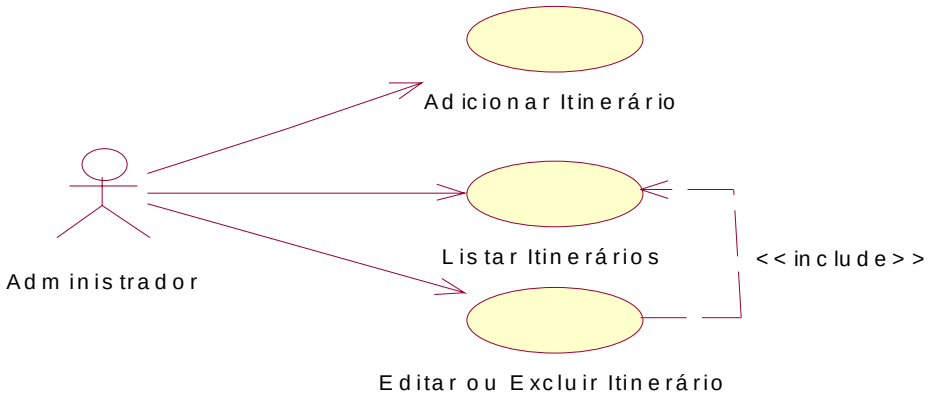

Figura 8 : Casos de Uso - Pacote Itinerários.

LISTAR ITINERÁRIOS

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todos os itinerários são listados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar itinerário no menu de operações;
2. Sistema exibe uma lista com todos os itinerários.

ADICIONAR ITINERÁRIO

Ator Principal: Administrador

Pré-Condição: Administrador autenticado.

Pós-Condição: Itinerário foi cadastrado.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar itinerário no menu de operações;
2. Administrador solicita cadastrar itinerário;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona o novo itinerário na lista de itinerários na tela.

EDITAR OU EXCLUIR ITINERÁRIO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Itinerário foi excluído ou seus dados foram editados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar itinerário no menu de operações;
2. Sistema exibe uma lista com todos os itinerários;
3. Administrador seleciona um itinerário a ser editado ou excluído;
4. Sistema exibe formulário com os dados do itinerário selecionado;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui o itinerário;
7. Sistema remove ou atualiza o itinerário na lista de itinerários na tela.

PACOTE: Veículos

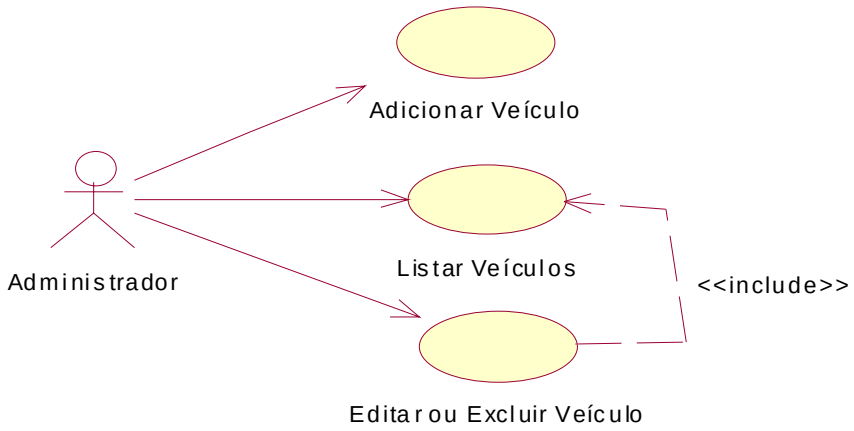

Figura 9 : Casos de Uso - Pacote Veículos.

LISTAR VEÍCULOS

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todos os veículos são listados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar veículo no menu de operações;
2. Sistema exibe uma lista com todos os veículos.

ADICIONAR VEÍCULO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Veículo foi cadastrado.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar veículo no menu de operações;
2. Administrador solicita cadastrar veículo;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona o novo veículo na lista de veículos na tela.

EDITAR OU EXCLUIR VEÍCULO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Veículo foi excluído ou seus dados foram editados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar veículo no menu de operações;
2. Sistema exibe uma lista com todos os veículos;
3. Administrador seleciona um veículo a ser editado ou excluído;
4. Sistema exibe formulário com os dados do veículo selecionado;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui o veículo, incluindo as configurações veículo-equipamento referentes ao veículo;
7. Sistema remove ou atualiza o veículo na lista de veículos na tela.

PACOTE: Paradas

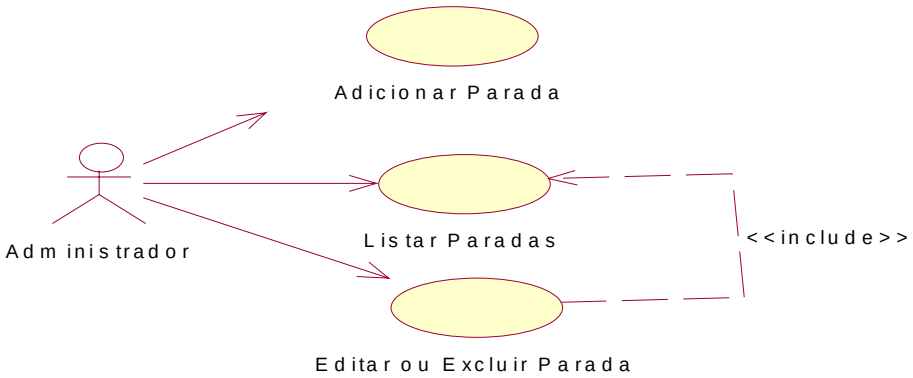

Figura 10 : Casos de Uso - Pacote Paradas.

LISTAR PARADAS

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todas as paradas são listadas.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar parada no menu de operações;
2. Sistema exibe uma lista com todas as paradas.

ADICIONAR PARADA

Ator Principal: Administrador.
Pré-Condição: Administrador autenticado.
Pós-Condição: Parada foi cadastrada.

Cenários:
Fluxo Principal
1. Administrador acessa função de administrar parada no menu de operações;
2. Administrador solicita cadastrar parada;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona a nova parada na lista de paradas na tela.

EDITAR OU EXCLUIR PARADA

Ator Principal: Administrador.
Pré-Condição: Administrador autenticado.
Pós-Condição: Parada foi excluída ou seus dados foram editados.

Cenários:
Fluxo Principal
1. Administrador acessa função de administrar parada no menu de operações;
2. Sistema exibe uma lista com todas as paradas;
3. Administrador seleciona uma parada a ser editada ou excluída;
4. Sistema exibe formulário com os dados da parada selecionada;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui a parada, incluindo os corredores e itinerários referentes à parada;
7. Sistema remove ou atualiza a parada na lista de paradas na tela.

PACOTE: Equipamentos de Rastreamento

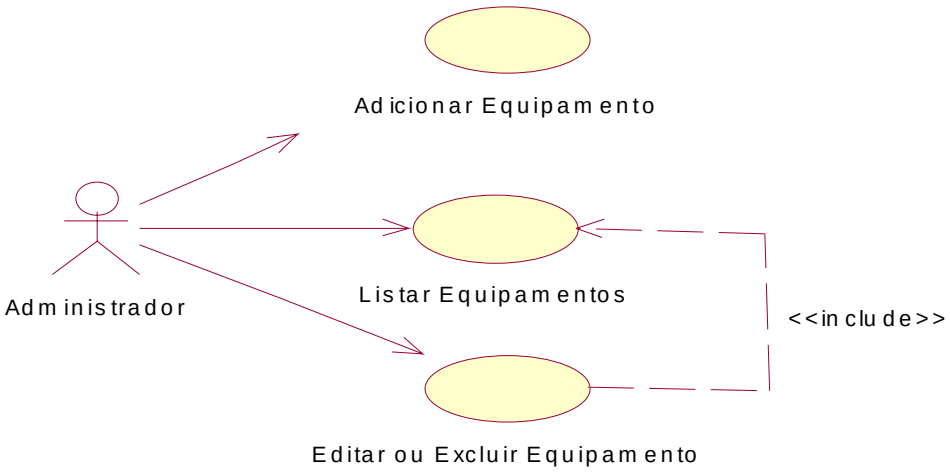

Figura 11 : Casos de Uso - Pacote Equipamentos de Rastreamento.

LISTAR EQUIPAMENTOS DE RASTREAMENTO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todos os equipamentos são listados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar equipamento no menu de operações;
2. Sistema exibe uma lista com todos os equipamentos.

ADICIONAR EQUIPAMENTO DE RASTREAMENTO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Equipamento foi cadastrado.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar equipamento no menu de operações;
2. Administrador solicita cadastrar equipamento;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona o novo equipamento na lista de equipamentos na tela.

EDITAR OU EXCLUIR EQUIPAMENTO DE RASTREAMENTO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Equipamento foi excluído ou seus dados foram editados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar equipamento no menu de operações;
2. Sistema exibe uma lista com todos os equipamentos;
3. Administrador seleciona um equipamento a ser editado ou excluído;
4. Sistema exibe formulário com os dados do equipamento selecionado;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui o equipamento, incluindo as configurações veículo-equipamento referentes ao equipamento;
7. Sistema remove ou atualiza o equipamento na lista de equipamentos na tela.

PACOTE: Configurações Veículo-Equipamento

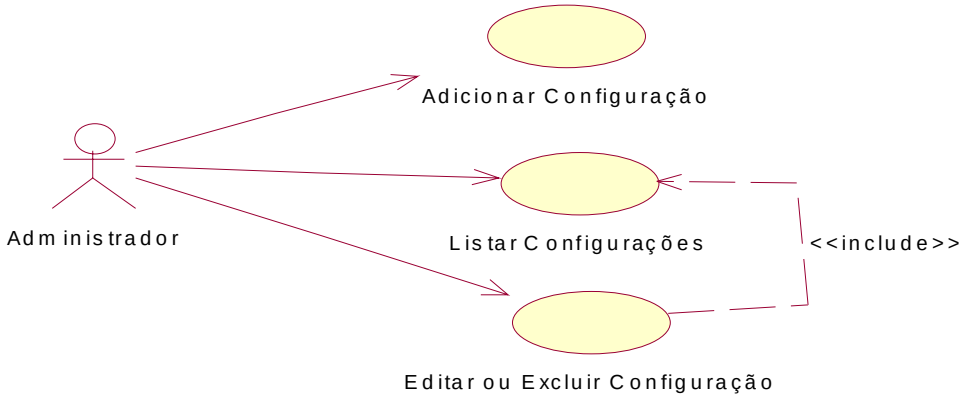

Figura 12 : Casos de Uso - Pacote Configurações Veículo-Equipamento.

LISTAR CONFIGURAÇÕES

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado no sistema.

Pós-Condição: Todas as configurações são listadas.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar configuração no menu de operações;
2. Sistema exibe uma lista com todas as configurações.

ADICIONAR CONFIGURAÇÃO

Ator Principal: Administrador

Pré-Condição: Administrador autenticado.

Pós-Condição: Configuração foi cadastrada.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar configuração no menu de operações;
2. Administrador solicita cadastrar configuração;
3. Sistema exibe formulário para preenchimento dos dados;
4. Administrador preenche os dados e submete;
5. Sistema salva os dados;
6. Sistema adiciona a nova configuração na lista de configurações na tela.

EDITAR OU EXCLUIR CONFIGURAÇÃO

Ator Principal: Administrador.

Pré-Condição: Administrador autenticado.

Pós-Condição: Configuração foi excluída ou seus dados foram editados.

Cenários:

Fluxo Principal

1. Administrador acessa função de administrar configuração no menu de operações;
2. Sistema exibe uma lista com todas as configurações;
3. Administrador seleciona uma configuração a ser editada ou excluída;
4. Sistema exibe formulário com os dados da configuração selecionada;
5. Administrador solicita exclusão ou edita as informações desejadas e submete;
6. Sistema altera os dados ou exclui a configuração;
7. Sistema remove ou atualiza a configuração na lista de configurações na tela.

PACOTE: Acesso ao Sistema

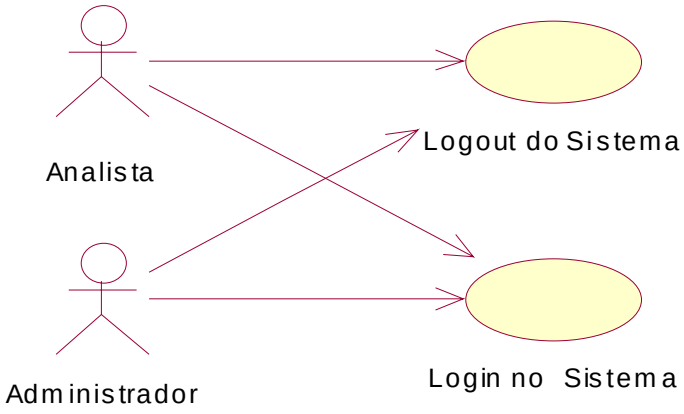

Figura 13 : Casos de Uso - Pacote Acesso ao Sistema.

LOGIN NO SISTEMA

Ator Principal: Administrador ou Analista.

Pré-Condição: Sistema acessível.

Pós-Condição: Administrador ou Analista autenticado.

Cenários:

Fluxo Principal

1. Administrador ou Analista solicita login acessando a interface do sistema;
2. Sistema exibe formulário obter “login de acesso” e senha;
3. Administrador ou Analista informa o “login” e a senha;
4. Sistema efetua login;
5. Sistema exibe interface com menu de operações.

LOGOUT DO SISTEMA

Ator Principal: Administrador ou Analista.

Pré-Condição: Administrador ou Analista autenticado no sistema.

Pós-Condição: Administrador ou Analista encerrou sua sessão no sistema.

Cenários:

Fluxo Principal

1. Administrador ou Analista solicita efetuar logout do sistema;
2. Sistema exibe mensagem para confirmar solicitação;
3. Administrador ou Analista confirma solicitação;
4. Sistema efetua logout;
5. Sistema encerra o programa.

3.3. Modelo de Classes do Domínio

Esta modelagem tem o objetivo de apresentar as classes conceituais do mundo real, identificadas a partir da análise dos casos de uso, e orientar a criação do modelo de classes de projeto, onde são aplicados os padrões de projeto.

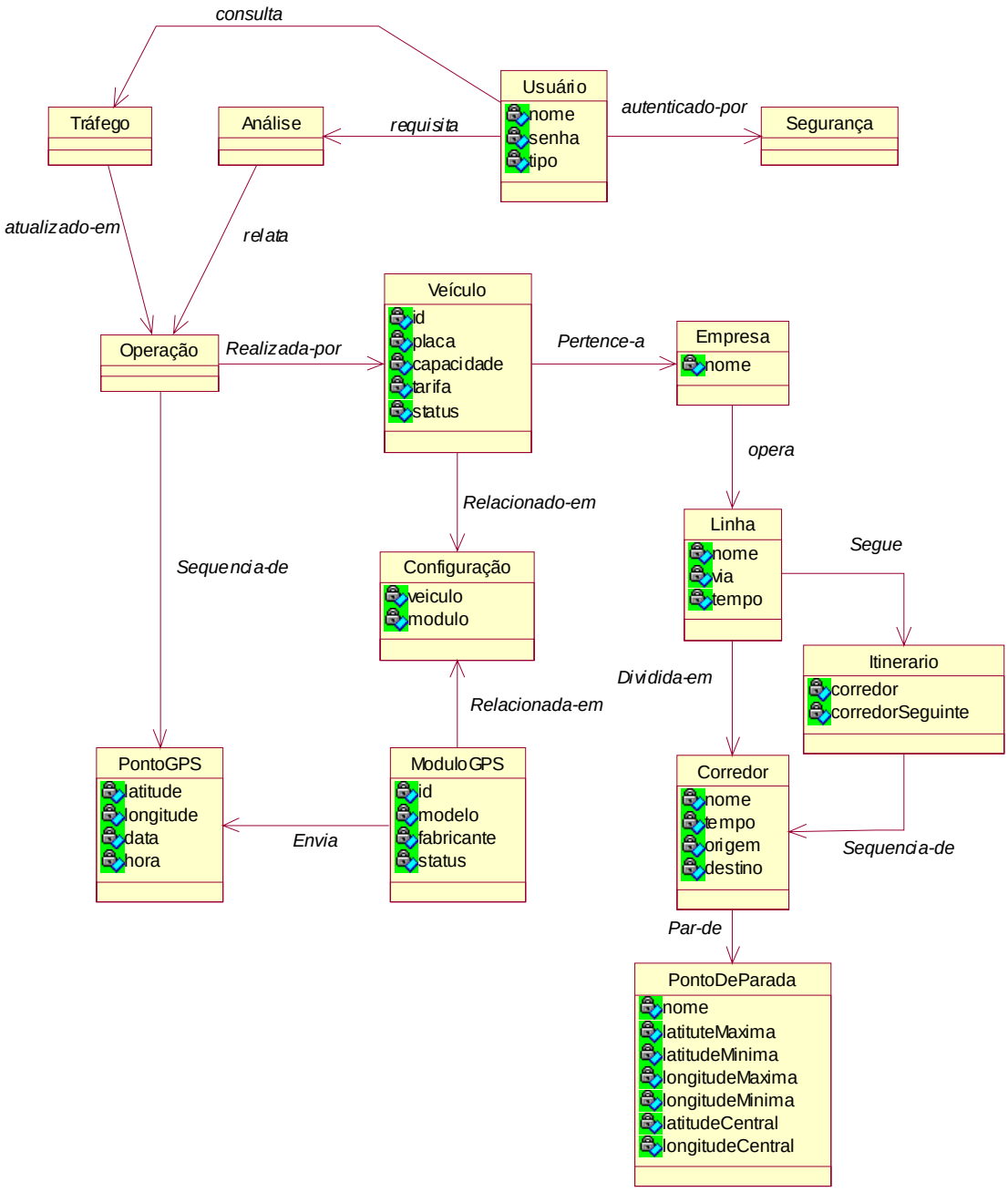

Figura 14 : Modelo do Domínio.

3.4. Projeto Arquitetural

O projeto do sistema tem início com a descrição em termos de sua organização conceitual em camadas e componentes, conforme apresentado no diagrama a seguir.

A camada do cliente inclui os componentes responsáveis pela interface com o usuário. *SIMT-Admin* permite a realização das operações de administração e análise do sistema. Este componente é implementado pelo *framework OpenSwing*, uma biblioteca de componentes gráficos. Esta camada ainda inclui os navegadores *web* e *wap*, utilizados pelos usuários para consultar as condições do tráfego e a localização dos veículos. A camada de apresentação trata das requisições originadas na camada do cliente, controlando o fluxo de trabalho e preparando as respostas para apresentação. *SIMT-W*eb e *SIMT-Wap* recebem as requisições dos respectivos navegadores *web* e *wap* e são implementados pelo *framework Struts*, um conjunto de classes de controle para aplicações *web*. A camada de negócios é responsável por realizar toda a lógica da aplicação, implementada pelo componente *SIMT-Server*. A camada de dados representa a lógica de acesso aos dados persistentes da aplicação. *MySql_Jdbc* é a interface para acesso ao banco de dados do sistema. *FirebirdSql_Jdbc* é a interface para acessar o aplicativo *GPS*.

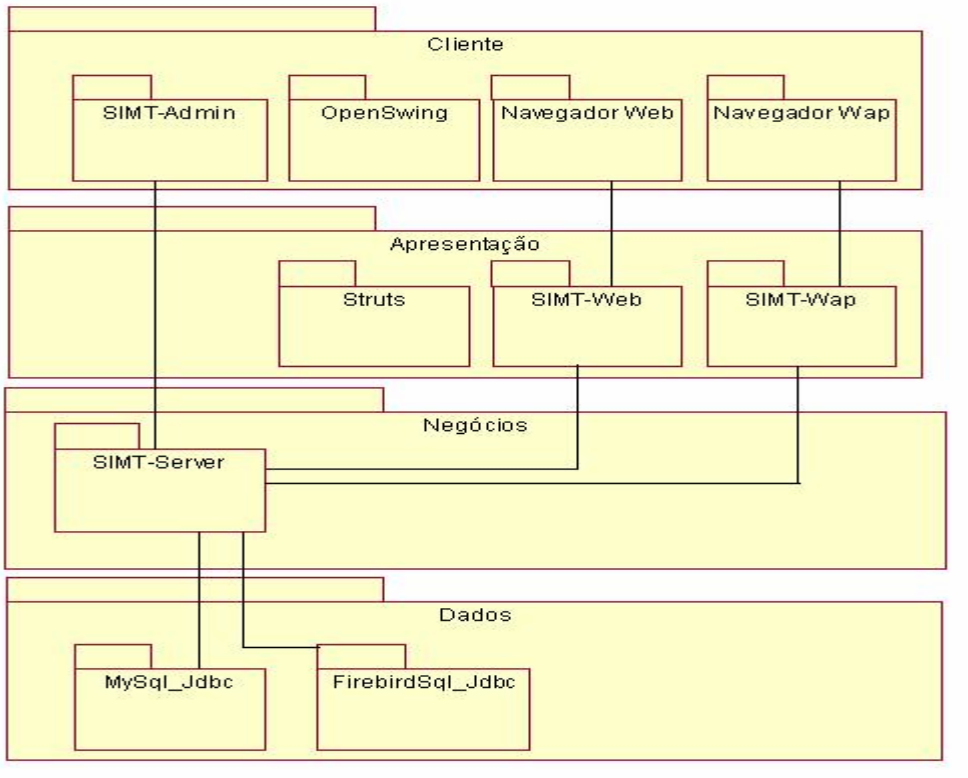

Figura 15 : Projeto Arquitetural.

3.5. Classes de Projeto

O modelo a seguir apresenta as classes que serão utilizadas na implementação do sistema. As classes de projeto são criadas junto com os diagramas de interação (seção 3.6), durante a realização dos casos de uso. Ambas as tarefas se baseiam no modelo de classes do domínio da seção 3.3 e nos padrões de projeto.

As classes são apresentadas seguindo o modelo de camadas, padrão arquitetural utilizado pelo sistema. Em cada camada as classes são agrupadas em pacotes, de acordo com a coesão funcional e menor acoplamento.

Camada de Negócios

Pacote modelo

Este pacote contém as classes que representam conceitos essenciais do domínio do problema, definindo objetos de valor, sendo utilizadas pelas classes de todas as outras camadas. Um *Value Object* (Objeto de Valor) é uma classe Java serializável que permite transportar todos os dados de seus atributos através de uma única chamada remota. È um padrão para reduzir o tráfego de rede.

Para ser enviado pela rede através do protocolo *RMI-IIOP*, um objeto é serializado, ou seja, é convertido em um formato binário reversível que preserva informações e o estado dos seus dados. Todas as classes deste pacote são serializáveis, pois implementam a interface *java.io.Serializable*. Os métodos *getters* e *setters* foram omitidos por simplificação.

Figura 16 : Classes de Projeto – Pacote *modelo*.

Pacote util

As classes deste pacote tratam de rotinas para operações recorrentes, podendo ser reutilizadas em outros projetos. Assim como no pacote *modelo*, estas classes são utilizadas por todas as outras camadas.

A classe *CriptoUtils* possui dois métodos, sendo um para criptografar e outro para descriptografar as senhas dos usuários.

ServiceLocator é a classe utilizada para centralizar a localização de componentes *EJBs* e fontes de dados em aplicações distribuídas, a partir do nome de registro no serviço de nomes (*JNDI*) existente no servidor de aplicação. Seus métodos permitem obter uma referência de um objeto remoto ou local, além de um *pool* de conexões com o banco de dados (*DataSource*).

Para facilitar as operações no algoritmo de pesquisa, usa-se a medida de tempo em segundos, embora os resultados devam ser apresentados no formato de horas e minutos. A classe *DataHora* possui um método para cada conversão.

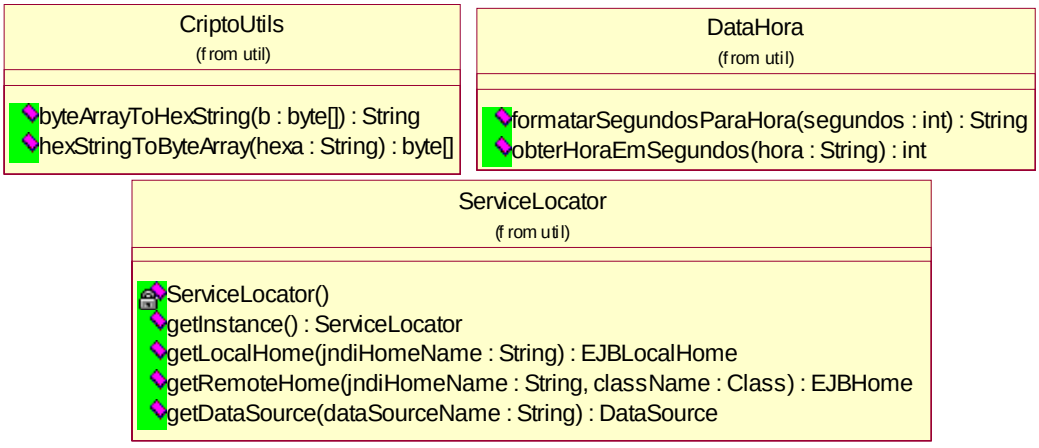

Figura 17 : Classes de Projeto – Pacote *util*.

Pacote processo

As classes que representam casos de uso são separadas neste pacote. Elas apresentam a interface da camada de negócios e são responsáveis pelo fluxo de execução dos casos de uso.

As classes que implementam o padrão *Business Delegate* (*AdminDelegate*, *AnaliseDelegate* e *TrafegoDelegate*) servem para encapsular os detalhes da lógica de negócios. Seus métodos apenas delegam responsabilidades, repassando-as para os respectivos métodos nas classes *Session Façade*, tornando chamadas remotas transparentes ao cliente. Uma *façade* (fachada) é um padrão para centralizar o controle e expor uma interface uniforme do sistema, utilizando as classes do domínio (*core*) para executar os serviços.

TrafegoSessionFaçade possui um método para atualizar os dados de operação dos veículos e outros três métodos para obter as listas de resultados. Também possui outros métodos que auxiliam na criação do formulário *web* de consulta.

AnaliseSessionFaçade possui um método de pesquisa para cada tipo de análise existente, cada um retornando uma lista de objetos de valor, cujas classes são do pacote *modelo*. Além deles, existem outros métodos para listagem de valores que são utilizados nos formulários da aplicação desktop.

AdminSessionFaçade possui métodos para listagem, atualização, remoção e adição das diversas entidades no banco de dados, referenciadas na figura por *XXX* para simplificação. Possui também um método para autenticação de usuário.

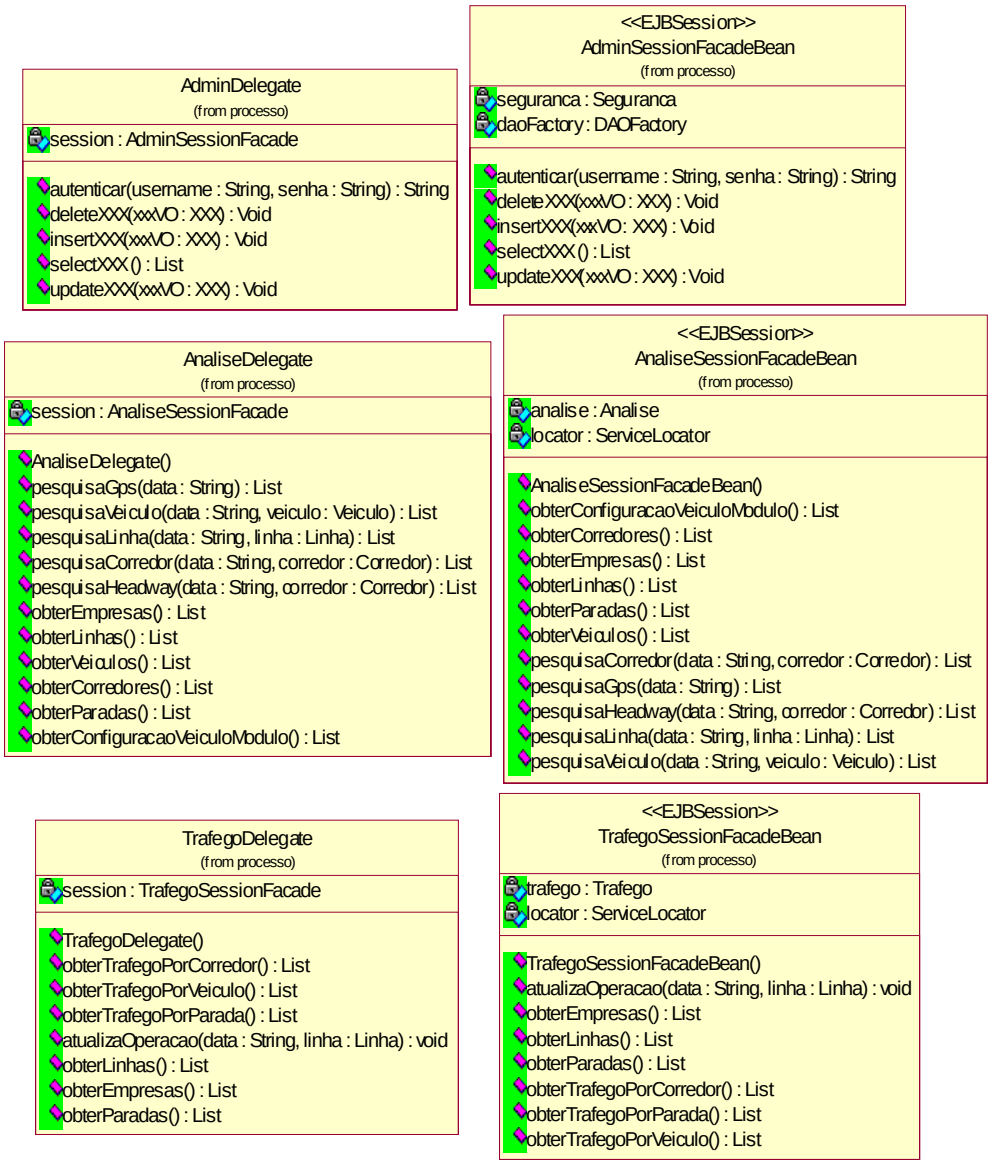

Figura 18 : Classes de Projeto – Pacote *processo*.

Pacote core

As classes neste pacote são responsáveis por implementar as operações básicas do sistema. Tais classes estão associadas aos conceitos que elas representam e não aos casos de uso que os contêm. O pacote é subdividido para melhor organização e distribuição de responsabilidades.

As classes *Tráfego* e *Análise* utilizam uma das implementações da interface *IOperacao*, instanciada através do método *setOperacao* pelo respectivo nome da classe.
Os demais métodos são para realizar as respectivas pesquisas e para obter as listas de resultados.

Seguranca possui métodos para autenticação do usuário e verificação do seu tipo. Possui também um método para evitar que seja adicionado um usuário cujo nome já exista. Outro método permite recuperar um usuário pelo seu nome.

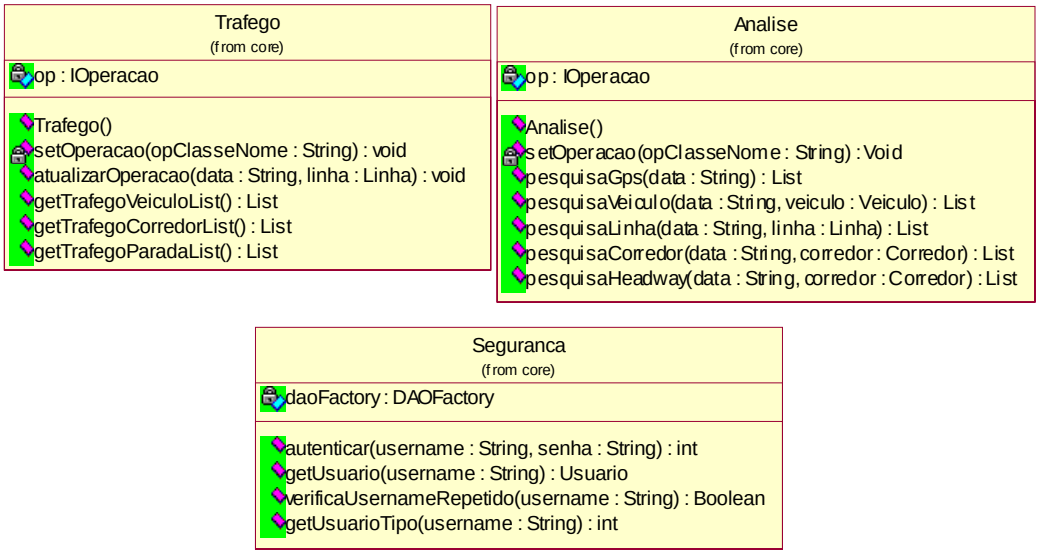

Figura 19 : Classes de Projeto – Pacote *core*.

Pacote core.op

Este pacote contém a interface *IOperacao* e todas as classes que a implementam. Cada linha de transporte possui seu próprio algoritmo de pesquisa implementado em classe separada.

Os métodos de pesquisa do tráfego e de análise implementam o algoritmo apresentado no diagrama de interação da seção 3.6. Ao final do algoritmo as respectivas listas estão preenchidas com os resultados. Para obter as listas basta invocar o respectivo método *get*.

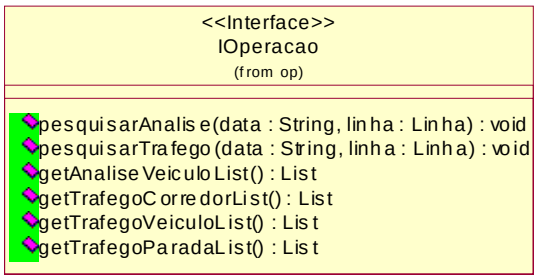

Figura 20 : Classes de Projeto – Pacote *core.op*.

Pacote core.vlh

As classes que implementam o padrão *Value List Handler* são alocadas neste pacote. Este padrão de projeto possibilita um *cache* de resultados, reduzindo o acesso à fonte de dados e melhorando o desempenho da rede. A interface *ValueListIterator* define os métodos básicos de iteração nas listas e é implementado pela classe *ValueListHandler*, que possui uma lista e um *iterator* como atributos. As operações utilizam a superclasse *Object* para que a lista possa conter objetos de qualquer tipo de classe Java. Todas as demais classes estendem *ValueListHandler*, o que facilita a implementação de novas classes neste padrão. Algumas delas são implementadas como componentes de sessão (*session beans*), permitindo que uma mesma instância seja reutilizada e o acesso aos dados seja mais rápido.

O método *executeSearch* é o responsável por acessar o respectivo *DAO* e preencher a lista com os objetos de valor. As listas de pesquisa de tráfego e análise não possuem este método, pois são listas preenchidas a partir de outras classes e não do banco de dados. Os métodos de seleção (*select*) permitem recuperar um objeto de valor da lista a partir de um código ou de outro atributo.

TrafegoListHandler possui o método *atualizaLista* para garantir que apenas o índice de congestionamento mais recente será adicionado à lista. Já o método *verificaCorredorOrigemDestino* da classe *CorredorListHandler* é utilizado para identificar o corredor de uma linha pelas paradas origem e destino.

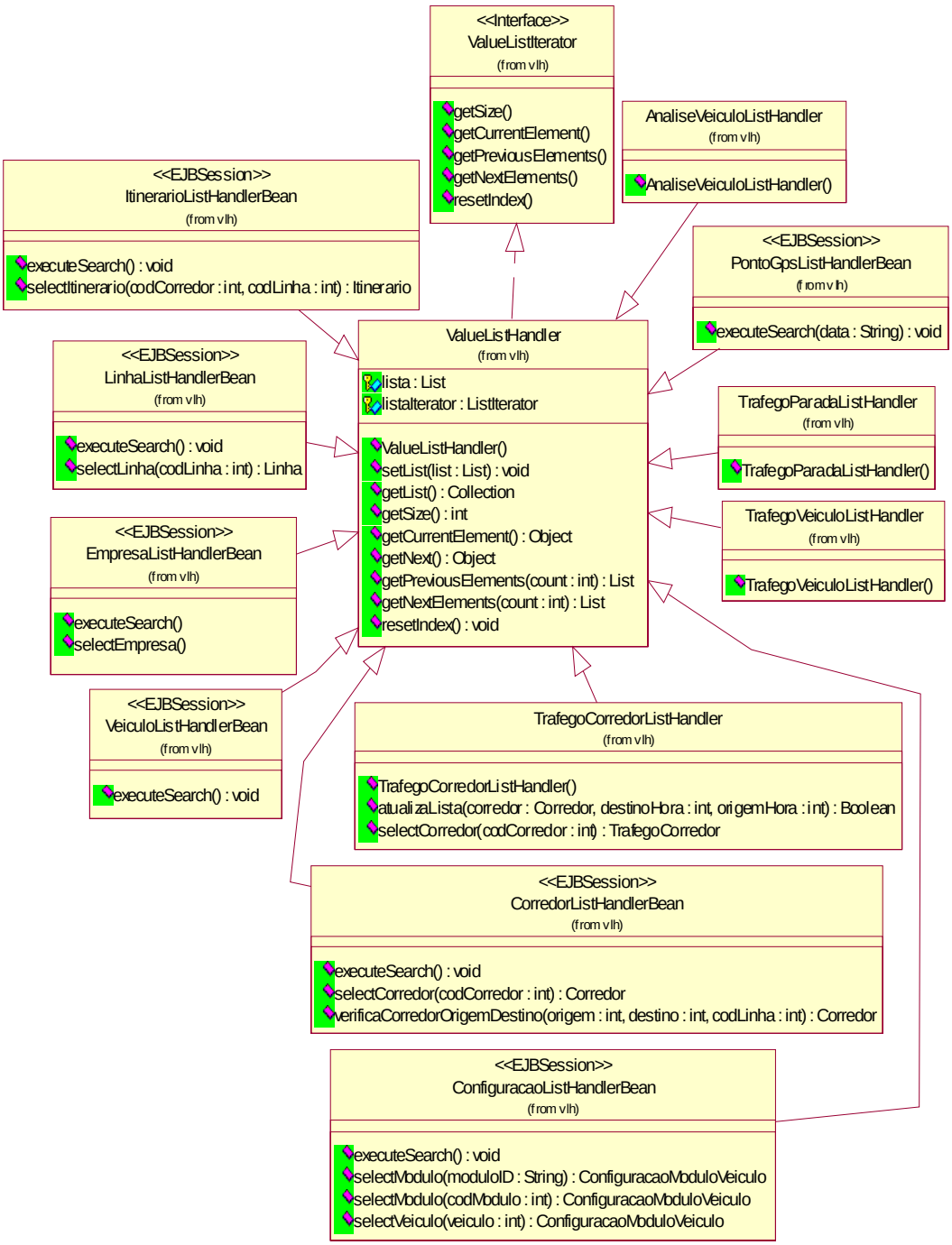

Figura 21 : Classes de Projeto – Pacote *core.vlh*.

Camada de Dados

Pacote dao.interfaces

DAO (*Data Access Object*) é um padrão que centraliza todo acesso aos dados e esconde as características de uma fonte de dados específica. Para realizá-lo, este pacote contém as interfaces a serem implementadas para cada nova fonte de dados usada, viabilizando a substituição de uma implementação por outra. Cada interface define uma operação de adição, de remoção, de atualização e de seleção. A fonte com os dados enviados pelo aparelho *GPS* oferece apenas um único método de listagem.

Figura 22 : Classes de Projeto – Pacote *dao.interfaces*.

Pacote dao

Neste pacote estão as implementações das interfaces de cada *DAO*.

Pacote dao.factory

No sistema existem duas fontes de dados, o banco *MySQL* e o banco *FirebirdSql*. Cada uma requer uma classe *factory*, que permita ao cliente obter instâncias das classes *DAO* específicas da fonte. *DAOFactory* define uma interface para instanciar classes *factory*, facilitando a adição de novas fontes de dados.

FirebirdGeocontrolDAOFactory é a classe *factory* do banco *FirebirdSql* e apenas o método *getPontoGpsDAO* retorna o *DAO* para acesso aos registros enviados pelo aparelho *GPS*. Os outros métodos retornam nulo.

MysqlSimtDAOFactory é a classe *factory* do banco *MySQL*. Cada método retorna um *DAO* para a respectiva tabela no banco. Apenas o método *getPontoGpsDAO* retorna nulo, pois os registros estão no outro banco.

Cada classe *factory* possui seu próprio método para obter uma conexão com o banco de dados, pois bancos diferentes possuem interfaces de comunicação diferentes.

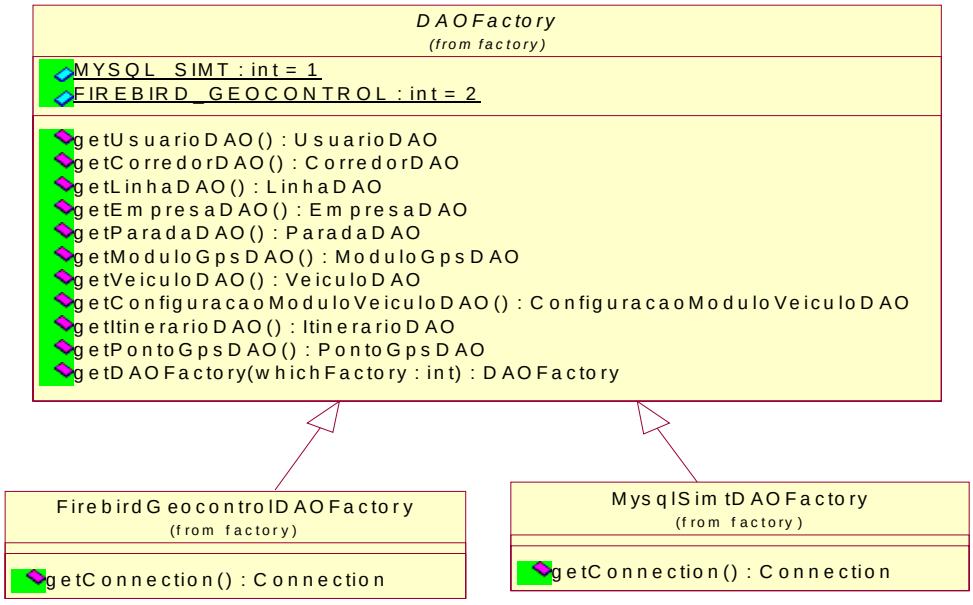

Figura 23 : Classes de Projeto – Pacote *dao.factory*.

Camada de Apresentação

Pacote web e Pacote wap

O funcionamento desta camada se baseia no *framework Struts*. Este possui a classe controladora *ActionServlet*, que recebe as requisições *HTTP* e as despacha para classes de comando criadas pelo desenvolvedor. Estas classes de comando devem extender a classe *Action* e implementar o método *execute*. Este método é automaticamente chamado pelo controlador para processar a requisição. Os detalhes das classes *Action* e *ActionServlet*, além de outras do *framework*, podem ser encontrados na documentação do *Struts*.

Nas classes *TrafegoAction*, o método *execute* não possui nenhuma lógica de negócios, apenas invoca os serviços da camada de negócios, através da classe *TrafegoDelegate*, controla o fluxo de execução e repassa os resultados para serem exibidos. A diferença entre as classes nos dois pacotes está no formato das respostas geradas. No pacote *wap* a resposta será exibida em celulares e no pacote *web* será vista nos navegadores *web*.

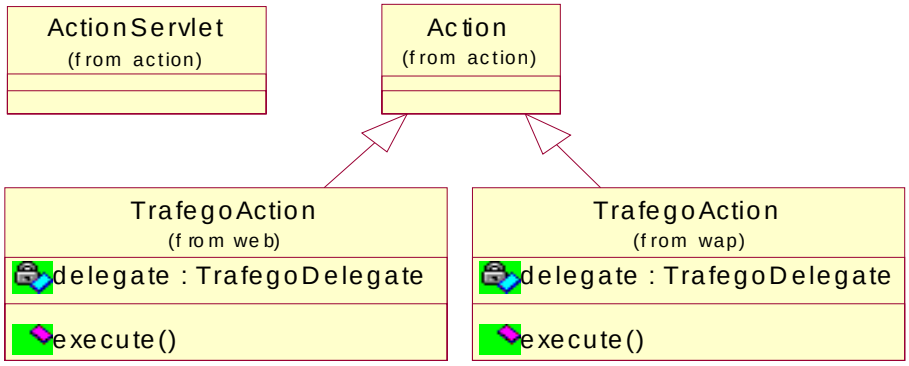

Figura 24 : Classes de Projeto – Pacotes *web* / *wap*.

Camada do Cliente

Pacote admin

Nesta camada o *framework OpenSwing* fornece um conjunto de classes para facilitar a implementação do padrão MVC (*Model-View-Controller*), que separa os dados (*Model*) da interface do usuário (*View*) e da lógica de negócios (*Control*). Ele se baseia na classe *MDIFrame* e no controlador *MDIController* para criar aplicações onde existe uma janela principal que permite ao usuário abrir múltiplas janelas internas. Toda janela aberta no interior de um *MDIFrame* deve estender *InternalFrame* para que seja gerenciado pelo controlador. O detalhamento destas e outras classes do *framework* pode ser obtido na documentação do *OpenSwing*.

SimtAdminApplication é a classe controladora da janela principal da aplicação. Ele implementa uma série de métodos do controlador *MDIController* que permitem customizar a janela principal da aplicação. Após realizar a autenticação do usuário, o controlador cria uma instância da janela principal (*MDIFrame*).

SimtAdminFacade implementa a interface *ClientFacade*, fornecida pelo *framework*. Esta interface tem por objetivo agrupar numa classe todos os métodos que serão invocados quando se acessa um item no menu da aplicação. Os métodos implementados invocam os respectivos controladores das janelas internas que serão exibidas.

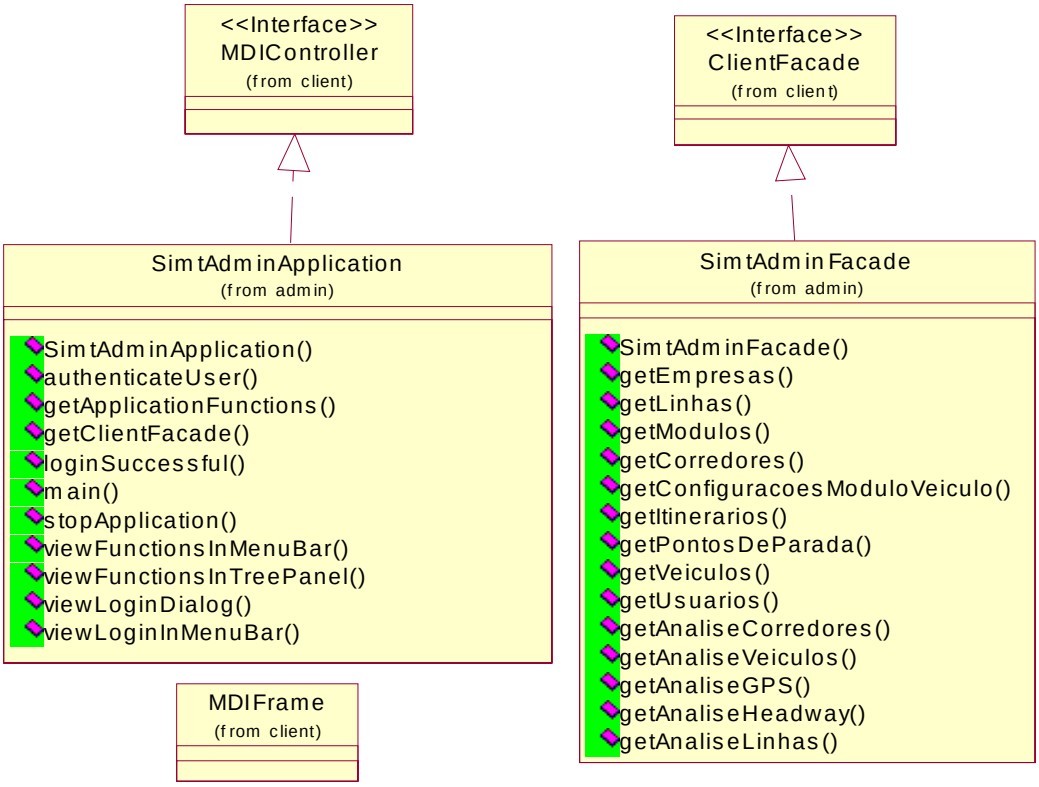

Figura 25 : Classes de Projeto – Pacote *admin*.

Pacote admin.vo

Neste pacote estão as classes de objetos de valor. São as mesmas classes do pacote *modelo*, com a diferença de que neste pacote elas estendem a classe *ValueObject* fornecida no *OpenSwing*. Isto é necessário, pois todas as operações realizadas pelos controles gráficos do *framework* referenciam *ValueObject*, que funciona como uma interface para a adição de novas classes ao aplicativo.

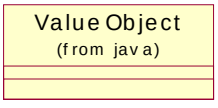

Figura 26 : Classes de Projeto – Pacote *admin.vo*.

Pacote admin.form e admin.grid

Cada janela exibida na aplicação desktop possui uma classe que estende *InternalFrame*. Se a janela possui um formulário, ela referencia um *Form* e utiliza uma classe controladora que estende *FormController*. Se possuir um grid, ela referencia um *GridControl* e utiliza uma classe controladora que estende *GridController*.

GridController e *FormController* possuem métodos que devem ser implementados nas classes que os estendem. Tais métodos acessam os serviços do sistema através das classes *delegates*.

Todos os controladores nestes pacotes são instanciados a partir da classe *SimtAdminFacade*, quando o usuário acessa um menu da aplicação.

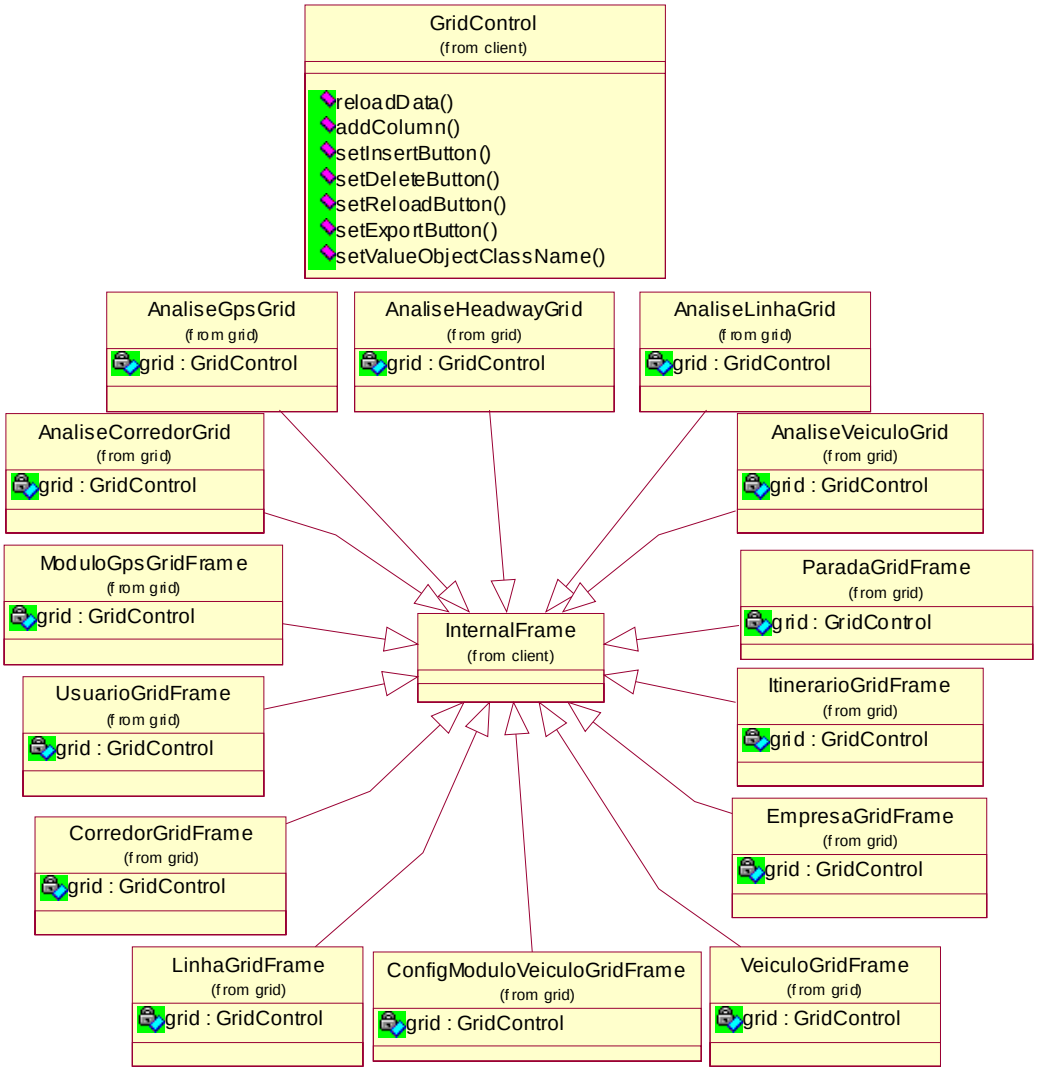

Figura 27 : Classes de Projeto – Pacotes *admin.form / admin.grid*.

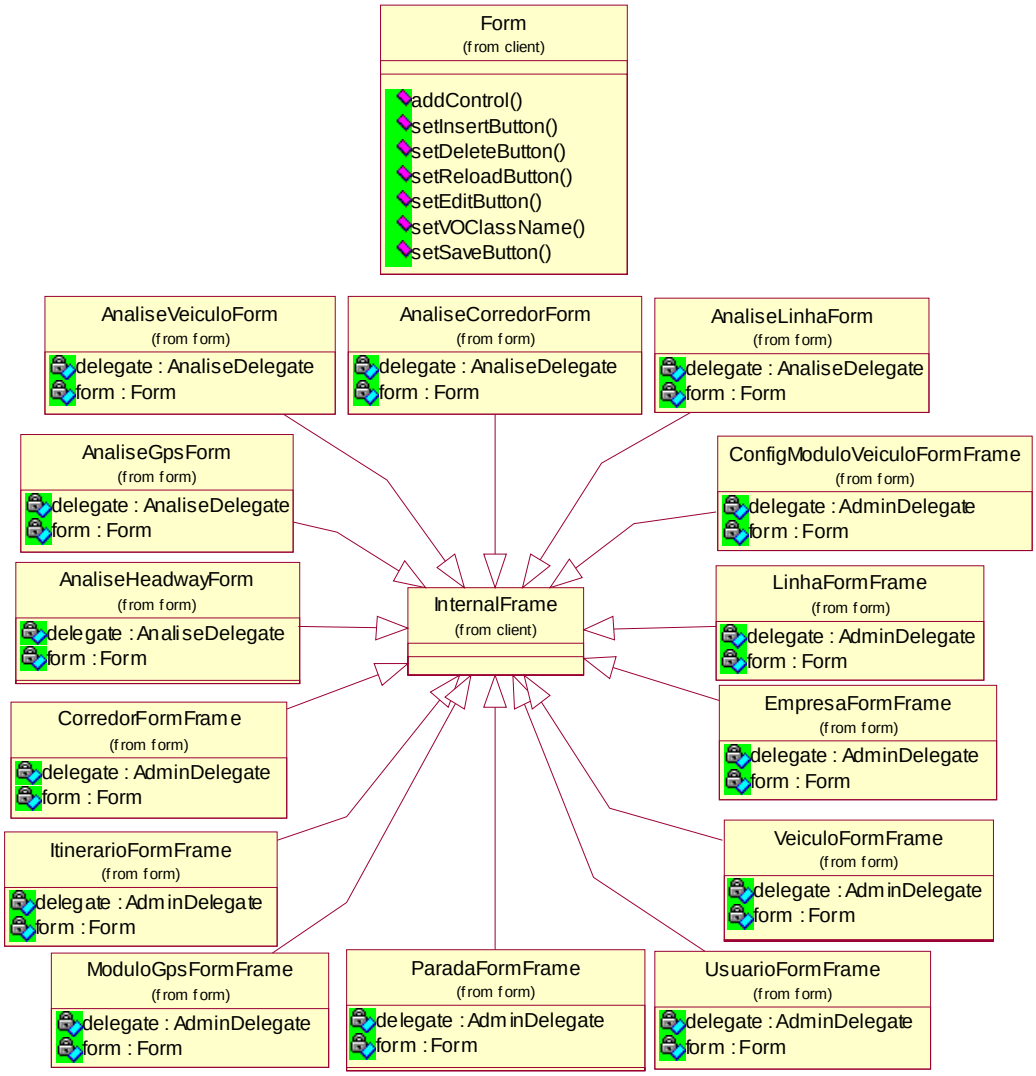

Figura 28 : Classes de Projeto – Pacotes *admin.form / admin.grid*.

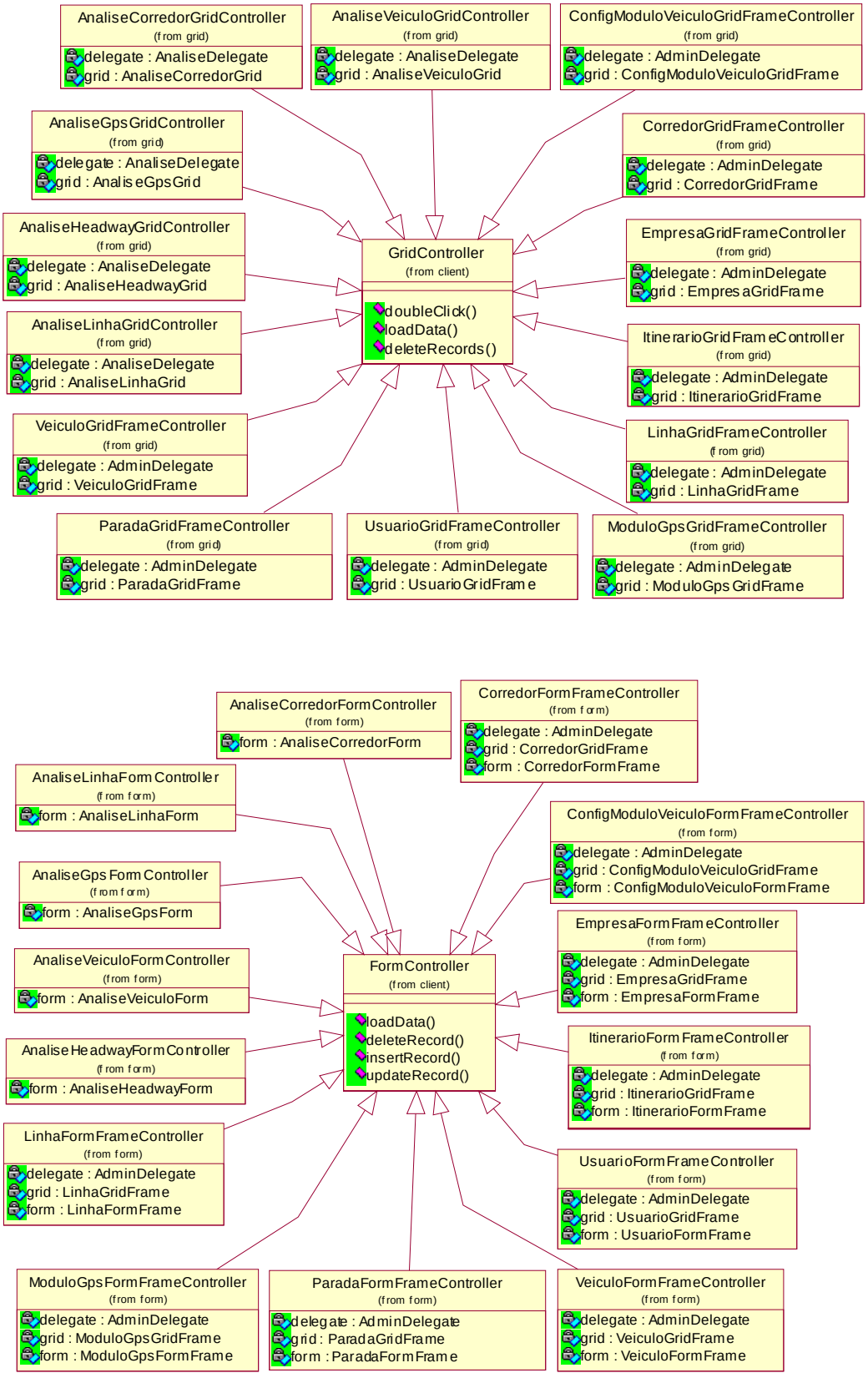

Figura 29 : Classes de Projeto – Pacotes *admin.form / admin.grid*.

3.6. Diagramas de Interação

Os diagramas de interação identificam conceitos e responsabilidades, descrevendo como um caso de uso é realizado em termos de classes de projeto que colaboram entre si. Para cada diagrama a seguir são descritos os principais passos envolvidos na realização das operações do sistema.

As operações de listar, adicionar, editar e excluir possuem a mesma seqüência de passos para cada entidade de dados administrada pelo sistema. Por simplificação, nestes diagramas foi utilizada a terminologia *XXX* para referenciar os diferentes nomes das entidades e das classes de projeto. Esta terminologia também foi utilizada nas operações de análise, onde a seqüência de passos é a mesma para as análises de *headway*, corredor, veículo, linha e *GPS*.

O último diagrama de interação não corresponde a um caso de uso, apenas descreve a lógica de pesquisa das classes que implementam a interface *IOperacao*, muito importante no funcionamento do sistema.

ACESSO AO SISTEMA

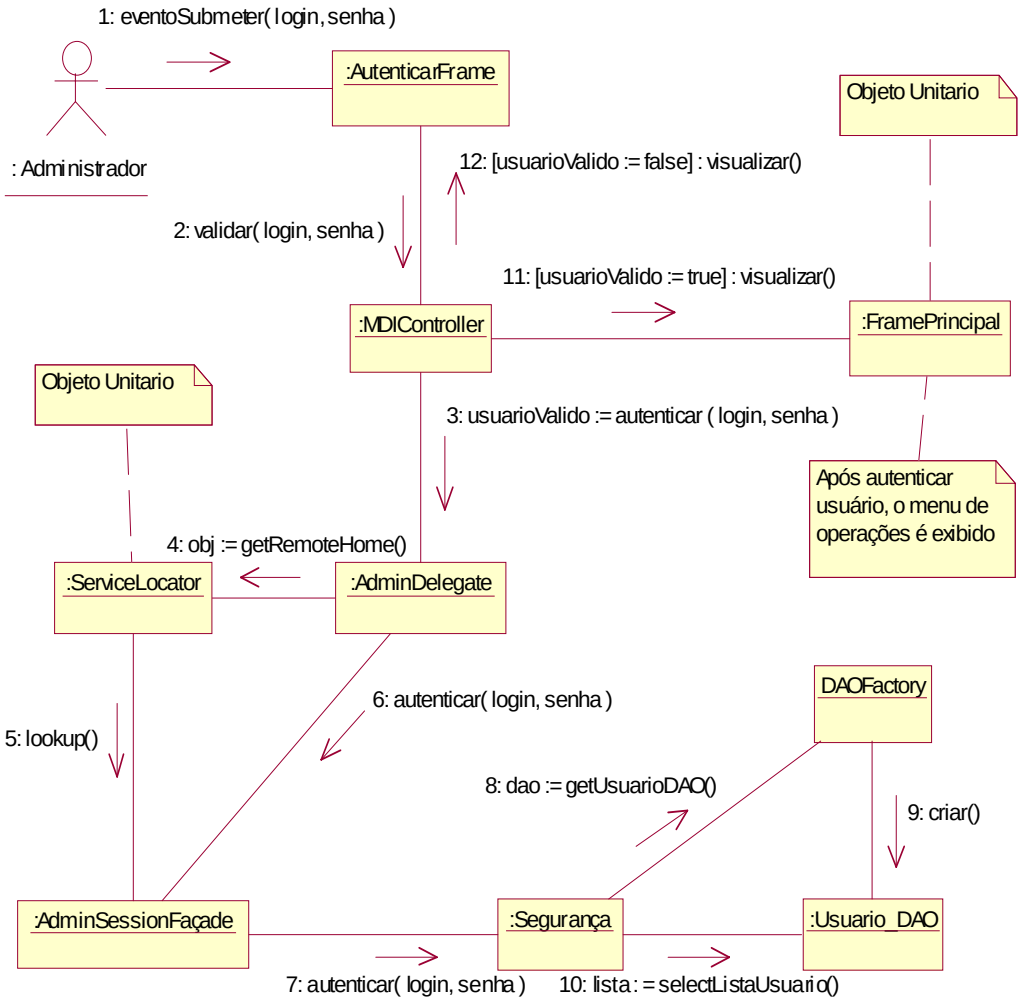

Figura 30 : Diagrama de Interação – Login no Sistema.

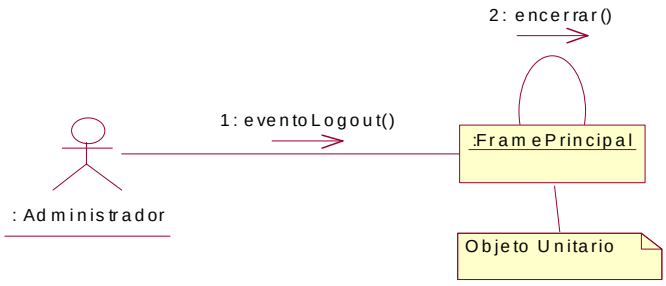

Figura 31 : Diagrama de Interação – Logout do Sistema.

De acordo com a figura 30:

- 1 - Ao ser iniciado, o programa exibe uma janela, representada pela classe *AutenticarFrame*, para que o administrador possa digitar seu login e sua senha;
- 2 e 3 – Os dados são submetidos e o evento é tratado pela classe *MDIController*. Este controlador, parte do *framework OpenSwing*, invoca o método *autenticar* da classe *AdminDelegate*;
- 4 e 5 – *AdminDelegate* obtém uma referência remota da classe *AdminSessionFaçade* através da classe *ServiceLocator*;
- 6 e 7 – *AdminDelegate* repassa a requisição para a fachada *AdminSessionFaçade*. Ela utiliza a classe *Segurança* para realizar a lógica de validação do login e senha;
- 8, 9 e 10 - A classe *Segurança* obtém uma instância da classe *Usuario_DAO* a partir da classe *DAOFactory* . *Usuario_DAO* retorna uma lista com as informações de todos os usuários, para que possam ser validados;
- 11 e 12 – Quando o controle volta à classe *MDIController*, esta verifica a validade do usuário. Se o usuário for válido a janela principal do programa é exibida. Caso contrário a janela de login continua a ser exibida.

De acordo com a figura 31:

- Para realizar o logout, o administrador acessa um menu na janela principal do programa e seleciona a opção de sair do programa. Este evento faz com que todas as janelas sejam fechadas e o programa seja encerrado.

ANÁLISE

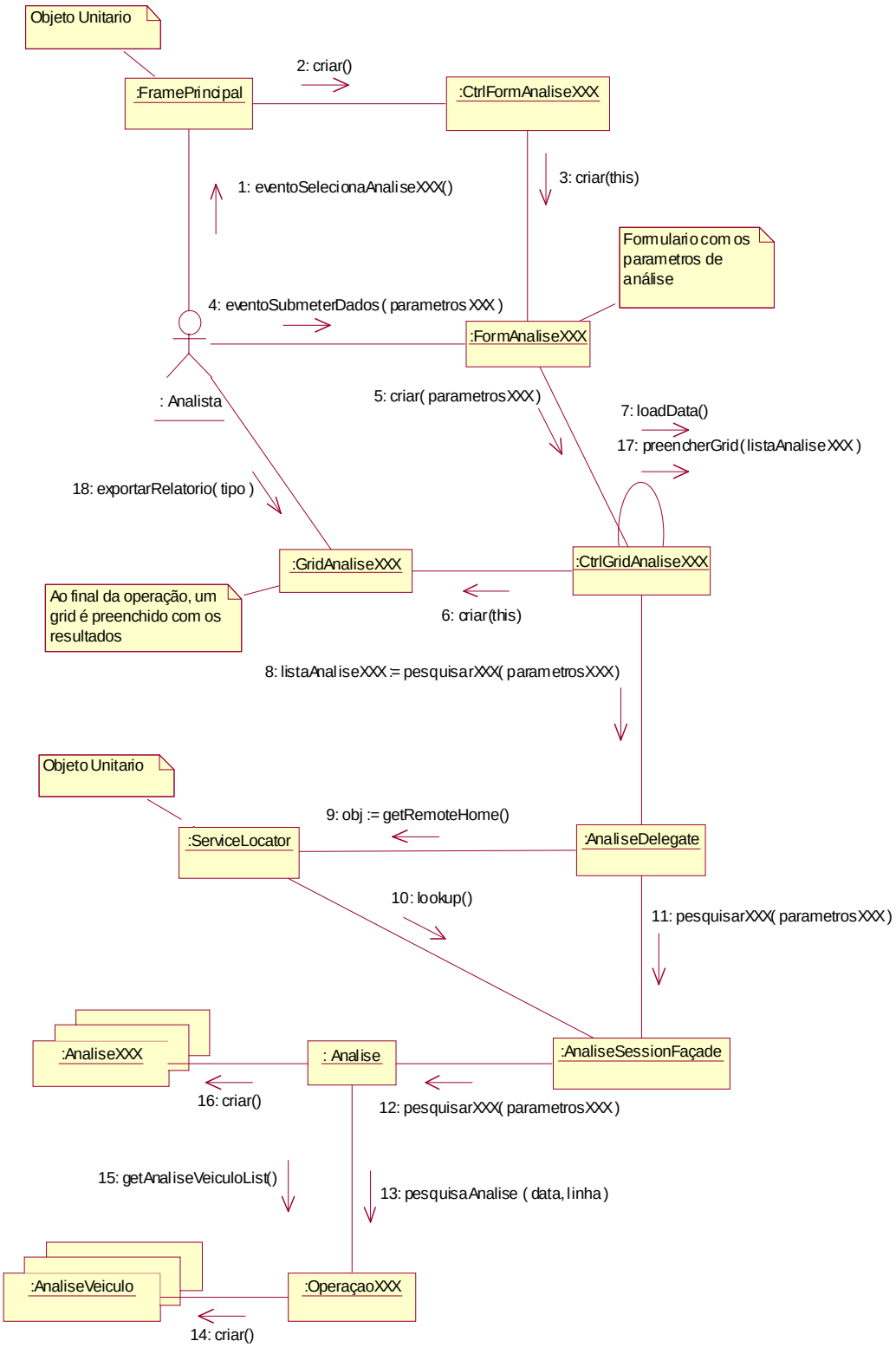

Figura 32 : Diagrama de Interação – Análise.

De acordo com a figura 32:

- 1, 2 e 3 – Após realizar o login no sistema, o analista acessa o menu na janela principal do programa e seleciona o tipo de análise que desejar. A classe *FramePrincipal* cria o respectivo controlador *CtrlFormAnaliseXXX*, que por sua vez cria a respectiva janela *FormAnaliseXXX*;
- 4 - O analista preenche o formulário com os parâmetros de análise e submete. Tais parâmetros são representados por *parametrosXXX*;
- 5 e 6 – Ao submeter, cria-se o *CtrlGridAnaliseXXX*, responsável pela criação do *GridAnaliseXXX*;
- 7 e 8 - *CtrlGridAnaliseXXX* invoca seu método para carregar os dados que serão exibidos pelo grid. Este método invoca a classe *AnaliseDelegate* para obter a lista de valores;
- 9 e 10 – *AnaliseDelegate* obtém uma referência remota da classe *AnaliseSessionFaçade* através da classe *ServiceLocator*;
- 11 e 12 – *AnaliseDelegate* repassa a requisição para a fachada *AnaliseSessionFaçade*. Esta utiliza a classe *Analise*, responsável pela lógica de análise;
- 13 e 14 - A lógica de *Analise* inicia ao invocar a respectiva classe *OperaçãoXXX*, de acordo com *parametrosXXX*. Seu método cria uma lista de objetos *AnaliseVeiculo*, contendo a seqüência de corredores, horários e índices de congestionamento de todos os veículos que percorreram a linha a ser analisada. Esta lista serve de base para a classe *Analise* realizar os outros tipos de análise;
- 15 e 16 - *Analise* obtém a lista de objetos *AnaliseVeiculo* e realizar um *looping* nesta lista. Ao final ela retorna da lista de objetos *AnaliseXXX*, de acordo com a opção de análise escolhida;
- 17 e 18 - *CtrlGridAnaliseXXX* recebe a lista de resultados e a utiliza para preencher seu grid com a lista de valores. O analista então tem a opção de exportar o grid para outro formato que desejar.

TRÁFEGO

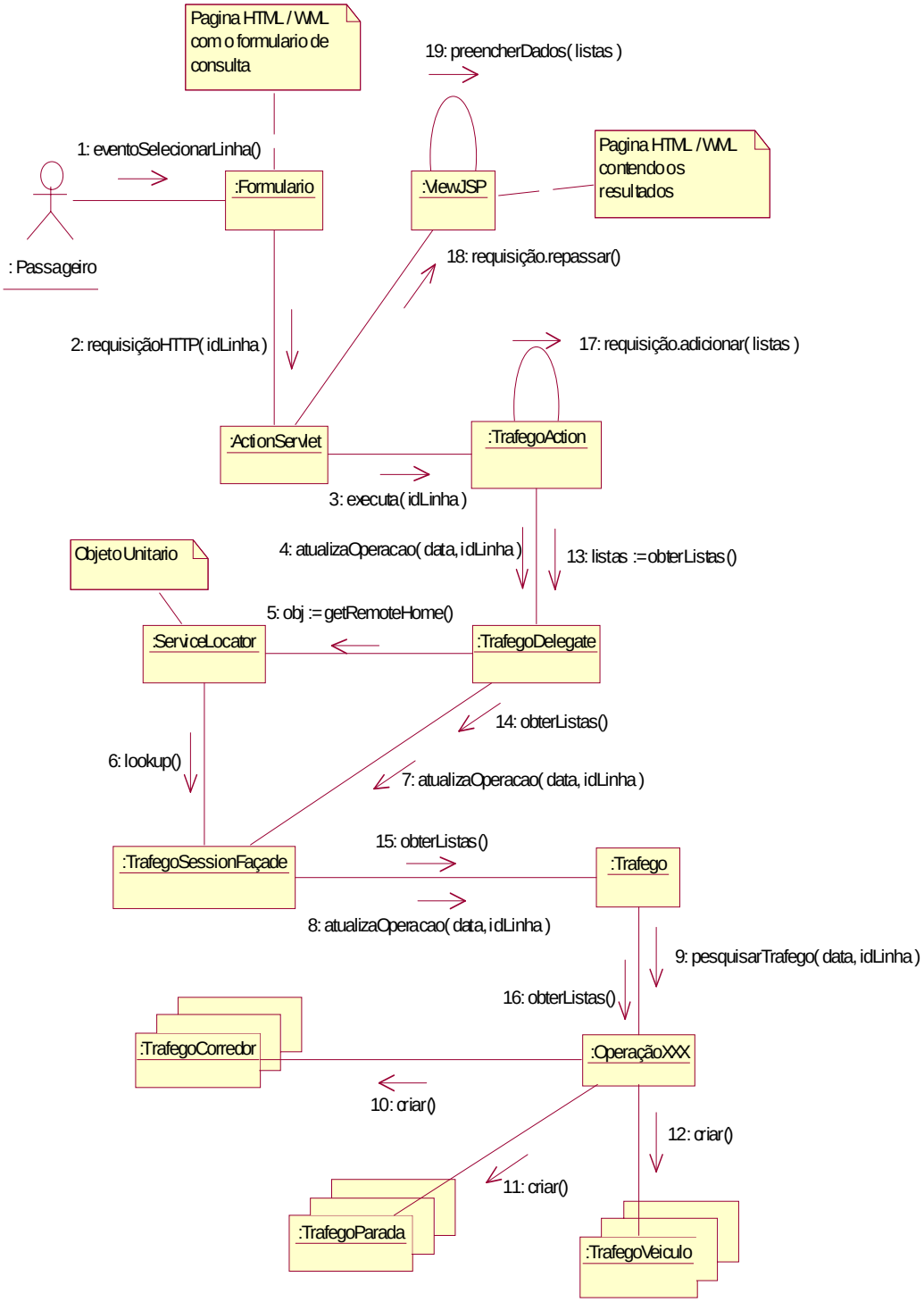

Figura 33 : Diagrama de Interação – Tráfego.

De acordo com a figura 33:

- 1, 2, 3 e 4 – O passageiro acessa uma página *web* e visualiza um formulário de consulta. Ele seleciona uma linha de transporte e faz a opção pela consulta das condições do tráfego ou pela localização dos veículos. Uma requisição *HTTP* é enviada a classe *ActionServlet*, parte do *framework Struts*, que executa uma classe responsável por tomar a ação adequada, no caso, a classe *TrafegoAction*. *TrafegoAction* então utiliza os serviços do sistema, invocando os métodos da classe *TrafegoDelegate*;
- 5 e 6 – *TrafegoDelegate* obtém uma referência remota da classe *TrafegoSessionFaçade* através da classe *ServiceLocator*;
- 7 e 8 – *TrafegoDelegate* repassa a requisição para a fachada *TrafegoSessionFaçade*. Esta utiliza a classe *Trafego*, responsável pela lógica de pesquisa do tráfego;
- 9, 10, 11 e 12 - A lógica de *Trafego* inicia ao invocar a respectiva classe *OperaçãoXXX*, de acordo com a linha selecionada. Seu método cria as listas de objetos *TrafegoVeiculo*, *TrafegoCorredor* e *TrafegoParada*. As listas de objetos *TrafegoCorredor* e *TrafegoParada* permitem a consulta das condições do tráfego. A lista de objetos *TrafegoVeículo* permite a localização dos veículos;
- 13, 14, 15 e 16 – Ao terminar o método de pesquisa do tráfego, *TrafegoAction* solicita todas as listas com os resultados;
- 17 e 18 – Após obter as listas com os resultados, a classe *TrafegoAction* salva estas listas no escopo da requisição. O *ActionServlet*, ao receber o controle de volta, repassa a requisição para a *ViewJSP* apropriada;
- 19 – *ViewJSP* é um arquivo *JSP (Java Server Page)* que gera a página *HTML* de resposta, utilizando as listas, de acordo com a opção de pesquisa. Cada opção de pesquisa, seja condições do Tráfego ou localizar veículo, possui sua própria *ViewJSP* de resposta.

ADMINISTRAÇÃO – LISTAR

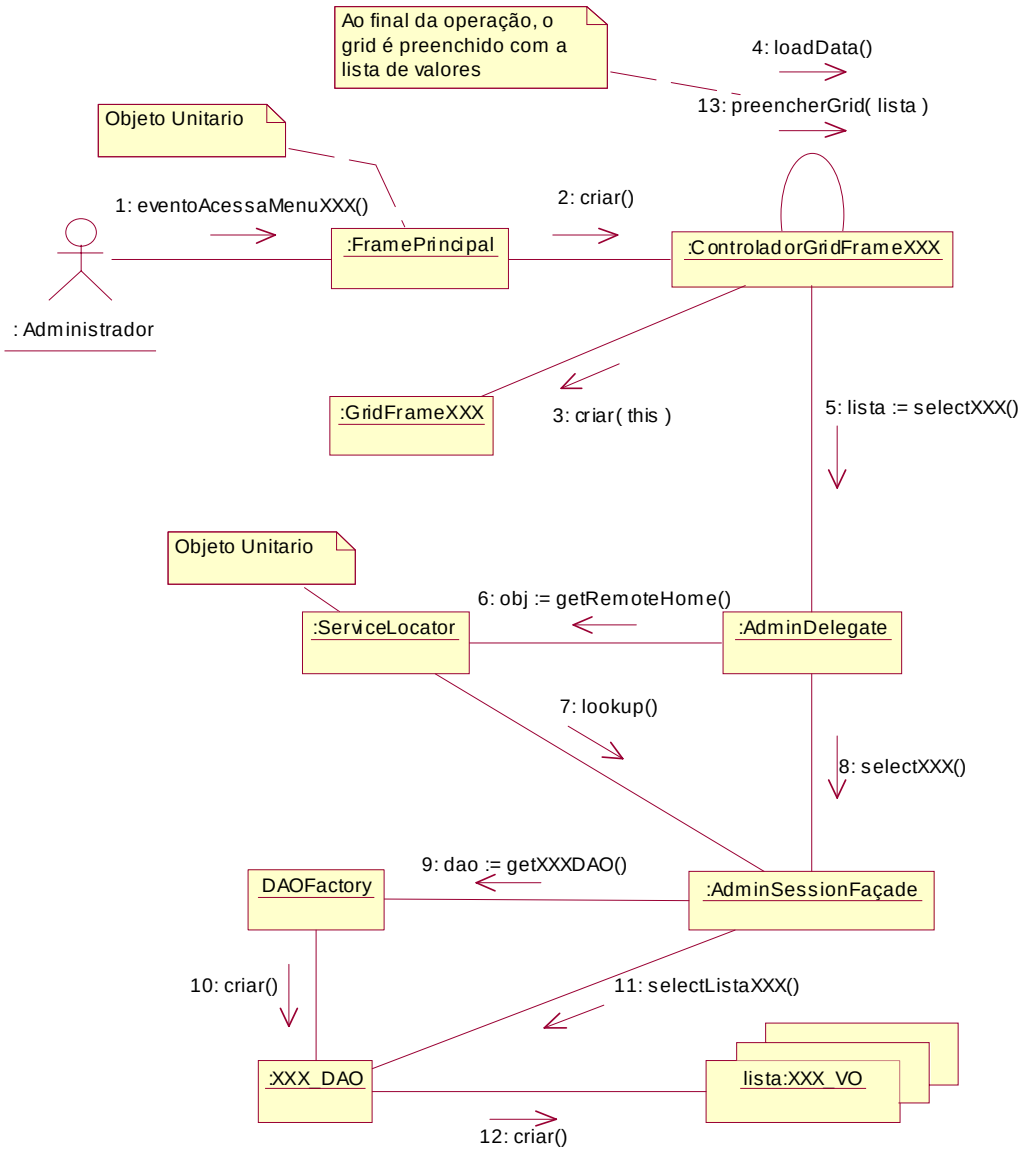

Figura 34 : Diagrama de Interação – Administração-Listar.

De acordo com a figura 34:

- 1, 2 e 3 – Após realizar o login no sistema, o administrador acessa o menu na janela principal do programa e seleciona a entidade que desejar. A classe *FramePrincipal* cria o respectivo controlador *ControladorGridFrameXXX*, que por sua vez cria a respectiva janela *GridFrameXXX*;
- 4 e 5 - O controlador invoca seu método para carregar os dados que serão exibidos pelo grid. Este método invoca a classe *AdminDelegate* para obter a lista de valores;
- 6 e 7 – *AdminDelegate* obtém uma referência remota da classe *AdminSessionFaçade* através da classe *ServiceLocator*;
- 8, 9 e 10 – *AdminDelegate* repassa a requisição para a fachada *AdminSessionFaçade*, que obtém uma instância da respectiva classe *XXX_DAO* a partir da classe *DAOFactory*;
- 11 e 12 – *AdminSessionFaçade* utiliza a classe *XXX_DAO* para obter uma lista de *Value Objects*, com todos os registros da entidade selecionada. Cada registro é armazenado em um *xxxVO*;
- 13 – Quando o controle volta à classe *ControladorGridFrameXXX*, esta preenche o grid com a lista de valores recebida.

ADMINISTRAÇÃO - ADICIONAR

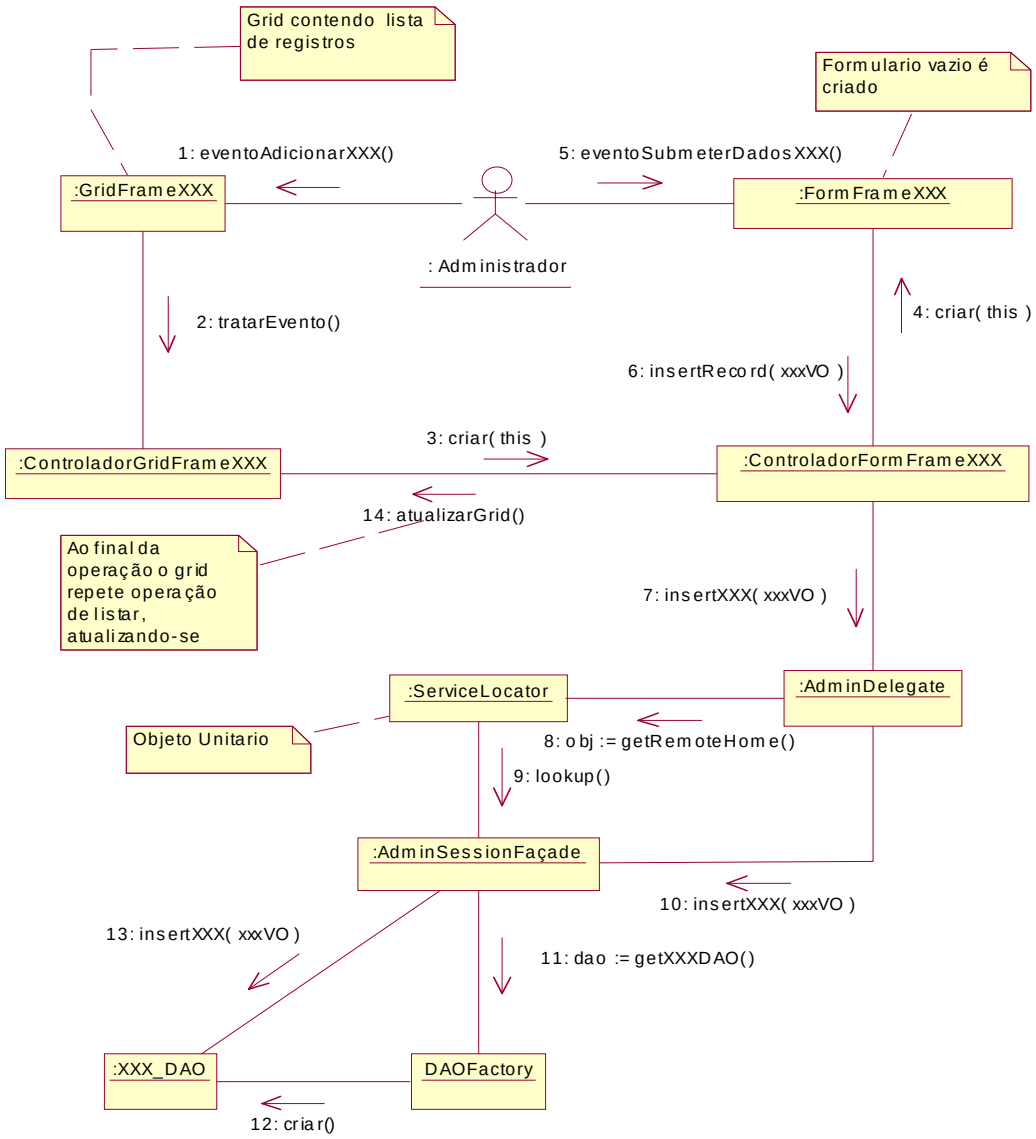

Figura 35 : Diagrama de Interação – Administração-Adicionar.

De acordo com a figura 35:

- 1, 2, 3 e 4 – Após selecionar a entidade desejada no menu da janela principal e obter a janela *GridFrameXXX* com a lista de valores, o administrador solicita adicionar uma nova entidade nesta mesma janela. O respectivo controlador *ControladorGridFrameXXX* trata este evento criando o controlador *ControladorFormFrameXXX*, que por sua vez cria a janela *FormFrameXXX* contendo um formulário em branco;
- 5 - O administrador preenche o formulário com os dados da entidade e submete. Os dados são armazenados no respectivo objeto de valor *xxxVO*;
- 6 e 7 - Este evento invoca a operação de adicionar no *ControladorFormFrameXXX*, que é repassada para a classe *AdminDelegate*;
- 8 e 9 – *AdminDelegate* obtém uma referência remota da classe *AdminSessionFaçade* através da classe *ServiceLocator*;
- 10, 11 e 12 – *AdminDelegate* repassa a requisição para a fachada *AdminSessionFaçade*, que obtém uma instância da respectiva classe *XXX_DAO* a partir da classe *DAOFactory*;
- 13 – *AdminSessionFaçade* utiliza a classe *XXX_DAO*, responsável pela lógica de inclusão no banco de dados;
- 14 – Para completar, o *ControladorFormFrameXXX* solicita ao ControladorGridFrameXXX que seu grid seja recarregado, para que o novo registro seja incluído.

ADMINISTRAÇÃO – EDITAR OU EXCLUIR

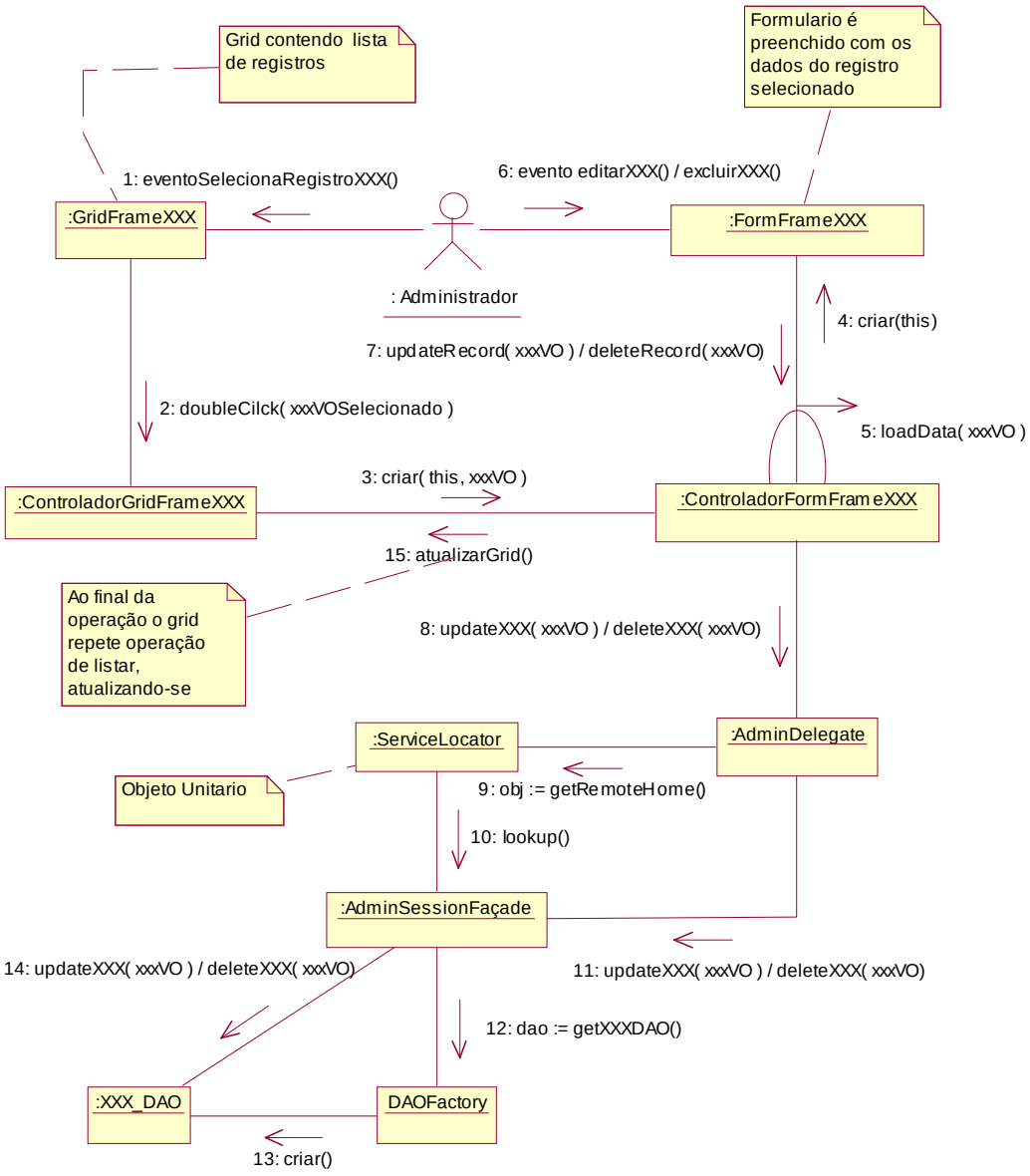

Figura 36 : Diagrama de Interação – Administração-Editar ou Excluir.

De acordo com a figura 36:

- 1, 2, 3 e 4 – Após selecionar a entidade desejada no menu da janela principal e obter a janela *GridFrameXXX* com a lista de valores, o administrador seleciona um registro no grid. O respectivo controlador *ControladorGridFrameXXX* trata este evento criando o controlador *ControladorFormFrameXXX*, que por sua vez cria a janela *FormFrameXXX* contendo um formulário;
- 5 - *ControladorFormFrameXXX* invoca método que faz o formulário ser preenchido com os dados do registro selecionado no grid. Este dados são armazenados no respectivo objeto de valor *xxxVO*;
- 6 - O administrador então edita os dados no formulário e submete ou então solicita sua exclusão;
- 7 e 8 - Estes eventos invocam as devidas operações no *ControladorFormFrameXXX*, que são repassadas para a classe *AdminDelegate*;
- 9 e 10 – *AdminDelegate* obtém uma referência remota da classe *AdminSessionFaçade* através da classe *ServiceLocator*;
- 11, 12 e 13 – *AdminDelegate* repassa a requisição para a fachada *AdminSessionFaçade*, que obtém uma instância da respectiva classe *XXX_DAO* a partir da classe *DAOFactory*;
- 14 – *AdminSessionFaçade* utiliza a classe *XXX_DAO*, responsável pela lógica de atualização ou exclusão no banco de dados. No caso de exclusão, todas as outras entidades referentes à entidade excluída também são removidas. Nesta situação, outras classes *DAO* são utilizadas, embora estejam omitidas no diagrama por simplificação;
- 15 – Para completar, o *ControladorFormFrameXXX* solicita ao *ControladorGridFrameXXX* que seu grid seja recarregado, para que as alterações sejam atualizadas.

IOPERAÇÃO (ALGORITMO)

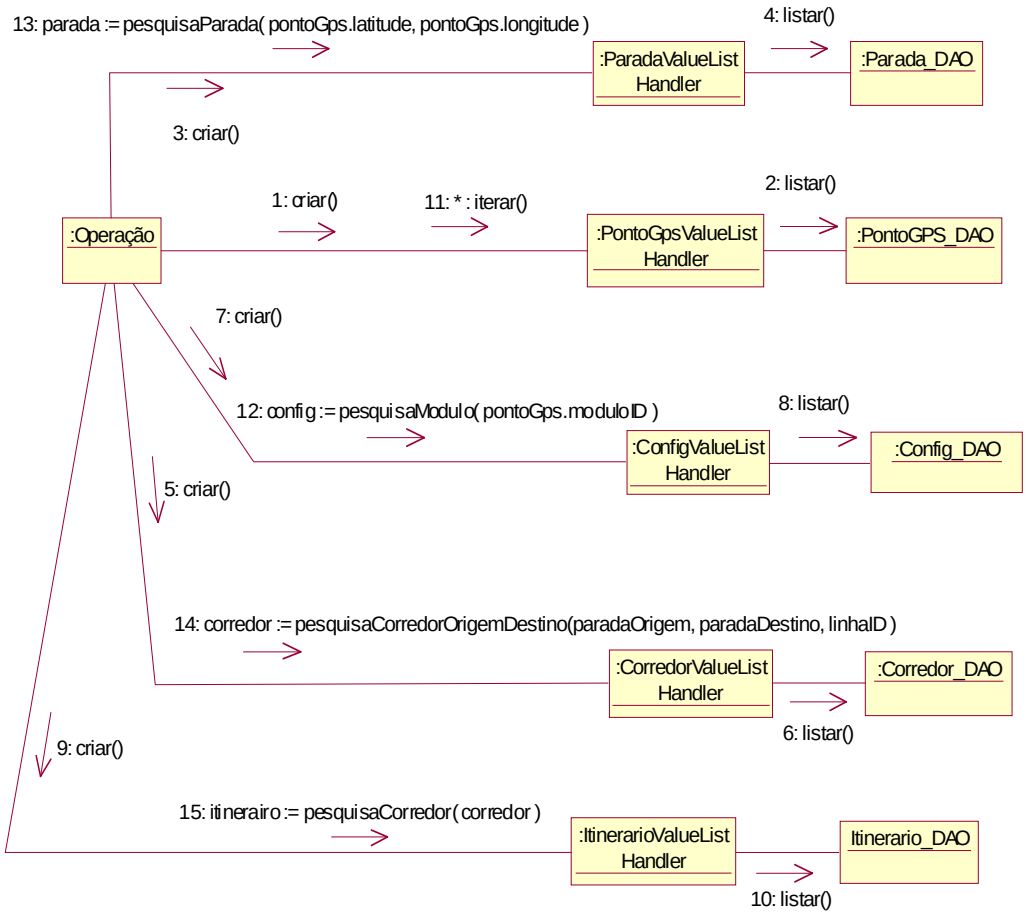

Figura 37 : Diagrama de Interação – Algoritmo da Classe Operação.

Para um melhor entendimento, apresenta-se a lógica da interface *IOperacao* em um diagrama de interação separado, na figura 37. Este diagrama não apresenta todos os passos do algoritmo, apenas aqueles considerados mais importantes na lógica.

- 1, 2, 3, 4, 5, 6, 7, 8, 9 e10 – De início, a classe obtém uma instância dos manipuladores de lista (*Value List Handler*) das diversas entidades. Eles acessam os *DAO´s* diretamente e obtém a lista de dados. A classe *DAOFactory* foi omitida no diagrama, mas é utilizada. Além destas listas, outras variáveis são instanciadas, mas também foram omitidas no diagrama;

- 11, 12, 13, 14 e 15 – A classe realiza um *looping* na lista de pontos *gps*. Para cada um deles, primeiro a lógica verifica se o modulo transmissor faz parte de um veículo que opera na linha selecionada, utilizando *ConfigValueListHandler*. Em caso positivo, a lógica então verifica se a latitude e longitude do ponto *gps* correspondem a uma parada, através de *ParadaValueListHandler*. À medida que as paradas são encontradas, a lógica busca pelos corredores percorridos, verificando *CorredorValueListHandler* e *ItinerarioValueListHandler*, e pelo corredor em que se encontra cada veículo. Ao final da operação, as listas de resultados são criadas, como apresentado nos diagramas de interação anteriores, de acordo com a pesquisa de tráfego ou análise. Muitas outras condições são verificadas ao longo do *looping*, mas foram omitidas por simplificação.

3.7. Diagrama de Componentes

Os componentes são módulos possíveis de serem implantados e substituíveis, preenchendo a arquitetura em camadas do sistema. Conforme apresentado na seção 3.4, o componente *SIMT-Server* encapsula a implementação das regras de negócio do sistema, utilizando as bibliotecas *FirebirdSql_Jdbc* e *MySql_Jdbc* para acessar os bancos de dados. Os componentes *SIMT-Web* e *SIMT-Wap* encapsulam a lógica de processamento de requisições feitas por clientes *web* e celulares. Ambos utilizam a biblioteca *Struts* para facilitar a implementação. Já o componente *SIMT-Admin* é um cliente desktop responsável por acessar os serviços de administração e análise do sistema, utilizando-se da biblioteca de aplicações gráficas *OpenSwing*.

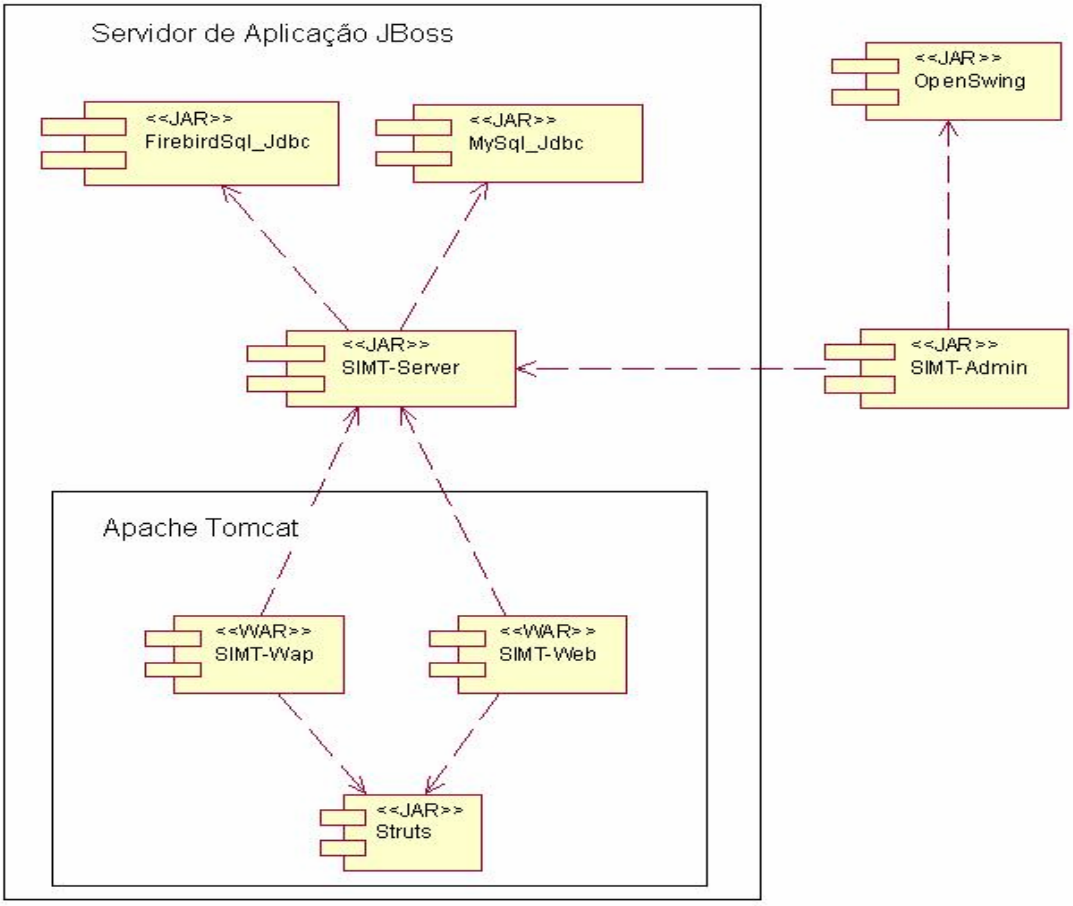

Figura 38 : Diagrama de Componentes.

Na figura abaixo temos a distribuição dos pacotes e bibliotecas pelos componentes.

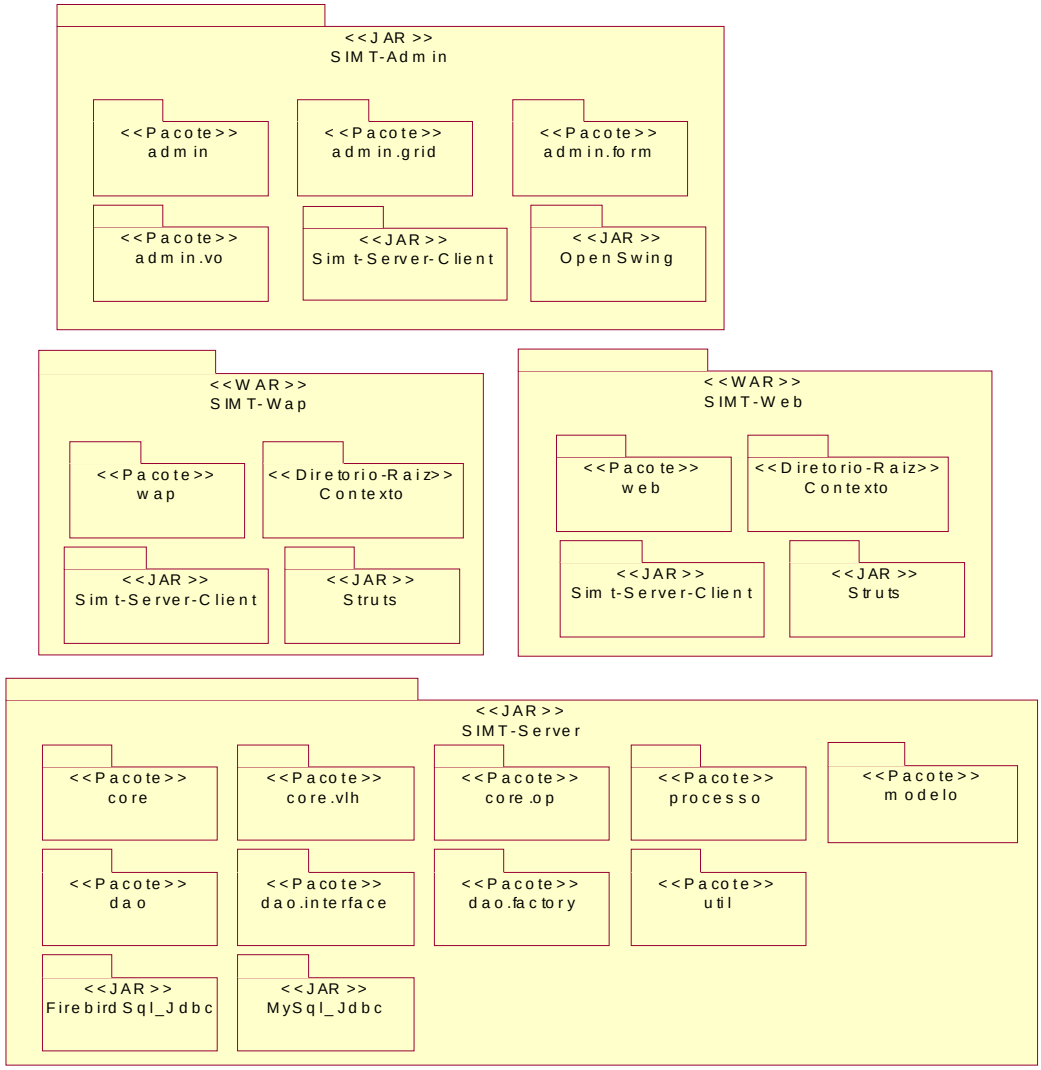

Figura 39 : Pacotes dos Componentes.

A pasta *Contexto* é o diretório onde estão os arquivos responsáveis pela apresentação das respostas geradas pelos componentes *WAR*, incluindo textos e imagens.

Importante observar que os pacotes que implementam o padrão *DAO* pertencem à camada de dados, numa visão lógica. Mas, numa visão de implementação, são incluídos no componente *SIMT-Server*, da camada de negócios. As classes desta camada podem então referenciar a interface de acesso às fontes de dados. O mesmo acontece com as classes que implementam o padrão *Business Delegate*, que pertencem à camada de negócios. Elas são incluídas no arquivo *Simt-Server-Client.jar* e este é distribuído pelos componentes das outras camadas. Assim é possível referenciar as funções do componente *SIMT-Server*.

3.8. Diagrama de Distribuição

O diagrama a seguir mostra a distribuição física da infra-estrutura de hardware sobre a qual o sistema irá rodar. As aplicações clientes, o serviço de lógica e o serviço de dados trabalham em máquinas diferentes, permitindo uma melhor organização e maior flexibilidade. Podemos observar a localização dos diversos componentes do sistema, bem como o protocolo de comunicação entre as diferentes máquinas e dispositivos.

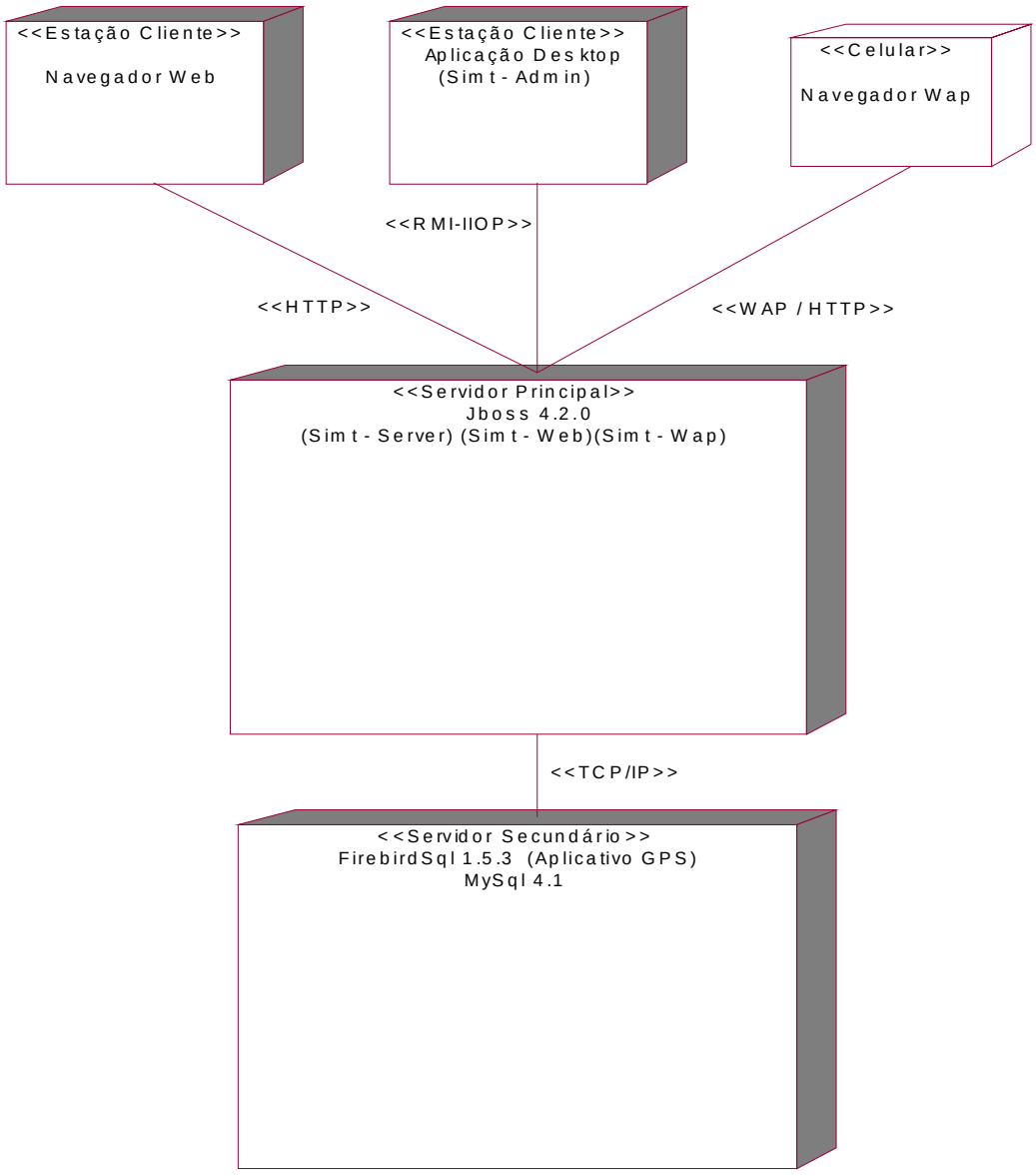

Figura 40 : Diagrama de Distribuição.

3.9. Modelo de Dados

O modelo relacional de dados a seguir é a representação dos objetos de dados persistentes do sistema. São mapeados a partir do modelo de classes para atender as funcionalidades do sistema e sua implementação corresponde ao banco de dados da aplicação.

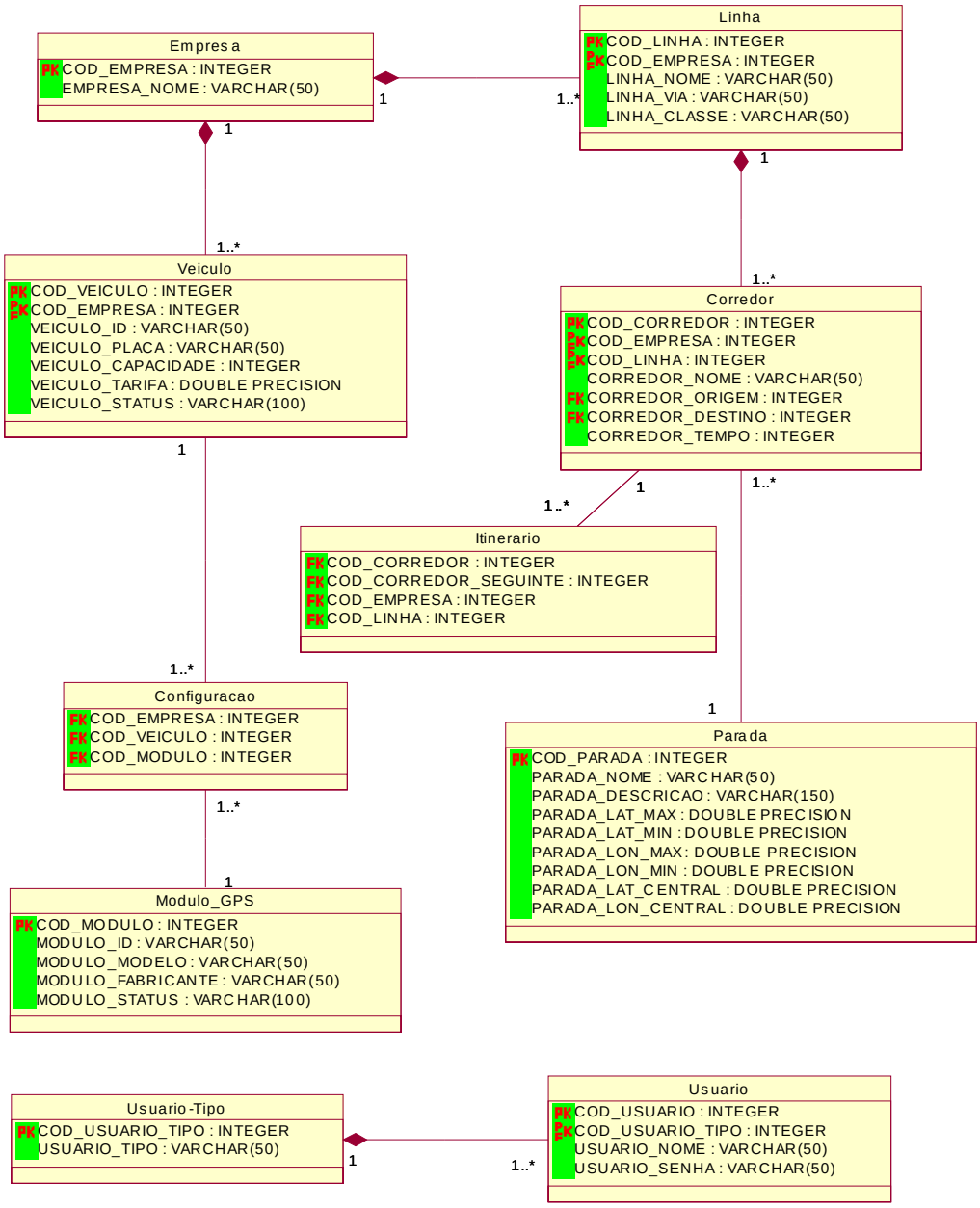

Figura 41 : Modelo de Dados.

Cada tabela mapeada no banco de dados é descrita a seguir:

Tabela 2 : Tabela Empresa

Tabela: Empresa				
Descrição: É uma organização que realiza serviço de transporte.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_EMPRESA	Sim	Não	Inteiro	Código da empresa.
EMPRESA_NOME	Não	Não	String	Nome da empresa.

Tabela 3 : Tabela Linha

Tabela: Linha				
Descrição: É um itinerário realizado por uma frota de veículos.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_LINHA	Sim	Não	Inteiro	Código da linha.
COD_EMPRESA	Sim	Sim	Inteiro	Código da empresa que realiza a linha.
LINHA_NOME	Não	Não	String	Nome da linha.
LINHA_VIA	Não	Não	String	Caminho percorrido na linha.
LINHA_CLASSE	Não	Não	String	Classe Java responsável pelo algoritmo de pesquisa.

Tabela 4 : Tabela Veículo

Tabela: Veículo				
Descrição: É o meio de transporte utilizado nas linhas.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_VEICULO	Sim	Não	Inteiro	Código do veículo.
COD_EMPRESA	Sim	Sim	Inteiro	Código da empresa proprietária do veículo.
VEICULO _ID	Não	Não	String	Identificador do veículo.
VEICULO _PLACA	Não	Não	String	Placa do veículo.
VEICULO _CAPACIDADE	Não	Não	Inteiro	Número de passageiros transportados.
VEICULO _TARIFA	Não	Não	Double	Preço cobrado para usar o veículo.
VEICULO_STATUS	Não	Não	String	Descrição do estado (parado ou operando) em que se encontra o veículo.

Tabela 5 : Tabela Modulo_GPS

Tabela: Modulo_GPS				
Descrição: É o aparelho eletrônico que faz o rastreamento dos veículos.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_MODULO	Sim	Não	Inteiro	Código do aparelho.
MODULO _ID	Não	Não	String	Identificador do aparelho.
MODULO _MODELO	Não	Não	String	Modelo do aparelho.
MODULO _FABRICANTE	Não	Não	String	Fornecedor dos aparelhos.
MODULO_STATUS	Não	Não	String	Descrição do estado (parado ou operando) em que se encontra o aparelho.

Tabela 6 : Tabela Corredor

Tabela: Corredor				
Descrição: É o percurso entre duas paradas dentro de uma linha.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_CORREDOR	Sim	Não	Inteiro	Código do corredor.
COD_LINHA	Sim	Sim	Inteiro	Código da linha a qual pertence o corredor.
COD_EMPRESA	Sim	Sim	Inteiro	Código da empresa que realiza a linha.
CORREDOR_NOME	Não	Não	String	Nome do corredor.
CORREDOR_ORIGEM	Não	Sim	Inteiro	Parada inicial do corredor.
CORREDOR_DESTINO	Não	Sim	Inteiro	Parada final do corredor.
CORREDOR_TEMPO	Não	Não	Inteiro	Tempo médio para percorrer o corredor.

Tabela 7 : Tabela Parada

Tabela: Parada				
Descrição: É o local geográfico onde ocorre embarque e desembarque de passageiros, caracterizada por uma região retangular num mapa geográfico.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_PARADA	Sim	Não	Inteiro	Código da parada.
PARADA_NOME	Não	Não	String	Nome da parada.
PARADA_DESCRICAO	Não	Não	String	Descrição sobre a localização da parada.
PARADA_LAT_MAX	Não	Não	Double	Latitude geográfica máxima.
PARADA_LAT_MIN	Não	Não	Double	Latitude geográfica mínima.
PARADA_LON_MAX	Não	Não	Double	Longitude geográfica máxima.
PARADA_LON_MIN	Não	Não	Double	Longitude geográfica mínima.
PARADA_LAT_CENTRAL	Não	Não	Double	Latitude geográfica central.
PARADA_LON_CENTRAL	Não	Não	Double	Longitude geográfica central.

Tabela 8 : Tabela Configuração

Tabela: Configuração				
Descrição: É a associação entre o veículo e o seu módulo GPS instalado.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_EMPRESA	Não	Sim	Inteiro	Código da empresa proprietária do veículo.
COD_VEICULO	Não	Sim	Inteiro	Código do veículo.
COD_MODULO	Não	Sim	Inteiro	Código do módulo GPS.

Tabela 9 : Tabela Itinerário

Tabela: Itinerário				
Descrição: É a associação entre o veículo e o seu módulo GPS instalado.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_EMPRESA	Não	Sim	Inteiro	Código da empresa proprietária do veículo.
COD_LINHA	Não	Sim	Inteiro	Código da linha.
COD_CORREDOR	Não	Sim	Inteiro	Código do corredor.
COD_CORREDOR_SEGUINTE	Não	Sim	Inteiro	Código do corredor seguinte.

Tabela 10 : Tabela Usuario

Tabela: Usuário				
Descrição: É uma pessoa que tem permissão para utilizar o sistema.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_USUARIO	Sim	Não	Inteiro	Código do usuário.
COD_USUARIO_TIPO	Sim	Sim	Inteiro	Código do tipo de usuário.
USUARIO_NOME	Não	Não	String	Nome do usuário.
USUARIO_SENHA	Não	Não	String	Senha do usuário.

Tabela 11 : Tabela Usuario_Tipo

Tabela: Usuário Tipo				
Descrição: É o tipo de usuário.				
Coluna	**Chave Primária**	**Chave Estrangeira**	**Tipo**	**Descrição**
COD_USUARIO_TIPO	Sim	Não	Inteiro	Código do tipo de usuário.
USUARIO_TIPO	Não	Não	String	Identificador do tipo.

4. DESENVOLVIMENTO

4.1. Aplicação da Metodologia

A engenharia de software orientada a objetos foi a metodologia utilizada no desenvolvimento do SIMT. Essa abordagem se faz com um processo de desenvolvimento, métodos que fornecem técnicas para construção de sistemas e ferramentas de apoio ao processo e aos métodos.

O decorrer do trabalho exigiu grande esforço a fim de adquirir o conhecimento necessário para aplicação correta da metodologia. Diversas decisões de projeto foram tomadas com base nesse aprendizado e na experiência adquirida.

4.1.1. Processo de Desenvolvimento

O processo adotado no desenvolvimento do sistema foi o processo iterativo e incremental, muito utilizado na construção de sistemas orientados a objeto. A idéia central desse processo é o desenvolvimento em iterações, cada uma gerando uma versão do sistema que será incrementada, até convergir para um sistema adequado.

Uma iteração é organizada nas fases de análise, projeto, implementação e teste. Nesse contexto, as seguintes técnicas se destacam:

- Levantamento e Definição de Casos de Uso: nessa atividade de análise, descrevem-se os processos no domínio do problema, identificando a fronteira do sistema, seus atores e os objetivos de cada um. Em seguida, definem-se os casos de uso, colocando características e funções do sistema num contexto orientado a objetivos. São levantadas também especificações suplementares, como atributos de qualidade e regras do domínio do problema;
- Definição do Modelo do Domínio: essa modelagem tenta decompor o modelo do domínio do mundo real em conceitos, atributos e relacionamentos, sem perspectiva de implementação. O artefato criado é um diagrama *UML* representando o modelo do domínio;
- Definição de Diagramas de Interação e Diagrama de Classes de Projeto: essa técnica define objetos de software e suas interações, permitindo uma visão dinâmica de objetos colaborativos. Paralelamente, descrevem-se classes de projetos com uma

perspectiva de implementação, detalhando suas responsabilidades e atributos. São extensões dos conceitos do modelo de domínio;

- Aplicação de Padrões de Projeto: os padrões são detalhados no item 4.1.3 e ajudam na definição de classes e distribuição de suas responsabilidades, tornando o sistema mais fácil de reutilizar através de uma arquitetura bem sucedida.

4.1.2. Arquitetura de Software

O padrão arquitetural adotado neste sistema foi o modelo de *camadas*. Uma aplicação típica é dividida nas camadas de integração, de negócios, de apresentação e cliente. A camada de dados encapsula a lógica relacionada ao acesso a uma fonte de dados. A camada de negócios encapsula toda a lógica da aplicação. A camada de apresentação encapsula a lógica relacionada à captura de requisições, fluxo de trabalho e processamento de respostas. A camada cliente é tipicamente uma aplicação desktop independente ou um navegador rodando applets ou páginas *html*.

O *MVC (Model-View-Controller)* foi outro padrão arquitetural utilizado no sistema. Na aplicação desktop temos a separação da interface do usuário (classes de janela) do modelo de dados (classes de objetos de valor) e do controle de fluxo da aplicação (classes de controle).

Para fazer o mapeamento desse modelo em um modelo de implementação, a plataforma escolhida foi o *Java 2 Enterprise Edition (J2EE)* [7]. Entre as características que justificaram a escolha dessa plataforma temos:

- Redução do custo e complexidade de desenvolvimento de aplicações distribuídas multicamadas feitas de componentes;
- Suporte a arquitetura de componentes EJB, serviços *web* e serviços de infra-estrutura (gerenciamento de recursos, persistência, distribuição de carga), através dos servidores de aplicação;
- Licença de uso gratuita e grande quantidade de material disponível na Internet;
- Especificação permite flexibilidade em relação ao fornecedor, através do reuso de componentes, além da portabilidade da tecnologia Java.

4.1.3. Padrões Micro-Arquiteturais (Design Patterns)

Vários padrões de projeto foram utilizados, visando as melhores práticas em aplicações que utilizam as tecnologias *JSP*, *Servlet* e *EJB*, além de evitar práticas não recomendadas e a reinvenção da roda. O ponto forte dos padrões adotados nesse projeto é que foram desenvolvidos e aplicados especificamente à plataforma J2EE. Entre eles, utilizaram-se os seguintes:

Data Access Object (DAO)

Esse padrão centraliza todo acesso aos dados em camada separada (camada de integração), oferecendo à camada de negócios uma interface comum de acesso à fonte de dados e escondendo as características de uma implementação específica, viabilizando assim a substituição de uma implementação por outra sem afetar a camada de negócios.

Session Façade

Esse padrão introduz uma fachada na camada de negócios, centralizando o controle e expondo uma interface uniforme e reduzida à camada superior. Sua implementação é feita através de *Session Beans* e deve conter o mínimo da lógica de negócio, normalmente delegada a outras classes da camada de negócios. Tipicamente o *Session Façade* precisa tratar exceções, controlar o fluxo de trabalho e gerenciar o ciclo de vida dos objetos que realizam os serviços.

A redução da interface exposta reduz o acoplamento e o número de chamadas remotas, pois estas estão concentradas na fachada do *Session Bean*, melhorando a desempenho da rede.

Business Delegate

Outra forma de diminuir o acoplamento entre as camadas é o padrão *Business Delegate*. Uma classe cliente que tenha referências para detalhes da lógica de negócios dificulta o desenvolvimento da camada de apresentação, pois esta fica vulnerável às mudanças da camada de negócios.

Business Delegate é uma classe Java comum que encapsula esses detalhes. Ela converte exceções no nível de serviço para o nível de aplicação e realiza operações corretivas

caso haja falha no serviço, sem expor o cliente. Não contém lógica de negócios, seus métodos apenas delegam responsabilidades aos *Session Façades*, tornando chamadas remotas transparentes ao cliente.

Value Object

Quando uma classe cliente precisa obter dados da camada de negócios, realiza diversas chamadas potencialmente remotas através da rede. Esse padrão é uma classe Java serializável que contém atributos suficientes para transportar todos os dados através de uma única chamada remota. O cliente pode então extrair informações usando chamadas locais, pois recebe os valores e não uma referência. Essa estratégia reduz o tráfego de rede e aumenta desempenho, pois permite a transferência de mais dados em menos chamadas.

Value List Handler

Esse padrão tem por objetivo facilitar pesquisas de busca em grandes quantidades de dados de diversas fontes. *Value List Handler* acessa um *DAO* diretamente e faz a pesquisa. Os dados são armazenados em uma coleção de *Value Objects*, permitindo que o cliente extraia informações sequencialmente através de um cursor. Esse cache de resultados no lado do servidor reduz o acesso a fonte de dados, melhorando o desempenho da rede, e mantém coesão das classes.

Service Locator

Aplicações distribuídas utilizam serviços de localização (*JNDI*) para obter referências para os recursos. A criação de objetos, se for freqüente, pode impactar na performance da aplicação, principalmente se clientes e serviços estão em camadas diferentes. Além disso, as classes clientes devem conhecer e usar a *API JNDI*, reduzindo a coesão.

O padrão *Service Locator* centraliza todo a acesso ao serviço *JNDI*, facilitando a localização de objetos *EJB* e a localização de serviços como conexões de bancos de dados. Tipicamente tem como classes clientes um *Business Delegate* ou um *DAO*. Pode ainda guardar as referências previamente localizadas, melhorando a desempenho da localização.

4.2. Escolhas de Projeto

Atualmente existem diversas ferramentas, aplicativos e componentes que auxiliam o processo de desenvolvimento de um software. Uma seleção correta desses artefatos, adequados às necessidades do projeto, pode reduzir consideravelmente o esforço de desenvolvimento, facilitando a organização e documentação.

4.2.1. Ferramentas

A ferramenta utilizada para codificação do sistema foi o *Eclipse* 3.2 [16], de licença gratuita e bastante utilizada por desenvolvedores. O *Eclipse* possui uma interface fácil de usar e recursos que permitem identificar erros no código no momento da edição, o que economiza tempo de depuração. Integrada ao *Eclipse*, outra ferramenta bastante utilizada foi o *Ant*, que auxilia na compilação e empacotamento das classes, através de um arquivo de regras *XML*.

Para trabalhar com o banco de dados foi utilizado o *DBManager* 3.1 [17], outra ferramenta de licença livre para gerenciamento dos dados.

O programa *Openwave SDK 6.2* [18], de licença livre, permitiu a simulação da interface *wap* para celular.

4.2.2. Aplicativos e Componentes

O servidor de aplicação *J2EE* adotado foi o *JBoss* 4.2.0 [8], uma das implementações mais utilizadas, com ampla documentação disponível na Internet. O *JBoss* contém o container *EJB*, que executa os componentes *EJB* da camada de negócios do sistema, controlando seus ciclos de vida e realizando outros serviços de infra-estrutura, e o container *Tomcat*, responsável por executar os códigos *JSP* e *Servlet* presentes na camada de apresentação do sistema.

O servidor de banco de dados utilizado foi o *MySQL* 4.1.2 [9], de licença gratuita e um dos mais utilizados existentes no mercado. Este aplicativo conta com ampla documentação e ferramentas de apoio disponíveis na Internet.

O *framework Struts* 1.1 [11] foi outro componente utilizado. O *Struts* tem o objetivo de facilitar a implementação do padrão arquitetural *MVC* / Camadas em aplicações *web*. Ele implementa um controlador que centraliza o processamento das requisições recebidas pela

camada de apresentação e delega os serviços à camada de negócios. Conta com uma coleção de classes utilitárias que encapsulam dados de formulários, erros, lógica para apresentação e redirecionamento de respostas, entre outras. Um ponto forte deste *framework* é o uso de arquivos de configuração para realizar alterações na aplicação. Sem a necessidade de editar e recompilar classes, a aplicação pode reagir rapidamente às alterações com um mínimo de esforço.

O *framework OpenSwing* [12] foi a solução utilizada para implementar o cliente desktop do sistema. Assim como o *Struts*, ele se baseia no padrão *MVC*. Utiliza a biblioteca Java *Swing* e implementa classes que encapsulam formulários, modelo de dados e lógica de controle de janelas. Possui uma coleção de componentes gráficos utilizados em formulários, como botões, menus, grids e tabelas. Este *framework*, de licença gratuita, possui uma documentação bem simples, presente em seu site, mas que se mostrou muito útil, possibilitando todo o aprendizado necessário a sua utilização.

A *API Google Maps* [13], desenvolvida na linguagem *JavaScript*, foi utilizada para a adição de mapas geográficos em páginas *web*. Possui ampla variedade de funções para manipulação de mapas, de fácil aprendizado. Sua utilização depende apenas de um registro, indicando em qual *URL* será publicada a página *web* contendo o mapa.

Outro componente importante no funcionamento do sistema foi o *Java Service Wrapper* [14]. Este componente permitiu instalar o *JBoss* como um "serviço" nos sistemas operacionais *Windows*. Além de eficaz, sua instalação foi bem simples.

4.3. Iterações de Projeto

De acordo com a metodologia do processo iterativo, desenvolveu-se a seguinte seqüência de interações:

1ª. Iteração

O trabalho teve início com a concepção do sistema pelos interessados. Procurou-se definir o problema a ser resolvido, escopo do produto e sua viabilidade. O levantamento dos requisitos e regras do domínio iniciou o processo de análise do sistema.

Continuando, fez-se o levantamento de casos de uso, identificando a fronteira do sistema, seus atores e objetivos. O problema chave nesta prática foi definir o nível e escopo

dos casos de uso, de acordo com os objetivos a serem atendidos pelo sistema. Apenas os casos de uso principais foram detalhados.

A partir dos casos de uso levantados, criou-se um modelo de classes do domínio, onde os principais conceitos, relacionamentos e atributos foram identificados. Nesta prática, a dificuldade esteve em distinguir conceitos, comportamentos e atributos.

A tarefa seguinte foi pesquisar e definir soluções para o estabelecimento do ambiente de desenvolvimento, como aplicativos e ferramentas de suporte, aparelhos de rastreamento e computadores para desenvolvimento. O último passo foi a instalação e teste das ferramentas e aplicativos, destacando-se o servidor *JBoss*, o banco de dados *MySQL* e o aplicativo de rastreamento.

2ª. Iteração

Nesta iteração, o processo de análise continuou sendo prioridade, com a adição de novos requisitos funcionais e detalhamento de mais casos de uso e do modelo de domínio.

Investigando um cenário simples de sucesso de um dos casos de uso principais, descrevemos como os atores interagem com o sistema através de eventos, que são tratados por operações de sistema. Estas operações foram detalhadas através de diagramas de interação e serviram de base para o projeto de classes de software. Algumas dessas classes representaram os conceitos do modelo de domínio.

O modelo de domínio permitiu identificar as entidades cujos dados teriam que ser salvos no banco de dados. O modelo de dados foi então implementado no banco *MySQL*.

A fase final foi a implementação e testes de algumas funcionalidades do componente *SIMT-Server*, permitindo a consulta das condições do tráfego nos corredores e o posicionamento dos veículos. Nesta primeira versão, entre as dificuldades encontradas podemos citar o uso das conexões com banco de dados gerenciadas pelo *JBoss* e a implementação de listas de objetos para pesquisas de informações.

3ª. Iteração

O foco passou a ser o projeto das classes de software, embora tenha continuado com o detalhamento dos casos de uso e atualização dos requisitos.

Nesta etapa procuramos utilizar os padrões de projeto na obtenção das classes. Atenção especial foi dada na definição das operações que seriam expostas pelo sistema em

classes de fachada, no componente *SIMT-Server*, permitindo que o sistema fosse separado em camadas sem acoplamento.

A tarefa mais importante foi a implementação do componente *SIMT-Web* e sua integração como componente *SIMT-Server*. Nesta primeira versão, o cliente *web* não contava com os mapas geográficos, apenas fornecia informações sobre o tráfego nos corredores. O esforço maior se deu no aprendizado do *framework Struts*.

Devido à semelhança na implementação, o componente *SIMT-Wap* também teve a sua primeira versão. Foi utilizada a linguagem *WML*, específica para navegadores de aparelhos celulares.

4ª. Iteração

Nesta iteração, operações para administração do banco de dados foram adicionadas à fachada do componente *SIMT-Server*, permitindo assim interagir com o componente *SIMT-Admin*. Este começou a ser implementado com o aprendizado do *framework OpenSwing*.

O componente *SIMT-Web* teve sua segunda versão com a inclusão de um mapa geográfico na página *HTML*, através da *API Google Maps*, além de ajustes na sua visualização. O componente *SIMT-Server* ainda não disponibilizava informações sobre o tempo de chegada nas paradas.

Todos os casos de usos foram detalhados, levando-se em conta todas as alterações realizadas nos modelos nas interações anteriores. O mesmo aconteceu para os componentes implementados, de modo a refletir as atualizações.

5ª. Iteração

A meta nessa iteração foi a implementação e testes das operações de análise no componente *SIMT-Server*. Outra tarefa importante foi o término da primeira versão do componente *SIMT-Admin*. Nessa versão, o componente não contava com as funcionalidades de análise do tráfego e geração de relatórios, apenas realizava o cadastro e edição das entidades no banco de dados do sistema.

Essa interação terminou com uma revisão completa dos modelos de análise e projeto, verificando se os componentes implementados até o momento atendiam os requisitos e objetivos dos usuários. Diversos acertos e melhorias foram estipulados, seguindo as necessidades do projeto.

6ª. Iteração

Ao final desta iteração obteve-se a primeira versão completa do sistema. Os modelos teóricos foram finalizados. O componente *SIMT-Server* teve sua funcionalidade completada, com a consulta do tempo de chegada nas paradas, permitindo que os componentes *SIMT-Web* e *SIMT-Wap* a utilizassem. O componente *SIMT-Admin* também foi finalizado com a geração de relatórios de análise.

4.4. Dificuldades e Contratempos

Ao longo da realização do projeto, várias dificuldades foram encontradas. Algumas delas necessitaram uma atenção maior em busca de soluções e são descritas a seguir:

Funcionamento das linhas de transporte

Quando analisamos o funcionamento das linhas de transporte, verifica-se uma diversidade de modos de operação, o que dificulta a criação de um modelo que atenda a todas as linhas. A idéia inicial seria a criação de um algoritmo de pesquisa genérico, capaz de processar os dados para qualquer linha. Mas ao realizar os testes na Ilha do Fundão já foi possível identificar as dificuldades em se criar tal algoritmo.

A situação ideal é aquela em que o sistema possa identificar a linha realizada por cada veículo mesmo que estes não tenham uma linha fixa para operar. A lógica do sistema se baseia na busca pelos veículos que se encontram nos corredores pertencentes à linha selecionada pelo usuário. O problema surge quando duas linhas possuem corredores em comum, o que dificulta saber qual linha o veículo está operando e assim qual o próximo corredor que ele irá seguir.

No caso da Ilha do Fundão, existem duas linhas que foram monitoradas. Uma delas é a linha Alojamento-Vila e a outra é a linha Especial. Ambas as linhas possuem um trecho em comum, o corredor Hospital-CCMN. Caso um ônibus tenha saído do hospital em direção ao CCMN, ou saído do CCMN em direção ao Hospital, a princípio não há com saber em qual linha ele está operando.

Uma primeira tentativa de solução é fixar os veículos nas linhas. Isto acaba com a dúvida sobre qual linha o veículo estaria fazendo, tornando o algoritmo de pesquisa mais

simples. De acordo com o responsável pela operação dos veículos na Ilha do Fundão, os veículos são alocados em linhas pré-definidas, com exceção do caso em que um deles tem problemas mecânicos e precisa ser substituído. Mas na prática não é isto que acontece, havendo algumas mudanças não programadas entre as linhas. Além disso, um veículo pode sair da operação, indo para a garagem ou então seguir para o campus da Praia Vermelha ou para Bonsucesso.

Além dos problemas destacados acima, outra situação deve ser considerada. É aquela em que, numa mesma linha, existem veículos com muitas paradas (“parador”) e outros com poucas paradas (“rápido”). Como o intervalo de deslocamento nos corredores será diferente, a estimativa de chegada nas paradas e o índice de congestionamento não estarão corretos. Logo terão que ser consideradas linhas diferentes

Infelizmente a Ilha do Fundão não se mostrou uma rede de transporte ideal para realizar os testes nesta versão do sistema. Além do problema da troca dos veículos, seus corredores têm um intervalo de deslocamento pequeno, o que aumenta o percentual de erro nas medidas de tempo.

Lógica de pesquisa na operação

Diante das dificuldades listadas anteriormente e da previsão de que outras linhas apresentariam os mesmo problemas, foi decidido criar um algoritmo de pesquisa para cada linha em sua própria classe Java. Tal classe identifica exatamente qual banco de dados do aplicativo *GPS* pesquisar, quais paradas procurar e, principalmente, a lógica correta para não confundir sua linha com outras.

Uma classe cliente usa a interface Java *IOperaçao,* com a definição dos métodos a serem utilizados, sem se preocupar em saber qual implementação será utilizada. Este desacoplamento permite que os algoritmos mudem independentemente. A única restrição é o cliente saber o nome da classe Java que implementa o algoritmo. Dessa forma existe a necessidade de se cadastrar no banco de dados do sistema, na tabela de linhas, o nome da classe.

Embora o algoritmo de pesquisa seja parecido, cada classe de operação de uma linha possui um método para análise e outro para tráfego. Inicialmente ambas as funções eram executadas num único método, mas isto criava um acoplamento indesejado, dificultando possíveis alterações nas lógicas de cada uma.

Como já foi visto na seção 3.6, o algoritmo de pesquisa inicia com a seleção, no banco de dados do aplicativo *GPS*, de todos os pontos geográficos enviados até o momento. Esta lista de pontos está agrupada por módulos e ordenada pela hora de registro. Para cada ponto registrado, o algoritmo verifica se é uma parada de algum corredor da linha selecionada. Esse processo continua até que todos os corredores sejam identificados e repete-se para todos os módulos. Definidos os corredores onde se encontram cada veículo, o algoritmo faz a estimativa do tempo de chegada nas paradas. Este cálculo foi motivo de dúvida sobre como seria feito, dispondo de duas opções.

Na primeira, usou-se o índice de congestionamento do corredor anterior, completado pelo veículo, para estimar o tempo de viagem até a parada final do corredor. O problema desta opção está no fato do índice de congestionamento do corredor anterior não refletir as condições reais do corredor atual do veículo. Cada corredor apresenta seu próprio índice de congestionamento. Assim o tempo estimado não seria uma medida confiável.

Na segunda opção, escolhida como solução, usou-se o índice de congestionamento do corredor atual do veículo, calculado a partir do último veículo que o completou. Embora esta opção seja mais confiável que a anterior, o tempo estimado pode não refletir as condições reais do tráfego no instante da consulta, pois se baseia na última vez que o corredor atual foi completado por outro veículo. Se houver uma variação no fluxo de veículos de um corredor, ela só será “notada” pelo sistema quando algum veículo terminar de percorrê-lo. O ideal é que tenhamos vários veículos igualmente espaçados no itinerário da linha, de modo que os corredores sejam completados com maior freqüência, aumentando o grau de confiabilidade das medidas de tempo.

Alguns detalhes são muito importantes na lógica de pesquisa. Para calcular o tempo previsto para um veículo chegar a uma parada é preciso saber quanto tempo ele já percorreu no corredor em que se encontra. Esse tempo é calculado pela diferença entre a hora da consulta (hora presente) e a hora que ele saiu da parada de origem do corredor (hora de partida). Pode acontecer casos em que um veículo está ligado, esperando por passageiros numa parada. O tempo de espera na parada deve ser descartado, para que o tempo percorrido pelo veículo não fique incorreto. Outro fator importante é o relógio da máquina onde se encontra instalado o sistema. Se este relógio estiver incorreto, o tempo previsto de chegada nas paradas também será incorreto.

Alguns veículos podem sair do itinerário da linha ou alguma falha nos aparelhos de rastreamento impede que seja feito o registro em alguma parada. Estas situações podem gerar

um tempo percorrido pelo veículo fora do normal. O algoritmo deve identificar estas situações e descartar o veículo da lista de previsões de chegada nas paradas.

Outro detalhe que requer atenção é o caso em que numa mesma parada exista a espera por veículos em dois sentidos. No caso da Ilha do Fundão, na parada CCMN ocorre espera por veículos vindos da Vila e do Alojamento. É preciso identificar qual a origem do veículo para que se possa saber o sentido do deslocamento, sem erro na identificação das linhas.

Uma linha de transporte pode ser circular ou não. A lógica de pesquisa se baseia no corredor seguinte ao último corredor percorrido para identificar em qual corredor o veículo se encontra. Se uma linha não for circular, seu primeiro corredor não é corredor seguinte de nenhum outro. Esta situação exige que o algoritmo saiba identificar o veículo no primeiro corredor usando outros critérios. Neste caso pode-se usar a parada de origem da linha e o tempo percorrido como parâmetros.

Todas as paradas de uma linha podem ser cadastradas, mas a estimativa do tempo de chegada só é realizada para as paradas de origem e de destino dos corredores da linha. A lógica do sistema se baseia no índice de congestionamento de um corredor. Poderíamos supor que cada par de paradas consecutivas na linha fosse considerado um corredor, mas não teríamos estimativas satisfatórias, pois o pequeno intervalo de tempo aumentaria o percentual de erro.

Foram escolhidas as paradas de maior movimento de passageiros na linha, sem deixar que os corredores fossem muito extensos, pois muitas paradas seriam excluídas. Além disso, os corredores são consecutivos. Se existirem trechos “vazios” não será possível estimar o tempo de chegada, pois não sabemos o tempo gasto neste trecho “vazio”.

Funcionamento dos aparelhos de rastreamento

Entre os pontos de dificuldade, um dos maiores problemas é o funcionamento dos aparelhos de rastreamento. Inicialmente, a maioria dos aparelhos estava com um intervalo de registro de 3 minutos, o que é bem superior ao tempo que um ônibus gasta quando está numa parada. Como não há o registro do momento em que o veículo está na parada, o algoritmo de pesquisa não consegue completar sua lógica, produzindo erros nos resultados. Após entrar em contato com a empresa Geocontrol, este intervalo passou para menos de 1 minuto, o que já diminuiu bastante os erros.

Além disso, eventualmente algum aparelho para de transmitir, necessitando uma visita técnica para concertá-lo. Esta falha causa um erro na análise de *headway* da linha, pois necessita que todos os veículos estejam operando.

Aplicação da tecnologia Java

O desenvolvimento desse sistema foi uma grande oportunidade para o aprendizado das tecnologias que compõem a plataforma *J2EE*. A inexperiência sobre o assunto causou algumas dificuldades, principalmente no início da implementação do projeto, quando se gastou mais tempo com o aprendizado da tecnologia do que com a codificação.

A instalação e execução do servidor de aplicação *JBoss* não apresentou maiores problemas. Seu processo de implantação de uma aplicação requer a edição de arquivos de configuração para os componentes e outros recursos do servidor. Entre essas configurações, existe o conceito de conexão gerenciada pelo servidor e conexão não gerenciada. Inicialmente, sem considerar este detalhe, o sistema utilizava conexões gerenciadas, mas isto estava causando erro. Nesta configuração não é permitido abrir conexões com mais de um banco de dados simultaneamente. Mas isto é necessário, pois existem dois bancos sendo utilizados, o *MySQL* e o *FirebirdSql*. Após configurar as conexões para não serem gerenciadas pelo servidor, o funcionamento passou a ser normal.

Durante a codificação, os maiores problemas aconteceram com a manipulação de listas de dados, usadas pelo padrão *Value List Handler*. A interação sobre estas listas gerava erros em tempo de execução difíceis de encontrar, exigindo um esforço maior de depuração. Tais erros eram decorrentes da não inicialização de variáveis nas listas.

A integração do componente *SIMT-Admin* com o *SIMT-Server* não foi simples. O *framework OpenSwing* possui a interface *ValueObject*, que deve ser implementada por todas as classes de objeto de valor, pois ela é referenciada por todos os componentes gráficos do *framework*. A princípio, para criar as classes de objeto de valor no *SIMT-Admin*, bastaria implementar esta interface e herdar as respectivas classes do *SIMT-Server*. Mas isto não foi possível. A biblioteca do *framework* utiliza classes de *wrapping* (embrulho) para tratar os tipos primitivos como objetos. Como as classes de objeto de valor no *SIMT-Server* foram implementadas utilizando os tipos primitivos, foi necessário reescrever todas estas classes para adequá-las ao *OpenSwing*.

A ferramenta *Eclipse* apresenta um erro que, no início da codificação, provocava uma perda de tempo considerável. Às vezes, quando se está editando o código fonte do sistema, o

Eclipse não consegue localizar a referência que existe entre classes de diferentes pacotes, acusando erro no código, embora esteja tudo correto. Cada vez que isso acontecia era necessário recriar todo o projeto no *Eclipse*. Como solução, basta remover a biblioteca nativa do Java da lista de referências e depois adicioná-la novamente. Esta ação provoca uma atualização no Eclipse que elimina o erro.

Uma das grandes vantagens do *JBoss* é o controle do ciclo de vida dos objetos instanciados na memória. Este controle permite que um *pool* de instâncias seja reutilizado em cada chamada, tornando o processamento mais rápido. Mas um problema ocorre quando há uma alteração no banco de dados do sistema, deixando algumas instâncias com seus dados desatualizados. Infelizmente não se encontrou outra solução senão deixar o próprio *JBoss* decidir remover todas as instâncias da memória ou então reiniciá-lo.

Depois de feitos os testes no sistema, era necessário colocá-lo na máquina servidora para que fosse possível acessá-los externamente à rede do laboratório *Planet*. Mas esta máquina já possuía um site rodando no servidor *web Apache*, o que criava um problema. Por padrão, toda requisição *HTTP* que chega ao servidor é direcionada a porta 80, utilizada pelo *Apache*. Mas o servidor *web Tomcat*, contido no *JBoss*, utiliza a porta 8080. Não era possível redirecionar as requisições para a porta 8080, pois o site contido no *Apache* sairia do ar. A solução estava no próprio *Apache* [15]. Este possui arquivos de configuração que permitem ao programa gerenciar múltiplos sites rodando numa mesma máquina. Primeiro criou-se um novo endereço de IP na máquina (*Virtual Host*). Depois se configurou o *Apache* para redirecionar as requisições *HTTP* que chegam neste novo IP para o *Tomcat*.

Uso de celulares para acessar o sistema

Atualmente existe uma ampla variedade de modelos de celulares com acesso a Internet. Mas não existe uma padronização que permita a todos os modelos visualizarem uma página *wap* da mesma maneira.

A pequena capacidade de memória dos navegadores de celular impõe limites ao conteúdo das páginas *wap*. Outro são os recursos gráficos. Os quadros suportam imagens, mas de um tipo especial, de tamanho bem menor. Alguns celulares nem são capazes de exibir imagens. Devido a isso, a maioria dos sites *wap* atuais trabalha apenas com texto puro. Além disso, os controles *WML* podem ser interpretados de maneira diferente entre os modelos, causando erros na interface com o usuário.

4.5. Resultados

Apesar das dificuldades encontradas, o resultado foi bastante satisfatório. Os requisitos propostos foram atendidos e a metodologia aplicada permitiu ao projeto atingir um ótimo nível de organização.

A cada nova iteração do projeto foi possível perceber os benefícios da modelagem orientada a objeto. Os diagramas e modelos serviram como guia para a adição de novas funcionalidades e alterações. Os padrões de projeto agilizaram a implementação, ao evitar que erros comuns relacionados à utilização da tecnologia fossem cometidos. Entre os padrões, vale destacar o *Value List Handler*, muito utilizado devido ao grande número de listas de dados existentes no sistema, permitindo operações mais rápidas. O padrão *Session Façade* facilitou o mapeamento dos casos de uso nas operações do sistema, melhorando o reaproveitamento e a organização do código fonte.

Nos testes realizados, as previsões e medidas de tempo foram bem satisfatórias. A margem de erro mínima equivale ao intervalo de transmissão dos aparelhos de rastreamento e esteve dentro do esperado, embora algumas situações anormais ainda não puderam ser solucionadas. Quando veículos desviam de seu itinerário normal ou ocorrem falhas no registro da passagem de um veículo por uma parada, como foi mencionado na seção 4.4, ambos os casos geram resultados incorretos. O sistema consegue detectar as condições anormais, mas isto não acontece de imediato, gerando informações incorretas. Partindo do princípio de que estas situações não ocorrem com freqüência, pode-se dizer que o sistema oferece um bom grau de confiabilidade.

Uma outra situação que pode gerar erro é aquela em que o tempo percorrido pelo veículo no corredor ultrapassa o tempo esperado para percorrê-lo. Nesta situação, o tempo previsto de chegada será negativo, fazendo o algoritmo supor que o veículo já saiu do corredor em que se encontra. Se uma consulta é realizada, o mapa localiza o veículo no corredor certo, mas a previsão de chegada na parada indica um outro veículo, seguinte ao atual. Esta situação tende a ocorrer quando os veículos estão bem próximos do final do corredor. Uma possível solução, ainda não implementada, é informar também a hora de partida da parada do último veículo, para que o usuário tenha, com base na margem de erro das medidas, uma informação mais completa.

Em algumas consultas realizadas, a previsão do tempo de chegada nas paradas mudou de forma inesperada. Estas previsões se baseiam nos veículos que estão operando na linha no

momento da consulta, ignorando aqueles que estão desligados. Caso um veículo seja ligado e entre em operação, as previsões de tempo de algumas paradas poderão mudar. Se fosse possível saber a hora exata de partida dos veículos desligados isto não aconteceria, mas esta informação é muito imprecisa para ser utilizada.

Os testes de análise da operação também foram satisfatórios, embora as linhas do campus da Ilha do Fundão não sejam apropriadas para um estudo mais aprofundado. Foi possível levantar todo trajeto realizado por um veículo ao longo do dia, além das medidas de *headway*. Em relação ao *headway*, apenas as paradas que são destino em um corredor possuem esta opção de análise.

Uma funcionalidade a ser destacada é a adição de uma nova linha monitorada pelo sistema. Quando isto ocorre, a primeira tarefa é criar uma nova classe de acesso ao banco do aplicativo *GPS*, caso os rastreadores utilizem um novo banco de dados. A segunda tarefa é criar a classe de operação da nova linha. Por último devemos realizar o cadastro da linha, empresa e veículos, além das paradas, corredores e itinerário, no banco de dados do sistema. Se por um lado, a adição de novas linhas ao sistema exige novas implementações, por outro, não há impacto sobre o que já foi desenvolvido, pois as linhas funcionam de maneira independente.

O desempenho do sistema foi considerado bom, exibindo um tempo de resposta aceitável e dentro do limite proposto, principalmente para as consultas via *web* e *wap*. Não foi possível avaliar o desempenho do sistema em atender muitos acessos simultâneos. O mesmo aconteceu em relação à situação em que existe uma grande quantidade de dados a serem processados no banco, pois existem apenas duas linhas sendo monitoradas pelo sistema. As consultas *wap* foram realizadas em poucos modelos de aparelho celular, o que impediu de verificar a compatibilidade dos diversos modelos com as páginas *wml* apresentadas pelo sistema.

Outro ponto de destaque foi a portabilidade do sistema. A instalação do sistema foi realizada em máquinas diferentes de maneira fácil e rápida. Basta instalar o servidor *JBoss* e implantar os componentes *SIMT-Server*, *SIMT-Web* e *SIMT-Wap*. O componente *SIMT-Admin* e o *JBoss* podem rodar em qualquer máquina onde esteja instalada a *Java Virtual Machine (JVM)*. Arquivos de configuração nos componentes permitem a definição de novos endereços de rede sem a necessidade de recompilação dos mesmos.

As interfaces gráficas apresentam algumas limitações visuais, sem uma estética desejável. Porém as consultas possuem um acesso prático e objetivo, tornando a ferramenta fácil de usar. Todas as funcionalidades propostas são realizadas pelo sistema, mas existe a

necessidade de se aprimorar as suas interfaces. Como podemos ver na figura 42, a interface do cliente desktop permite ao usuário operar sobre diversas janelas ao mesmo tempo, com um desempenho satisfatório.

O cliente *web* apresenta uma interface bem simples, exibindo um formulário para seleção das linhas de transporte e opções de consulta, além de um mapa geográfico com a localização dos veículos, conforme podemos ver nas figuras 44 e 45. A figura 43 apresenta um simulador da interface do cliente *wap*, ainda mais simples e sem a opção de localizar veículos, devido às limitações dos navegadores *wap*. O endereço eletrônico http://rastreamento.pet.coppe.ufrj.br permite acesso a interface *web*. Já o endereço http://146.164.61.172, quando acessado pelo celular, apresenta a interface *wap*.

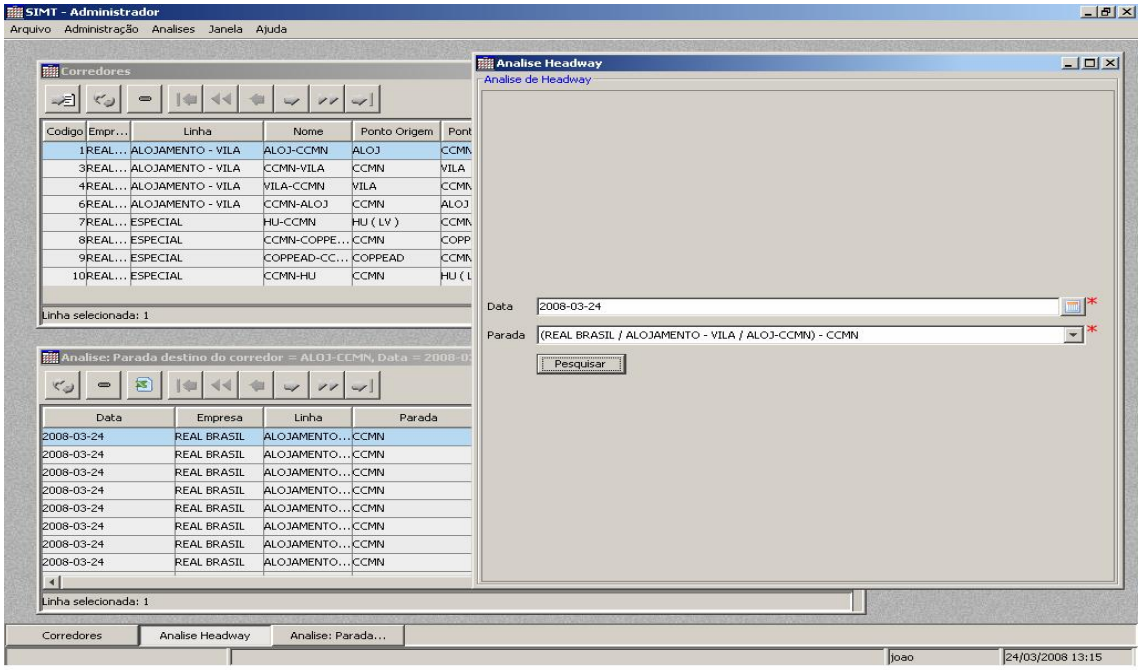

Figura 42 : Interface Desktop.

Figura 43 : Interface *Wap*.

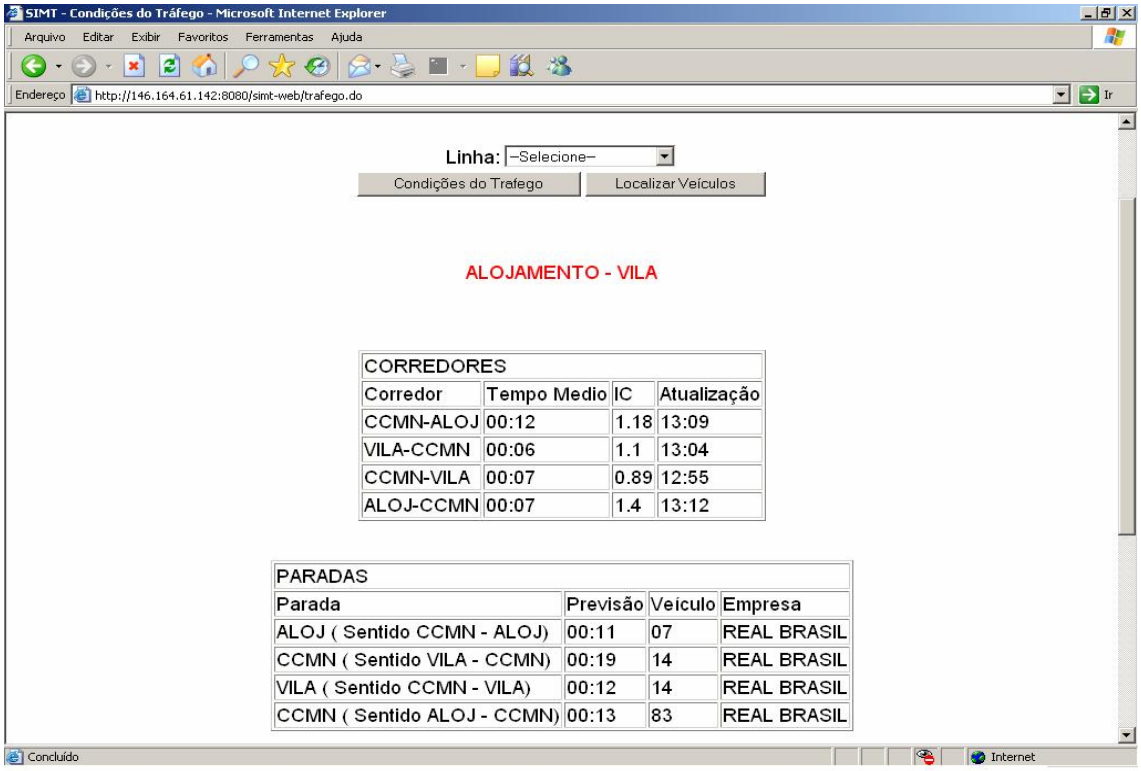

Figura 44 : Interface *Web* – Condições do Tráfego.

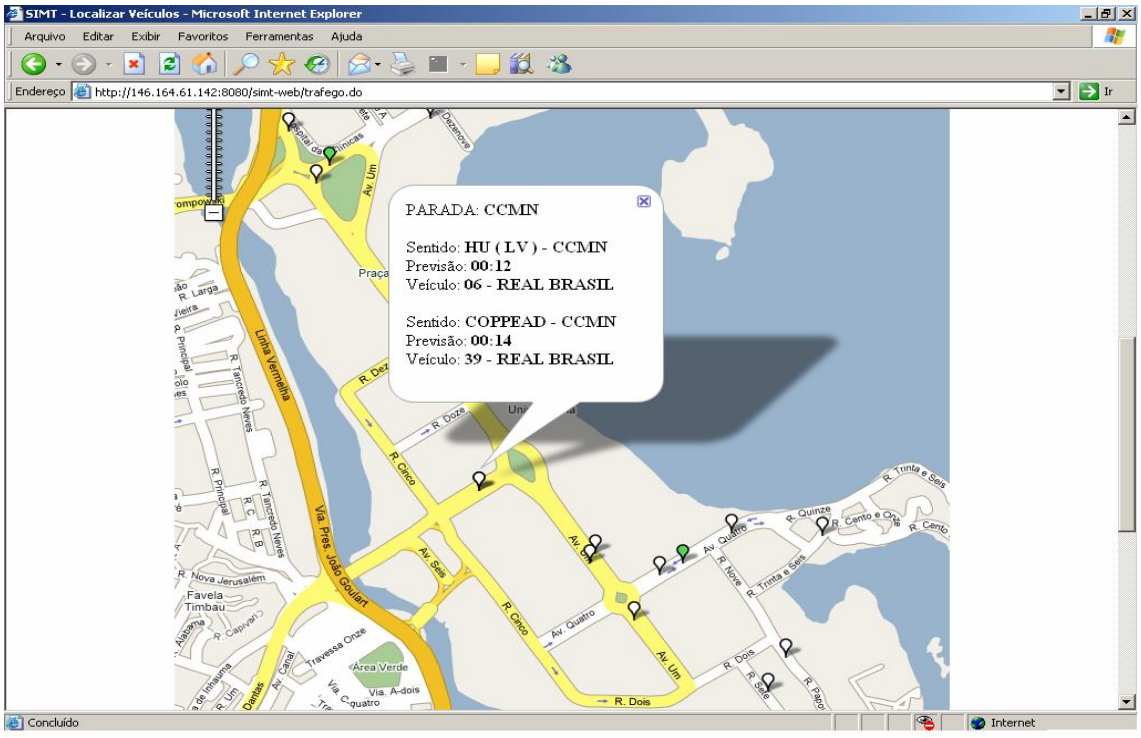

Figura 45 : Interface *Web* – Localizar Veículo.

5. CONCLUSÕES

O sistema finalizado atendeu aos requisitos de projeto com um desempenho satisfatório. Contudo, no decorrer do projeto algumas funcionalidades adicionais e melhorias foram sendo identificadas, mas não foram implementadas, evitando que o escopo e o cronograma do projeto fossem muito estendidos.

A melhoria da interface gráfica do usuário é uma tarefa muito importante, para dar um aspecto mais profissional ao sistema. Ela foi deixada em segundo plano, em favor da implementação da lógica de negócios, mas deve ser aprimorada.

Para uma melhoria da interface via celular é necessário a adição de mapas geográficos. As aplicações Java para celulares tem se tornando cada vez mais freqüentes e a API *Google Maps* já possui uma interface própria para este tipo de aplicação. Esta tarefa talvez seja a mais importante no momento e requer um novo projeto para o componente utilizar a *J2ME* (*Java 2 Micro Edition*), plataforma Java para aplicações de celular.

Em relação às funções de análise oferecidas pelo sistema, algumas melhorias podem ser desenvolvidas. A começar pelas opções de pesquisa. Além da data, o programa pode realizar a pesquisa pela hora do dia, o que facilitaria em muito o trabalho do analista. Pesquisas semanais ou mensais também podem ser adicionadas. Outra melhoria a ser desenvolvida diz respeito ao formato de exibição dos relatórios quando são exportados. O layout pode ser melhor organizado.

Um complemento que precisa ser implementado é a exibição do tempo de saída do último veículo a deixar uma determinada parada, além do tempo de chegada do próximo veículo. Devido a margem de erro do sistema, isto permite ao usuário a possibilidade de encontrar um veículo que supostamente já deveria ter deixado a parada, mas que na realidade encontra-se chegando na mesma, conforme comentado na seção 4.5.

Uma funcionalidade de extrema utilidade para pesquisas de transporte público é a contagem de passageiros. Poderia ser desenvolvido um dispositivo de contagem que, acoplado ao aparelho de rastreamento, informaria o número de passageiros a cada registro. Esta informação permitiria um melhor controle sobre a operação do veículo em tempo real, além de apoiar estudos sobre a demanda de passageiros na linha de transporte.

Por último, temos que a conexão do sistema com painéis instalados nas paradas facilitaria ainda mais o acesso às informações. Tal funcionalidade requer um projeto bem elaborado devido à utilização de interfaces de hardware pelo sistema.

A engenharia de software permitiu colocar em prática o planejamento e a organização de projetos. Ao longo do desenvolvimento, foi possível perceber o grande número de alterações que acontecem num projeto, além das melhorias exigidas após seu término. Essa necessidade por mudanças confirmou também a eficácia do desenvolvimento em iterações, onde a realização de testes é fundamental para a validação dos requisitos. O aprendizado dos modelos *UML* foi muito importante, pois permite criar especificações de alto nível. A documentação do projeto ajudou a melhorar a capacidade em organizar textos claros e objetivos.

Em relação às tecnologias utilizadas, todas corresponderam às expectativas do projeto. O estudo da plataforma *J2EE* e sua aplicação no projeto garantiram um bom nível de conhecimento. A utilização dos padrões reduziram o esforço de implementação e tornaram o sistema mais flexível às mudanças. Além disso, os aplicativos e ferramentas também deram a oportunidade de trabalhar com o que existe de mais recomendado no mercado.

A idéia desta ferramenta demonstrou-se interessante e sua aplicação prática é garantida. Ela pode servir de exemplo para que outras aplicações deste tipo apareçam, contribuindo com a melhoria do transporte público urbano e a qualidade de vida nas cidades.

Para concluir, além do conhecimento teórico e prático adquirido, a capacidade de enfrentar obstáculos sem desistir foi uma experiência a ser tirada como exemplo. Os momentos difíceis serviram de oportunidade para se aprender mais em busca de soluções e para mostrar a importância da dedicação e organização em nosso dia-a-dia.

BIBLIOGRAFIA

A lista a seguir é a bibliografia recomendada. Todos os sites da Internet citados abaixo foram registrados no mês de abril de 2008.

[1]. Roger S. Pressman, *Engenharia de Software*, Editora McGraw-Hill, 6ª Edição.

[2]. Craig Larman, *Utilizando UML e Padrões*, Editora Bookman, 2ª Edição.

[3]. Alur Deepak, Crupi John e Malks Dan, *Core J2EE Patterns*, Editora Campus, 2002.

[4]. Ted Husted, Cedric Dumoulin, George Franciscus, David Winterfeldt, *Struts em Ação*, Editora Ciência Moderna, 2004.

[5]. Helder L.S. da Rocha, *Curso J500 – Aplicações Distribuídas com J2EE e JBoss*, http://www.argonavis.com.br

[6]. Guilherme Silveira, Paulo Silveira, Sérgio Lopes, *Apostila FJ-11 – Java e Orientação a Objetos*, http://www.caelum.com.br

[7]. Plataforma *J2EE*, http://java.sun.com/javaee/

[8]. Servidor de Aplicação *JBoss*, http://www.jboss.org

[9]. *MySQL*, http://www.mysql.com

[10]. *FirebirdSql*, http://www.firebird.com.br/wiki/

[11]. *Framework Struts*, http://struts.apache.org

[12]. *Framework OpenSwing*, http://oswing.sourceforge.net

[13]. *API Google Maps*, http://www.google.com/apis/maps

[14]. *Java Service Wrapper*, http://wrapper.tanukisoftware.org

[15]. *Apache Virtual Hosts*, http://httpd.apache.org/docs/2.0/vhosts/examples.html

[16]. *IDE Eclipse*, http://www.eclipse.org

[17]. *DBManager*, http://www.dbtools.com.br

[18]. *Openwave SDK*, http://developer.openwave.com

[19]. Forum de discussão *GUJ*, http://www.guj.com.br

[20]. Comunidade *JavaFree*, http://www.javafree.org

[21]. Portal Java, http://www.portaljava.com