UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LUAN LEONARDO BOTURA

SISTEMA PARA CONTROLE DO PARQUE DE MÁQUINAS DE UMA PREFEITURA

PATO BRANCO
2011

LUAN LEONARDO BOTURA

SISTEMA PARA CONTROLE DO PARQUE DE MÁQUINAS DE UMA PREFEITURA

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, Campus Pato Branco, como requisito parcial para obtenção do título de Tecnólogo.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2011

ATA Nº: **177**

DEFESA PÚBLICA DO TRABALHO DE DIPLOMAÇÃO DO ALUNO LUAN **LEONARDO BOTURA**.

Às 09:35 hrs do dia 4 de julho de 2011, Bloco S da UTFPR, Campus Pato Branco, reuniu-se a banca avaliadora composta pelos professores Beatriz Terezinha Borsoi (Orientadora), Géri Natalino Dutra (Convidado) e Marco Antonio de Castro Barbosa (Convidado), para avaliar o Trabalho de Diplomação do aluno Luan Leonardo Botura, matrícula 609080, sob o título **Sistema para Controle de um Parque de Máquinas de uma Prefeitura**; como requisito final para a conclusão da disciplina Trabalho de Diplomação do Curso Superior de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, Coordenação de Informática. Após a apresentação o candidato foi entrevistado pela banca examinadora, e a palavra foi aberta ao público. Em seguida, a banca reuniu-se para deliberar considerando o trabalho **APROVADO**. Às 10:15 hrs foi encerrada a sessão.

Profa. Beatriz Terezinha Borsoi, Dr.
Orientadora

Prof. Géri Natalino Dutra, M.Sc.
Convidado

Prof. Marco Antonio de Castro Barbosa, Dr.
Convidado

Prof. Omero Francisco Bertol, M.Sc.
Coordenador do Trabalho de Diplomação

Prof. Edilson Pontarolo, Dr.
Coordenador do Curso

RESUMO

BOTURA. Luan Leonardo. Sistema para controle do parque de máquinas de uma prefeitura. 2011. 57 f. Monografia (graduação de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) - Universidade Tecnológica Federal do Paraná, 2011.

A automatização de sistemas é útil na medida em que reduz a quantidade de atividades realizadas manualmente (especialmente as repetitivas), agiliza a realização de atividades e provê controle mais eficiente das operações de negócio realizadas. Um sistema para controle do parque de máquinas auxilia no gerenciamento das despesas de cada veículo e máquina da frota, nas manutenções realizadas, sejam preventivas ou corretivas, e acompanhamento da vida útil da frota. O uso da orientação a objetos, na modelagem e implementação de um sistema, auxilia a entender e representar as entidades de negócio como um conjunto de objetos que podem ser caracterizados por atributos e operações que realizam e que se comunicam por troca de mensagens. O desenvolvimento de um *software*, seja orientado a objetos ou não, é dependente de um processo que determina o seu ciclo de vida. Como forma de fornecer uma maneira de auxiliar no controle do parque de máquinas de uma prefeitura, neste trabalho é reportada a implementação de um sistema para essa finalidade utilizando conceitos de orientação a objetos. Este trabalho também apresenta alguns processos de desenvolvimento de *software*.

Palavras-chave: Linguagem Delphi. Orientação a objetos. Controle de parque de máquinas.

LISTA DE FIGURAS

LISTA DE LISTAGENS DE CÓDIGO

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade / Consistência / Isolamento / Durabilidade
ADO	*ActiveX Data Objects*
AOO	Análise Orientada a Objetos
API	*Application Programming Interface*
BDE	*Borland Database Engine*
CLX	*Component Libraries for Cross-platform*
DB	*Data Base*
HTML	*HyperText Markup Language*
IBX	*Interbase Express*
IDE	*Integrated Development Environment*
IECA	Incluir, Excluir, Consultar e Alterar
IP	*Internet Protocol*
IPVA	Imposto sobre a Propriedade de Veículos Automotores
JDBC	*Java Database Connectivity*
ODBC	*Open Data Base Connectivity*
OLEDB	*Object Linking and Embedding Data Base*
OMG	*Object Management Group*
PDF	*Portable Document Format*
PHP	*Hypertext Preprocessor*
POO	Projeto Orientado a Objetos
PSQL	*PostgreSQL*
RTF	*Rich Text Format*
RUP	*Rational Unified Process.*
SGBD	Sistema de Gerenciamento de Banco de dados
SQL	*Structured Query Language*
TMO	Técnicas de Modelagem de Objetos
UML	*Unified Modeling Language*
VCL	*Visual Classes Library*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações inicias com uma visão geral do trabalho, os objetivos, a justificativa e a organização do texto.

1.1 CONSIDERAÇÕES INICIAIS

De maneira geral, o desenvolvimento de *software* é tratado como projeto. São realizadas várias atividades, por profissionais distintos que utilizam tecnologias e recursos diversos para definir e implementar um sistema ou aplicativo denominado *software*. Esses projetos são muitas vezes complexos e o seu desenvolvimento ocorre em ambientes dinâmicos nos quais pode haver alterações em requisitos de negócio e nas tecnologias utilizadas. Uma das causas das alterações é que os usuários do sistema (os clientes) não possuem, muitas vezes, certeza das suas necessidades e frequentemente mudam os requisitos definidos para o sistema, mesmo após a sua implementação.

Além das possíveis alterações definidas pelos usuários e cliente, estão os problemas causados pelo desconhecimento por parte da equipe das tecnologias utilizadas e pela falta de experiência, tanto nas atividades relacionadas à análise como à implementação de *software*. Como consequência, a produção de *software* é caracterizada por custos excedentes, entregas atrasadas, baixa confiabilidade e descontentamento do usuário.

Atualmente existem diversas contribuições científicas, técnicas e práticas na tentativa de reduzir os problemas apresentados tanto no processo de desenvolvimento como no produto, que é o *software*. Dentre essas iniciativas citam-se a participação ativa e constante do cliente e/ou usuário no processo - como define uma das premissas do *eXtreme Programing* ou programação aos pares (HIGHSMITH, 2010) - e a definição de marcos de verificação e validação periódicos como elemento de garantia de qualidade, como estabelecido pelo processo iterativo e incremental (BOOCH, RAMBAUGH, JACOBSON, 2001).

Contudo, é ainda bastante evidente a existência de problemas que podem ser verificados por meio de dados estatísticos como os apresentados em Lycett et al. (2003). Segundo esses autores, 80 a 90% dos sistemas de *software* são entregues com atraso e com custos acima do orçamento original; cerca de 40% dos projetos falham ou são abandonados; menos de 25% dos sistemas integram adequadamente negócio e tecnologia e somente entre 10 a 20% são entregues de acordo com o planejamento de prazo, orçamento e qualidade.

O relatório de 2004 do Standish Group[1] mostra que 29% de todos os projetos obtiveram sucesso (entregues em tempo, dentro do orçamento e com as características e funções requeridas); 53% sofreram alterações (atraso, aumento em orçamento e com menos características ou funções que requerido) e 18% falharam (cancelados antes da conclusão ou entregues e nunca utilizados). Os números do Standish Group de 2003 são um pouco mais otimistas em relação ao percentual de projetos que terminaram sem entregar resultados, foram 15%. Contudo, esses números apontam que 66% dos projetos foram considerados como não atendendo as necessidades do usuário.

Esse contexto ressalta a necessidade da existência de melhorias no desenvolvimento de *software*. Uma das possibilidades é atuar no processo de desenvolvimento, como o uso de orientação a objetos, pela forma de tratamento dos requisitos do sistema defindos em entidades que contém atributos e operações sobre esses atributos. Neste trabalho é apresentada a implementação de um sistema para controle do parque de máquinas de uma prefeitura, com uso de orientação a objetos. A modelagem inicial desse sistema foi realizada como trabalho de estágio.

1.2 OBJETIVOS

O objetivo geral apresenta o resultado principal do trabalho realizado e os objetivos específicos o complementam, no sentido de valores agregados.

1.2.1 Objetivo Geral

- Implementar um aplicativo computacional para o controle do parque de máquinas de uma prefeitura.

1.2.2 Objetivos Específicos

- Prover controle de gastos com veículos e de manutenção preventivas para o parque de máquinas de uma prefeitura.

[1] disponível em http://www.standishgroup.com/sample_research/PDFpages/q3-spotlight.pdf

- Possibilitar o controle de pagamento de imposto e seguro obrigatório dos veículos que compõem o parque de máquinas de uma prefeitura.

1.3 JUSTIFICATIVA

O uso de orientação a objetos tem se tornado comum no desenvolvimento de sistemas atuais, pelas facilidades que essa técnica apresenta e pelas tecnologias e ferramentas, como linguagens de programação e de modelagem de sistemas, existentes que a utilizam. Assim, verificou-se a possibilidade de utilizá-la na modelagem e na implementação de um sistema para controle de um parque de máquinas de uma prefeitura. É uma forma de exemplificar o uso de conceitos que visam entender o negócio (escopo do problema) e a implementação computacional desse negócio (escopo da solução) como um conjunto de objetos que interagem entre si.

A justificativa da escolha do tipo de sistema se deve pelo fato que o autor deste trabalho é funcionário da empresa, no caso a prefeitura, que serve de base para a definição dos requisitos do sistema. Assim, o levantamento dos requisitos e mesmo os testes de usuário são facilitados.

O sistema desenvolvido como resultado deste trabalho é para uma prefeitura específica. E, portanto, os requisitos são definidos de acordo com uma análise detalhada para essa instituição. Possibilitando, assim, um acompanhamento dos possíveis usuários, das suas necessidades e interesses. Contudo, como é provido o controle de todos os veículos, utilitários, equipamentos agrícolas (como roçadeiras) e máquinas pesadas (retroescavadeiras, tratores e outros) o sistema pode ser utilizado por prefeituras em geral.

1.4 ORGANIZAÇÃO DO TEXTO

Este texto está organizado em capítulos, dos quais este é o primeiro e apresenta as considerações iniciais, os objetivos e a justificativa.

O Capítulo 2 contém o referencial teórico do trabalho, centrado na modelagem e no desenvolvimento de sistemas orientados a objetos.

No Capítulo 3 estão as tecnologias utilizadas para a modelagem e a exemplificação da

implementação do sistema.

O Capítulo 4 apresenta o sistema modelado e a exemplifica a sua implementação.

Por fim, estão a conclusão e as referências bibliográficas utilizadas no desenvolvimento deste trabalho.

2 ANÁLISE E PROJETO ORIENTADOS A OBJETOS

Este capítulo contém a referencial teórico do trabalho e se refere à orientação a objetos e processo de desenvolvimento de software. A seção que ser refere à orientação a objetos que é complementada com conceitos relacionados à Linguagem de Modelagem Unificada.

2.1 ORIENTAÇÃO A OBJETOS

A orientação a objetos é uma maneira de modelar e construir sistemas que suportam uma variedade de técnicas para analisar, projetar e implementar sistemas flexíveis e robustos, provendo benefícios tais como encapsulamento, polimorfismo, herança e reusabilidade (BOOCH, 1991).

Um objeto é caracterizado basicamente por identidade (que o individualiza dos demais objetos), atributos (que são as características que definem a estrutura do objeto, representando os seus dados) e comportamento (que são as operações realizadas pelo objeto sobre os seus atributos, os dados). O comportamento é definido em termos de operações que são implementadas por métodos definidos por uma linguagem de programação. Uma mesma operação pode ser implementada por métodos distintos, definindo polimorfismo. Um objeto pode ser herdado, em termos de atributos e operações, por outro objeto. É o mecanismo de herança.

A UML (*Unified Modeling Language*) ou Linguagem de Modelagem Unificada (BOOCH, RAMBAUGH, JACOBSON, 2000) estabelece uma notação para especificar e documentar sistemas de *software* orientados a objetos. A notação UML é útil para representar graficamente modelos orientados a objetos; especificar requisitos de sistema e capturar decisões de projeto; e promover comunicação entre envolvidos no projeto de desenvolvimento de um sistema (AGARWAL, SINHA, 2003).

A UML é uma linguagem de modelagem que especifica um vocabulário (símbolos), a semântica e a sintaxe desses símbolos e a forma de uso dos mesmos. Essa linguagem possui duas classes de construtores de modelagem ou visões, chamadas diagramas e são (WAGENHALS et al., 2002):

a) Diagramas estruturais – representam elementos como classes, objetos, componentes e a distribuição das partes do sistema, documentando os aspectos estáticos do sistema sendo modelado;

b) Diagramas comportamentais – representam ações como atividades, colaborações entre objetos, sequência de troca de mensagens entre objetos, estados que os objetos assumem e casos de uso, referindo-se ao comportamento dinâmico do sistema.

Cada um desses diagramas representa um aspecto específico do mesmo sistema. Existem três fases básicas para o desenvolvimento de *software* utilizando métodos orientados a objeto: análise orientada a objetos (AOO), projeto orientado a objetos (POO) e a implementação (POTOK, MLADEN, 1995). A análise e o projeto se referem à modelagem do sistema e a implementação à sua codificação por meio de uma linguagem de programação.

2.1.1 Modelagem Orientada a Objetos

A estrutura básica da modelagem - análise e projeto - baseados em objetos é o objeto que combina a estrutura e o comportamento dos dados em uma única entidade (BOOCH, RUMBAUGH, JACOBSON, 2001). Técnicas de modelagem de objetos abrangem desde a definição inicial do sistema até a sua implementação.

Para a definição inicial do sistema pode ser elaborado um modelo que sumarize os aspectos essenciais do domínio do problema e do contexto da solução. Esse modelo contém os objetos encontrados no domínio da aplicação, incluindo uma descrição das propriedades dos mesmos e do seu comportamento. Os objetos do domínio da aplicação compõem a estrutura do modelo em termos de negócio e não de objetos computacionais. Desse modelo são definidas classes com os atributos e as operações que serão realizadas pelos respectivos objetos. É a fase de análise e projeto. Esses modelos são utilizados na definição das tabelas com seus campos que compõem o banco de dados e a implementação do sistema, com as classes e os seus atributos e métodos. As classes podem ser definidas considerando mecanismos como herança e os métodos por polimorfismo.

Uma metodologia baseada em objetos consiste na construção de um modelo de um domínio da aplicação e na posterior adição a este dos detalhes de implementação durante o projeto do sistema. Essa abordagem, referenciada como TMO (Técnicas de Modelagem de Objetos) por Rumbaugh et al. (1997), é composta das seguintes fases (POTOK, MLADEN, 1995):

a) Análise – o modelo de análise é uma abstração concisa e precisa do que o sistema desejado deverá fazer. Os objetos desse modelo devem ser conceitos do domínio da aplicação e não conceitos de implementação computacional, como, por exemplo, a estrutura de dados. A análise orientada a objetos é o processo que define um modelo que resolve um determinado problema ou encontra um conjunto de requisitos. Este modelo é, em termos do mundo real, o espaço do problema. Consiste de um conjunto de objetos com as interações entre os mesmos e pode incluir os processos envolvidos, estados e demais interações entre objetos.

b) Projeto do sistema e dos objetos – o projeto do sistema é a organização do sistema em subsistemas baseados tanto na estrutura da análise como na arquitetura proposta. O projeto dos objetos constrói um modelo de projeto baseado no modelo de análise, mas contendo detalhes de implementação. O enfoque são as estruturas de dados e os algoritmos necessários à implementação de cada classe. As classes de objetos provenientes da análise são acrescidas das estruturas de dados do domínio computacional e de algoritmos definidos para resolver, implementar, a solução definida. Na fase de projeto orientado a objetos ocorre a transformação do modelo de análise para a descrição de projeto.

c) Implementação – as classes e os relacionamentos são traduzidos para implementação por meio de linguagens de programação. A implementação orientada a objetos é um refinamento do modelo de projeto em um *software* executável. O uso da orientação a objetos pode aumentar a produtividade porque a AOO e o POO, se elaborados corretamente, facilitam a implementação, os testes e provem o reuso, além de reduzir o esforço de manutenção.

A metodologia TMO faz uso de três tipos de modelos para descrever um sistema (RUMBAUGH et al., 1997):

a) Modelo de objetos – descreve a estrutura estática de um sistema em termos de objetos e relacionamentos correspondentes a entidades do mundo real por meio da identidade, relacionamentos, atributos e operações dos objetos.

b) Modelo dinâmico – descreve as interações entre os objetos do sistema, são os aspectos que se modificam com o tempo. É usado para especificar e implementar os aspectos de controle do sistema. Descreve também aspectos do sistema relacionados à sequência de operações – eventos que assinalam modificações, sequência de eventos, estados que definem o contexto para os eventos e a organização de eventos e estados.

c) Modelo funcional – descreve as transformações dos valores de dados de um sistema: funções, mapeamentos, restrições e dependências funcionais. Abrange o que o

sistema faz, independentemente de como ou quando é feito. Descreve a estrutura computacional de um sistema em termos de valores e funções. Especifica os resultados de um processamento sem especificar como ou quando são processados.

Cada um desses modelos descreve um aspecto do sistema, mas contém referências aos outros modelos. O modelo de objetos descreve a estrutura de dados sobre a qual atuam os modelos dinâmico e funcional. As operações do modelo de objetos correspondem a eventos do modelo dinâmico e funções do modelo funcional. O modelo dinâmico está relacionado a estrutura de controle de objetos e apresenta as decisões que dependem dos valores dos objetos e que provocam ações que modificam esses valores e que chamam funções. O modelo funcional se refere às funções chamadas pelas operações do modelo de objetos e pelas ações no modelo dinâmico. As funções operam sobre os valores de dados especificados pelo modelo de objetos. O modelo funcional também contém restrições relativas aos valores dos objetos.

2.1.2 Técnicas de Modelagem Orientadas a Objetos

A ideia central do paradigma estruturado é determinar as funcionalidades do sistema. E do paradigma orientado a objetos, o objetivo principal é a identificação de objetos a partir das necessidades levantadas com o cliente para o sistema de *software* a ser desenvolvido.

Para Li et al. (2000), o projeto de sistemas orientado a objetos não tem pontos definidos para início e fim no processo de desenvolvimento de sistemas. Esse projeto inicia com a análise do domínio da aplicação e termina somente quando o sistema estiver totalmente implementado. Um sistema orientado a objetos abrange desde a abstração do domínio de uma aplicação até a implementação completa das classes. Os conceitos identificados a partir da abstração dos requisitos da aplicação são mapeados em classes que se comunicam por um mecanismo denominado troca de mensagens.

Os conceitos principais em termos de modelagem de sistemas orientados a objetos são (BOOCH, RAUMBAUGH, JACOBSON, 2000):

a) Classe - é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações (métodos) e relacionamentos. Exemplo: classe veículos.

b) Objeto - é uma ocorrência específica (uma instância) de uma classe. Exemplo: o veículo da marca ABC que possui o chassi número 1234567ABC. Um objeto de acordo com Booch (1991) é caracterizado por:

b.1) Identidade – é a identificação exclusiva de cada um dos objetos pertencentes a uma mesma classe, é a propriedade que o permite distinguir dos demais objetos;

b.2) Estado – o estado de um objeto compreende todas as propriedades (geralmente estáticas) do objeto mais os valores atuais (geralmente dinâmicos) de cada uma dessas propriedades.

b.3) Comportamento – o comportamento define como um objeto age e reage em termos das suas mudanças de estados e troca de mensagens.

c) Mensagem – os objetos se comunicam por mensagens enviadas de um objeto a outro requisitando um serviço por meio da execução de uma operação. O ciclo é completo, ou seja, uma mensagem é enviada a um objeto; operações são executadas nesse objeto com base nos dados de seu alcance na hierarquia de classes; e uma mensagem contendo o resultado da operação é enviada ao objeto solicitante.

d) Polimorfismo – se refere aos vários comportamentos que uma mesma operação pode assumir. São métodos distintos que se aplicam a uma mesma operação de um determinado objeto.

e) Herança – é a capacidade de um novo objeto herdar atributos e operações de um objeto existente.

As técnicas orientadas a objetos têm como principal objetivo identificar classes e seus relacionamentos que estão nas regras de negócios do sistema de informação a ser desenvolvido. A orientação a objetos é um passo importante para a reusabilidade de partes de *software*, ou seja, a partir das classes definidas para um projeto, essas podem ser utilizadas ou evoluídas para outros projetos (MENASCÉ, GOMAA, 2000).

A UML é uma linguagem gráfica definida como padrão para a orientação a objetos pela OMG (*Object Management Group*) (LARMAN, 2000). A UML é utilizada por métodos de análise e de projeto para expressar projetos de *software* orientados e objetos (FOWLER, SCOTT, 2000).

Por meio da UML são definidos diagramas que possibilitam a visualização do sistema de *software* sob diferentes perspectivas. De certa forma, cada diagrama constitui uma visão do sistema. Uma visão representa o sistema sob o ponto de vista de determinados usuários, ou de acordo com determinados interesses. É colocado “de certa forma” porque a UML segue as visões de arquitetura definidas pelo Processo Unificado (BOOCH, RAMBAUGH, JACOBSON, 2000) que são: visão de projeto, visão do processo, visão da implementação, visão da implantação e visão de caso de uso.

A UML, na versão 1.3, é composta por nove diagramas (BOOCH, RUMBAUGH, JACOBSON, 2000) que são: casos de uso, classes, objetos, sequência, colaboração, estado, atividades, componentes e implementação. A UML, na versão 2.0, é composta por treze diagramas agrupados em diagramas estruturais, comportamentais e de interação (AMBLER, 2004):

a) Diagramas estruturais:

a.1) Diagrama de classes;

a.2) Diagrama de objetos;

a.3) Diagrama de componentes;

a.4) Diagrama de instalação;

a.5) Diagrama de pacotes;

a.6) Diagrama de estrutura;

b) Diagramas comportamentais:

b.1) Diagrama de casos de uso;

b.2) Diagrama de transição de estados;

b.3) Diagrama de atividade;

c) Diagramas de interação:

c.1) Diagrama de sequência;

c.2) Diagrama de interatividade;

c.3) Diagrama de comunicação (colaboração);

c.4) Diagrama de tempo.

Diagramas de casos de uso e de classe foram utilizados para representar a modelagem do sistema implementado como resultado deste trabalho. A seguir esses dois tipos de diagramas são apresentados e exemplificados.

a) Diagrama de casos de uso - os requisitos funcionais do sistema são definidos em forma de casos de uso. Um caso de uso descreve a interação entre o ator (que pode ser um usuário, um dispositivo ou outro sistema) e o sistema sendo considerado ou implementado. Na fase de análise de requisitos, o caso de uso considera o sistema como uma “caixa preta” e descreve as interações entre o ator e o sistema em forma de uma narrativa composta de entradas do ator e as respostas do sistema. Essa descrição será utilizada na identificação das classes do sistema de informação.

A Figura 1 mostra um exemplo do diagrama de caso de uso com as interações entre o ator denominado “Usuário comum” para um sistema de controle de um parque de máquinas.

As interações são: IECA (incluir, Excluir, Consultar e Alterar) veículo e IECA tipo de veículo.

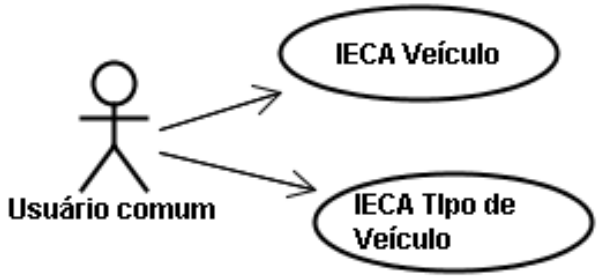

Figura 1 – Exemplo de diagrama de casos de uso

O diagrama apresentado na Figura 1 é parcial, apresenta apenas dois casos de uso, mas tem o objetivo de exemplificar elementos de composição de um diagrama de caso de uso que são atores (Usuário comum) e casos de uso (IECA Veículos e IECA Tipo de veículo) e os relacionamentos entre o ator e os casos de uso. Nesse exemplo, a seta direcionada indica que o ator é quem inicia o respectivo caso de uso.

b) Diagrama de classes – exibem classes, interfaces e colaborações, bem como os seus relacionamentos.

A Figura 2 exemplifica as classes veículo e tipo de veículo

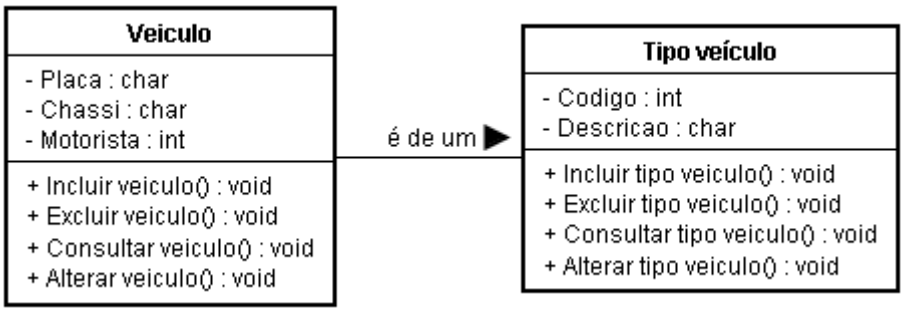

Figura 2 – Exemplo de diagrama de classes

Os atributos da classe Veículo são Placa, Chassi e Motorista e possui como operações Incluirveiculo(), Excluirveiculo(), ConsultarVeiculo() e Alterarveiculo(). Ressalta-se que as operações estão representadas sem parâmetros e sem retorno, em Excluirveiculo(), por exemplo, seria informado o identificador único do veículo, que poderia ser a placa, e como retorno um valor lógico (booleano) indicando que a operação foi ou não realizada com êxito. O relacionamento indica que um veículo está associado a um tipo (um veículo é de um

determinado tipo ou um veículo possui um tipo). Isso é representado pela seta direcionada que liga as classes Veículo e Tipo veículo.

2.2 PROCESSO DE DESENVOLVIMENTO DE SOFTWARE

Um processo de *software* fornece organização e controle às atividades relacionadas ao desenvolvimento de *software*. Contudo, os processos não devem ser muito complexos ou excessivos em atividades e documentação de controle porque pode desestimular o seu uso ou mesmo serem utilizados de forma parcial ou incorreta. De maneira geral, o processo de *software* pode ser compreendido como o conjunto de todas as atividades necessárias para transformar os requisitos do usuário em *software* (HUMPHREY, 1989).

A descrição do processo de *software* caracteriza um modelo. Várias informações devem ser integrados em um modelo de processo para indicar quem, quando, onde, como e por que os atividades que compõem esses processos são realizadas. A execução de um processo ou de um conjunto de processos realiza um projeto de software. Para Conradi et al. (1994), projeto é a instância de um processo, com objetivos e restrições específicas. Assim, pode-se dizer que um projeto é um esforço para desenvolver um produto de *software*, ou seja, envolve uma estrutura organizacional, prazos, orçamentos, recursos e um processo de desenvolvimento.

2.2.1 Modelos de Processos

Os modelos da engenharia de *software* dividem o processo de desenvolvimento em fases ou etapas e associam atividades a cada uma dessas fases. Embora os modelos difiram quanto a quantidade, sequência e abordagem das fases e das atividades, em geral os processos envolvem: a definição dos requisitos do sistema, a modelagem desses requisitos de forma a representar a solução para o problema computacional definido na fase de requisitos, a implementação dessa solução, os testes e a implantação do sistema. O tipo de sistema, a complexidade e o tamanho definem a documentação necessária e a forma de realização das fases, se realizadas apenas uma vez ou em ciclos complementares.

Para Schmietendorf, Dimitrov e Dumke (2002) os modelos que tem prevalecido no ambiente prático são: em particular o modelo em cascata, o modelo V, modelos de processos

evolucionário tal como o modelo em espiral e mais recentemente os modelos para desenvolvimento orientado a objetos, como o *Rational Unified Process* (RUP).

O modelo em cascata também é conhecido como ciclo de vida clássico ou modelo sequencial linear (PRESSMAN, 2002). Esse modelo usa uma abordagem estritamente sequencial, os resultados de uma fase são utilizados na próxima fase. Assim, cada fase precisa ser trabalhada completamente para que o projeto possa avançar. Esse modelo considera que os requisitos uma vez definidos não sofrem alterações e não é permitido trocá-los. Esse modelo traz dificuldade na implementação de sistemas complexos, grandes, nos quais os usuários estão incertos quantos aos requisitos do sistema ou que a equipe do projeto não tem muito conhecimento do domínio do problema ou da sua solução.

As principais fases do modelo sequencial linear como definido por Pressman (2002) estão representadas na Figura 3. Os retângulos com cantos arredondados significam atividades, o circulo com bordas delgadas determina o início do processo e o círculo no final (com borda mais espessa) o seu final. As setas que ligam as atividades representam a sequência em que as mesmas são realizadas, representando a dependência entre elas.

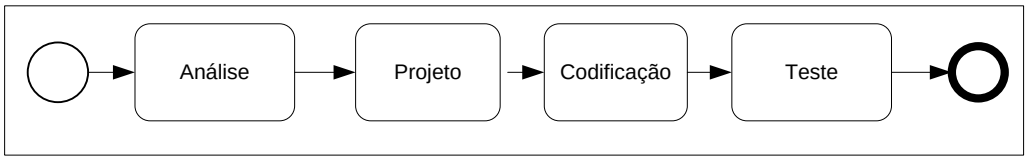

Figura 3 – O modelo sequencial linear
Fonte: Pressman (2002, p. 27).

De acordo com a Figura 3, as fases do modelo sequencial linear são:

a) Análise – nesta fase é realizado o levantamento, a definição e a análise dos requisitos do sistema. O escopo do projeto e as necessidades do cliente por meio do detalhamento das funcionalidades e suas restrições são definidos. Os requisitos e a sua análise definem o problema para o qual será implementada uma solução computacional.

b) Projeto – no projeto os requisitos definidos são modelados de forma a representar a solução computacional para o problema implementado. O projeto define como a solução será implementada.

c) Codificação - é a fase em que o projeto do *software* é transformado em programa ou aplicativo computacional por meio do uso de linguagens de programação, banco de dados e outros recursos tecnológicos.

d) Teste – nesta fase o programa é testando visando buscar erros, inconsistências e verificar se os requisitos estabelecidos foram atendidos.

A caracterização básica do modelo sequencial linear em que uma fase só é iniciada após a anterior ser finalizada e não há possibilidade de redefinição de atividades de fases já realizadas trouxe a necessidade de outros modelos, como o desenvolvimento baseado em protótipo, processos iterativos e incrementais e modelos de processos com ampla participação do cliente, dentre outros.

Modelos de processo evolucionário são caracterizados pelo uso de protótipos no início do ciclo de vida. Outra característica típica dessa abordagem é o uso de incrementos o que envolve prover rapidamente versões para clientes ou usuários e gradualmente implementar os requisitos definidos para o sistema.

O modelo espiral integra esses procedimentos, adicionado análise de riscos. Esse modelo possui tipicamente de três a seis regiões de tarefas que, de forma geral, abrangem (SOMMERVILLE, 2003): a) definição de objetivos, alternativas, parâmetros; b) avaliação de alternativas; reconhecer, avaliar e resolver riscos; c) implementar e verificar um produto intermediário; d) planejar as próximas iterações.

A Figura 4 exemplifica o modelo espiral sendo composto por: planejamento, análise de riscos, engenharia, construção e liberação, avaliação do cliente e comunicação com o cliente (SOMMERVILLE, 2003).

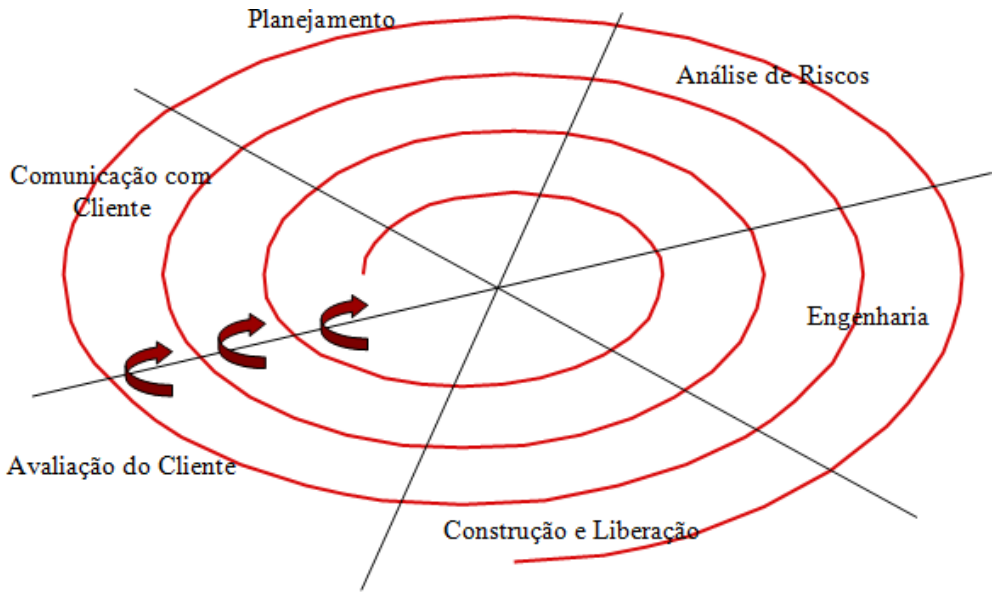

Figura 4 – Ciclo de vida espiral

Fonte: baseado em Sommerville (2003).

Booch, Raumbaugh e Jacobson (2001) propuseram um processo da Engenharia de Software que fornece uma abordagem disciplinada para assumir tarefas e responsabilidades em uma organização de desenvolvimento. Esse processo é denominado processo unificado.

O processo unificado visa minimizar riscos e utiliza a UML (BOOCH, RAUMBAUGH, JACOBSON, 2000) para a representação dos modelos e é definido como sendo iterativo e incremental. O desenvolvimento é baseado em casos de uso e centrado em arquitetura e componentes. A característica de ser iterativo e incremental determina que a sua evolução seja realizada por meio de iterações ao longo do tempo, com produtos bem definidos. Cada iteração pode conter as atividades de análise de requisitos, projeto, codificação e teste, dentre outros, em proporções variadas.

A Figura 5 apresenta um esquema dos fluxos de trabalho, fases e iterações do processo unificado.

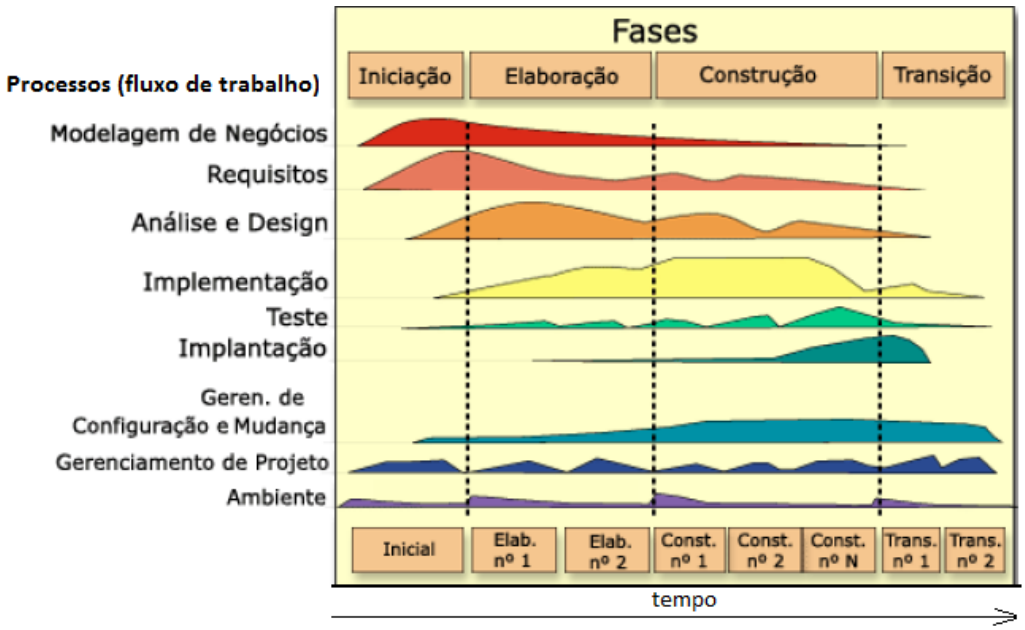

Figura 5 – Estrutura do processo unificado

Fonte: adaptado de Booch, Raumbaugh e Jacobson (2000).

As representações gráficas associadas a cada um dos fluxos de trabalho (Figura 5), indicam de forma ilustrativa o trabalho realizado em cada uma das fases e iterações. Essas representações são meramente ilustrativas, não significando que a quantidade de trabalho realizada corresponda a essa disposição. Contudo, elas fornecem a ideia que atividades de

quase todos os processos são realizadas em cada iteração. A seguir a descrição resumida das fases do Processo Unificado, como representado na Figura 5:

a) Fase de Concepção - estabelece os requisitos de forma que o sistema atenda as necessidades e interesses do usuário.

b) Fase de Elaboração - define uma arquitetura estável, por meio do refinamento dos requisitos identificados na fase de concepção.

c) Fase de Construção - implementação de um produto de *software* pronto como versão inicial operacional.

d) Fase de Transição - implantação do sistema para o ambiente do usuário. A implantação é feita à medida que partes do sistema são implementadas.

As fases são compostas por iterações e juntamente com os fluxos de trabalho (também conhecidos como processos, disciplinas ou *workflows*) compõem a essência do processo unificado.

As fases estabelecem as metas intermediárias a serem cumpridas durante o desenvolvimento. E por isso elas definem marcos de verificação. Essas verificações são técnicas e contam com a participação do cliente ou usuário do sistema. As avaliações definem ajustes no direcionamento do projeto e mesmo a determinação da sua continuidade ou não. Cada uma das iterações que compõe as fases representa um miniprojeto e correspondem às metas intermediárias do desenvolvimento. As iterações iniciais costumam ter mais atividades de requisitos e de análise e, com o andamento do processo, as iterações passam a incluir as atividades de implementação e teste com mais intensidade.

Os fluxos de trabalho agregam as atividades relacionadas que são realizadas não obrigatoriamente de forma sequencial, por diversos participantes. São definidos, como fluxos de trabalhos técnicos, por exemplo, os fluxos de requisitos, análise, projeto, implementação e teste. Os demais fluxos de trabalho são gerenciais ou administrativos e estão relacionados aos fluxos técnicos ou ao ambiente de desenvolvimento.

A estruturação do Processo Unificado resulta em um modelo mais flexível, pois permite definir as iterações mais apropriadas para cada projeto. O conjunto de fluxos de trabalhos técnicos pode ser complementado com os de gerência, garantia de qualidade, preparo de infra-estrutura, dentre outros.

3 MATERIAIS E MÉTODO

Este capítulo contém os materiais e o método utilizados para a análise e o desenvolvimento do sistema. Os materiais se referem às ferramentas e as tecnologias, incluindo linguagem de programação, banco de dados e ferramenta para análise e modelagem. O método contém os principais procedimentos utilizados no ciclo de vida do sistema, abrangendo da definição dos requisitos à implantação.

3.1 FERRAMENTAS E TECNOLOGIAS

As ferramentas e as tecnologias utilizadas são: a ferramenta Jude Community para a modelagem do sistema, a linguagem Delphi 2009 para implementação, o banco de dados Firebird com o IBExpert versão 1.5 para administração do banco de dados e o Rave Reports para os relatórios.

3.1.1 Jude Community

Jude é uma ferramenta de modelagem gratuita para projeto de sistemas orientados a objeto. Essa ferramenta foi criada com Java e é de uso fácil e intuitivo (CAMPOS, 2010). Ela foi descontinuada e substituída por Astah*, mas a interface e os tipos de diagramas permanecem, com alguns acréscimos e pequenas mudanças de interface de composição dos modelos.

Jude ou Jude Commuunity é baseada nos diagramas e na notação da UML 2.0 e permite gerar código em Java, mas esse código é apenas para a definição da classe e de seus atributos e métodos. A Figura 6 apresenta a tela inicial da ferramenta Jude. Ela é basicamente composta por uma área de edição, à direita, na qual são colocados os componentes gráficos para a representação dos modelos. Esses componentes são pré-definidos e estão representados na barra imediatamente superior à área de edição. Os componentes são apresentados de acordo com o tipo de modelo de diagrama que pode ser escolhido por meio do menu "*Diagram*".

Na área a esquerda da edição está um menu em forma de árvore que exibe os diagramas gerados e seus componentes e permite o acesso para edição dos mesmos. Esses

componentes são objetos definidos para compor os modelos a partir das classes que estão na barra de componentes.

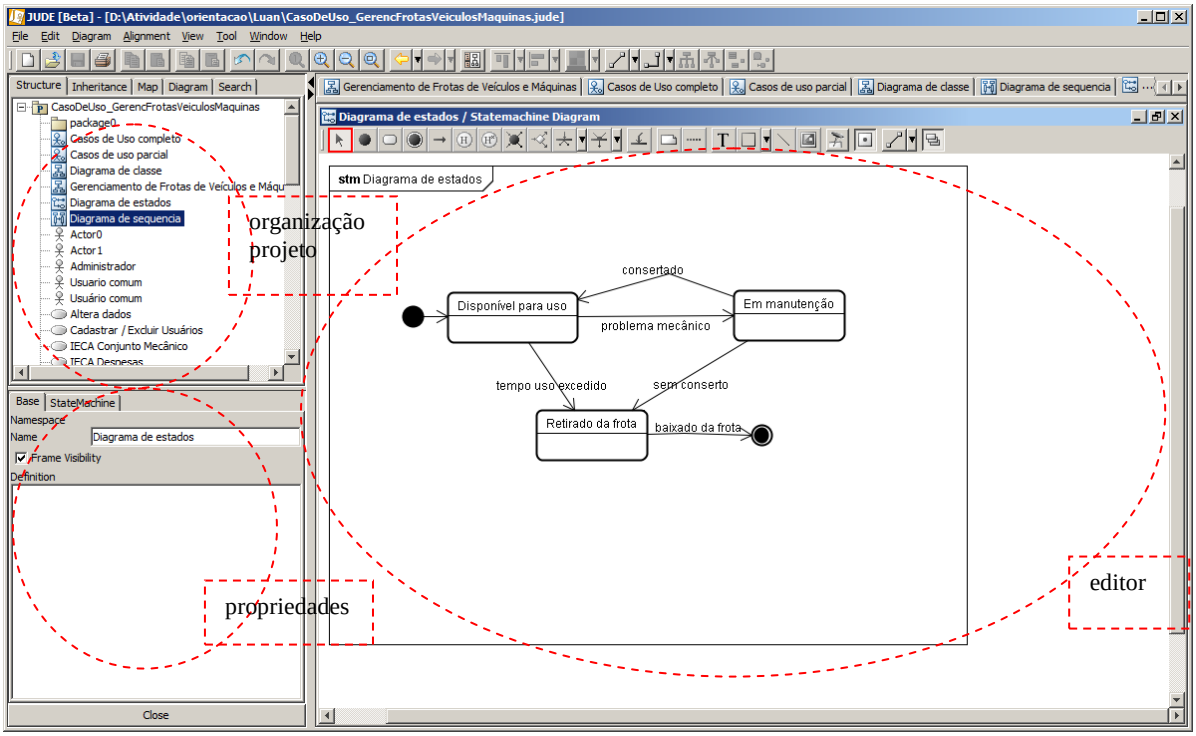

Figura 6 – Ferramenta de modelagem Jude

A ferramenta Jude está dividida em três partes principais, como pode ser visualizado na Figura 6:

a) Organização do projeto (área superior à esquerda da Figura 6) - é a área na qual estão as diferentes visões do projeto, são elas: *Support Structure Tree* que é a árvore de estrutura do projeto, *Inheritance Tree* que exibe as heranças identificadas, *MapView* para mostrar todo o editor de diagrama, *DiagramList* que mostra a lista de diagramas do projeto e *Search and Replace* para a localização de modelos e substituição de nomes.

b) Visão das propriedades (área inferior à esquerda da Figura 6) - é a área na qual podem ser alteradas as propriedades dos elementos do diagrama. As propriedades de um item selecionado são exibidas e podem ser editadas. Por exemplo, com um diagrama de casos de uso aberto e um ator selecionado são exibidas todas as propriedades definidas para esse ator, como nome e o tipo de visibilidade.

c) Editor do diagrama (área à direita da Figura 6) - é a área na qual são exibidos e compostos os diagramas. Ao ser selecionado um determinado diagrama, dos constantes na lista, o mesmo é carregado e todos os seus elementos gráficos são mostrados nesta área.

3.1.2 Linguagem Delphi

O ambiente de desenvolvimento Delphi se baseia em uma extensão orientada a objetos da linguagem de programação Pascal, também conhecida como Object Pascal (CANTU, 2005). Com esse ambiente de desenvolvimento, os recursos que o desenvolver mais necessita estão agrupadas, não sendo necessário instalá-las ou executá-las separadamente (CANTU, 2002).

Como ambiente de desenvolvimento, Delphi é um compilador e também uma IDE (*Integrated Development Environment*) para o desenvolvimento de *software*. A partir da versão 7.0, a linguagem utilizada pelo Delphi, o Object Pascal (Pascal com extensões orientadas a objetos) passou chamar-se Delphi Language. Esse ambiente tem as seguintes particularidades:

1) Visual - a definição da interface e até mesmo de parte da estrutura de um aplicativo Delphi pode ser realizada com o auxílio de ferramentas visuais. Por exemplo, uma tela é composta a partir de componentes, como formulários que contém os demais itens, botões para realizar ações, caixas de texto para a entrada de dados e rótulos para apresentar texto.

2) Orientada a objetos – com uso de conceitos de classe, herança e polimorfismo.

3) Orientada a eventos - cada elemento de uma interface de aplicativo é capaz de capturar e associar ações a uma série de eventos.

4) Compilada - a geração de código em linguagem de máquina agiliza a execução dos aplicativos.

O ambiente de desenvolvimento da linguagem Delphi consiste de vários elementos, ferramentas de projeto e de banco de dados. Esses elementos estão resumidamente apresentados no Quadro 1.

Elemento	Descrição
Formulário (*form*)	É uma janela, elemento básico no qual são colocados os componentes para compor a interface do sistema como o usuário.
Unidade (*unit*)	Arquivo que contém código em *object* pascal. Cada formulário possui uma unidade associada.
Componente	Elementos utilizados para a construção do sistema de *software*. Exemplo: botões, caixas de texto.
Propriedade	Representam os atributos dos componentes e dos objetos.
Método	São os procedimentos e as funções dos componentes e dos objetos.
Evento	Representam a capacidade de resposta dos componentes a estímulos.
Processador de eventos	Procedimento responsável para responder a determinado evento.
Projeto (*project*)	Conjunto de formulários, componentes e unidades que compõem um sistema de *software*.

Quadro 1 – Elementos da IDE Delphi

3.1.3 IBExpert

IBExpert (IBEXPERT, 2011) é uma ferramenta para administração de bancos de dados Interbase e Firebird que permite criar e gerenciar usuários e tabelas. A interface do aplicativo é bastante simplificada consistindo basicamente de uma barra de ferramentas e do DB (*Data Base*) Explorer.

A criação de um banco de dados é feita pela opção *Create Database* do Menu *Database*. Para criar uma nova base é necessário indicar:

a) *Server* - determina o tipo de servidor, se remoto ou local. A opção remoto permite optar pela criação em outras máquina de uma rede.

b) *Server Name* - se escolhida a opção remoto no campo anterior, é necessário informar o nome ou IP (*Internet Protocol*) do computador servidor da base de dados.

c) *Protocol* - indica o protocolo usado para efetuar a comunicação com o servidor. Selecionando a opção de servidor local, os campos *Server Name* e *Protocol* são desabilitados.

d) *Database* - nesse campo é informado o caminho da base de dados seguido do nome da base de dados acrescido de sua extensão.

e) *Client Library Name* - nesse campo é informada a biblioteca (.dll) (*dymanic-link library*) do banco.

Os outros campos são Usuário/Senha do Firebird (SYSDBA/*masterkey*), tamanho da página do arquivo, o *charset* (por exemplo, WIN1252), o dialeto SQL (*Structured Query Language)* e a opção para registrar a base de dados após sua criação.

O banco de dados criado precisa ser registrado. Para isso é necessário informar a versão do banco de dados e o *alias* a ser referenciado do arquivo, o banco de dados no DBExplorer.

3.1.4 Firebird

O banco de dados Firebird é derivado do código do Borland InterBase 6.0 e possui o código fonte aberto e gratuito. O Firebird é um SGBD (Sistema de Gerenciamento de Banco de dados) completo e possui suporte nativo para diversos sistemas operacionais, incluindo o Windows, Linux, Solaris, MacOS, *triggers* de conexão e transação (CANTU, 2010).

Dentre os recursos do Firebird destacam-se:

a) Transações compatíveis com ACID (Atomicidade, Consistência, Isolamento,

Durabilidade);

b) Integridade referencial;

c) Utiliza poucos recursos de processamento;

d) Linguagem nativa para *stored procedures* e *triggers* (PSQL (PostgreSQL));

e) Suporte para funções externas;

f) Versão *embedded* do SGBD;

g) Diversas formas de acesso ao banco de dados, como: nativo/API (*Application Programming Interface*), dbExpress, ODBC (*Open Data Base Connectivity)*, OLEDB (*Object Linking and Embedding Data Base)*, .Net provider, JDBC (*Java Database Connectivity)* nativo tipo 4, Python *module*, PHP (*Hypertext Preprocessor*), Perl;

h) *Backups* incrementais;

i) Tabelas temporárias.

3.1.5 Rave Reports

O Rave Reports é uma ferramenta de geração de relatórios integrada ao Delphi. Essa ferramenta é mantida pela empresa Nevrona Designs (www.nevrona.com) e possui um ambiente visual para a criação de relatórios. É uma ferramenta que permite tanto CLX (*Component Libraries for Cross-platform*) quanto VCL (*Visual Classes Library*) para plataformas Linux e Windows. As principais características e funcionalidades do Rave Reports são (BERLITZ, 2007):

a) Permite o desenvolvimento de relatórios com acesso direto ao banco de dados, utilizando diversas tecnologias, como: BDE (*Borland Database Engine)*, dbExpress, ADO (*ActiveX Data Objects)* e IBX (*Interbase Express);*

b) Possui um editor visual para criação de instruções SQL integrado ao ambiente;

c) Os relatórios podem ser salvos nos formatos RTF (*Rich Text Format)*, HTML (*HyperText Markup Language)*, PDF (*Portable Document Format)*, texto e um formato proprietário (ndr);

d) Possui diversos recursos para formatação dos relatórios, como alinhamento e posicionamento de objetos;

b) Os relatórios são baseados em páginas, regiões e bandas. É possível visualizar e testar os relatórios em tempo de projeto a partir do Rave Visual Designer;

d) Disponibiliza a criação de páginas padrão para serem usadas como base para vários relatórios;

e) Possibilita acesso aos objetos do relatório a partir da aplicação Delphi;

f) Possui componentes para suporte a código de barras, linguagem de programação própria a Rave Language para codificação de eventos do relatório;

O Rave Report trabalha com o conceito de projetos. Um projeto pode conter vários relatórios. Os relatórios podem ser distribuídos separadamente do executável da aplicação ou incorporados ao mesmo. Todos os relatórios da aplicação podem ser salvos em um único arquivo Rave (.rav).

3.2 ATIVIDADES PARA IMPLEMENTAÇÃO DO SISTEMA

A modelagem e a implementação do sistema para controle de um parque de máquinas para uma prefeitura reportado neste trabalho foram realizadas com base na orientação a objetos. Uma metodologia baseada em objetos é composta das fases: análise e projeto do sistema, projeto dos objetos e implementação (RUMBAUGH et al., 1997).

Neste trabalho essas três fases são consideradas, mas foram redefinidas de forma a identificar o foco principal de cada fase. As fases utilizadas são: planejamento, levantamento de requisitos, modelagem do sistema (incluindo análise e projeto do sistema) e implementação que abrange os testes unitários (realizados pelo programador) e os testes dos usuários.

a) Planejamento

O planejamento centrou-se na divisão das atividades para implementar e testar o sistema e para elaborar o texto da monografia. Uma primeira versão da modelagem do sistema e um exemplo de sua implementação foram realizados como trabalho de estágio. Como trabalho de estágio, a implementação consistiu na manutenção de um cadastro e foi realizada com o objetivo de estudar as tecnologias utilizadas, mais especificamente da linguagem de programação.

No planejamento foi decidido que inicialmente seriam revistos os requisitos e as demais representações realizadas na modelagem. Na sequência seriam implementados os cadastros e depois as demais funcionalidades. Em seguida estaria a implementação dos relatórios. E, por fim, os testes realizados tanto pelo autor deste trabalho, como os testes de

usuário realizados por funcionários da empresa (prefeitura) que serviu de base para a definição dos requisitos do sistema.

b) Revisão dos requisitos, análise e projeto

Nesta fase, os modelos gerados como trabalho de estágio foram revistos. Ajustes, especialmente na modelagem das tabelas do banco de dados foram realizados. Isso porque à medida que o sistema era implementado verificou-se que algumas funcionalidades identificadas não eram completamente atendidas pelo modelo definido.

c) Implementação do sistema

A implementação do sistema foi realizada utilizando a linguagem Delphi, com o banco de dados Firebird e o Rave Reoprts para os relatórios.

d) Testes do sistema

Os testes foram informais e realizados pelo autor deste trabalho, visando encontrar erros de implementação. Testes de usuário foram realizados com o objetivo de validar as funcionalidades do sistema. Esses testes foram realizados por colegas de trabalho da Garagem da Prefeitura do Município e Pato Branco que serviu de base para a implementação do sistema.

4 SISTEMA PARA CONTROLE DE UM PARQUE DE MÁQUINAS

Este capítulo apresenta um resumo da análise e do projeto do sistema denominado controle de um parque de máquinas e mostra como a implementação foi realizada por meio de exemplos do código gerado.

4.1 APRESENTAÇÃO DO SISTEMA

O sistema desenvolvido se destina ao gerenciamento de um parque de máquinas de uma prefeitura. Máquinas incluem todos os itens motorizados que uma prefeitura possui como, por exemplo, veículos, roçadeiras, utilitários, tratores, escavadeiras e caminhões. O sistema possibilitará controle dos gastos de cada máquina, incluindo abastecimentos, serviços de mecânica, troca ou complemento de óleo e troca ou conserto de pneus. Outro controle disponibilizado é o de serviços, como seguro obrigatório e IPVA (Imposto sobre a Propriedade de Veículos Automotores*).* Também poderão ser cadastrados usuários, materiais, tipo de materiais, veículos, tipo de veículos, motoristas, localidades, tipos de despesas e de serviços, conjunto mecânico, entre outros.

O sistema permitirá fazer consultas aos itens cadastrados, alterações ou exclusões e emitir relatórios de todas as tabelas do sistema. Com isso será possível controlar a troca de óleo do veículo (verificar se já ultrapassou a quilometragem ou horas permitida para próxima troca) e também a troca de pneus.

4.2 MODELAGEM DO SISTEMA

A Figura 7 apresenta a visão geral do sistema. Os elementos constantes nessa figura não são classes, tabelas ou outras entidades de modelagem. São apenas conceitos relacionados entre si que tem o objetivo de apresenta a ideia geral do sistema. É um modelo conceitual do sistema. Alguns desses conceitos certamente serão classes, tabelas e atributos ou campos dessas classes ou tabelas, mas outros podem simplesmente auxiliar a definir requisitos funcionais ou não funcionais ou no entendimento do que o sistema faz.

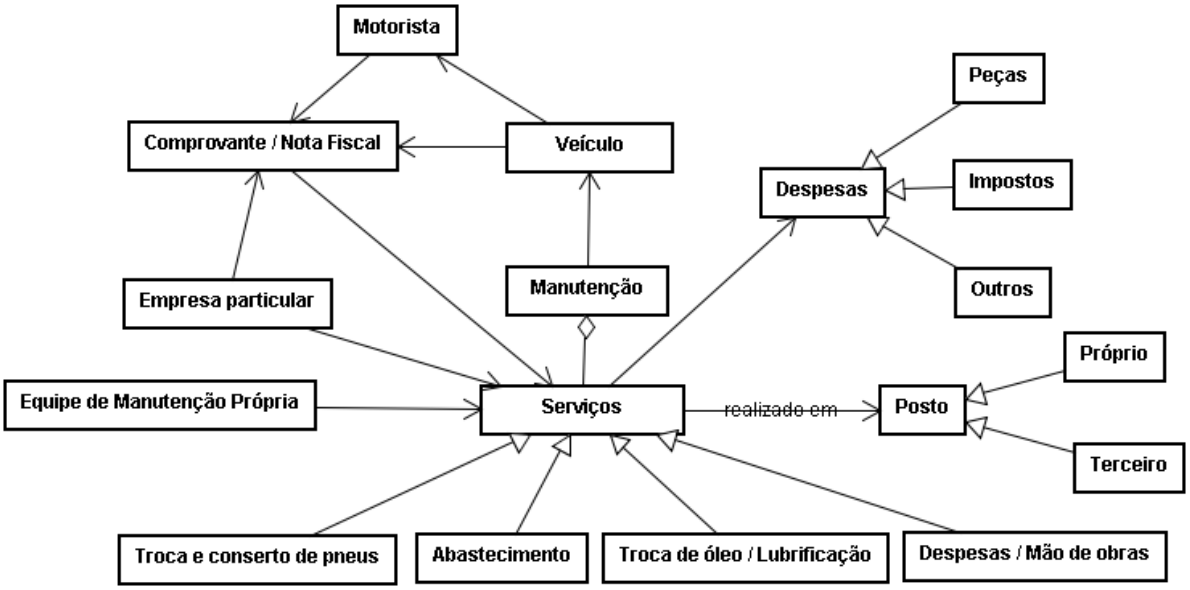

Figura 7 – Visão geral do sistema

A partir da visão geral do sistema foram definidos os requisitos, os casos de uso, as tabelas do banco de dados e outros itens de modelagem, como as classes.

Os requisitos funcionais definidos para o sistema foram:

a) Cadastro de máquinas que possuem um tipo que também é cadastrado e controle do seu estado de conservação;
b) Controle de abastecimento de cada máquina;
c) Controle de despesas de cada máquina com cadastro de tipos de despesas;
d) Controle de pagamento de seguro obrigatório e IPVA dos veículos;
e) Cadastro de motoristas;
f) Cadastro de materiais e seus tipos utilizados nos serviço realizados nas máquinas;
g) Cadastro de serviços realizados em cada máquina.

Os requisitos não funcionais definidos para o sistema foram:

a) A indicação do valor para o hodômetro atual deve ser superior à última indicação anterior;

b) Os campos obrigatórios para cada um dos cadastros.

A Figura 8 apresenta o diagrama de casos de uso.

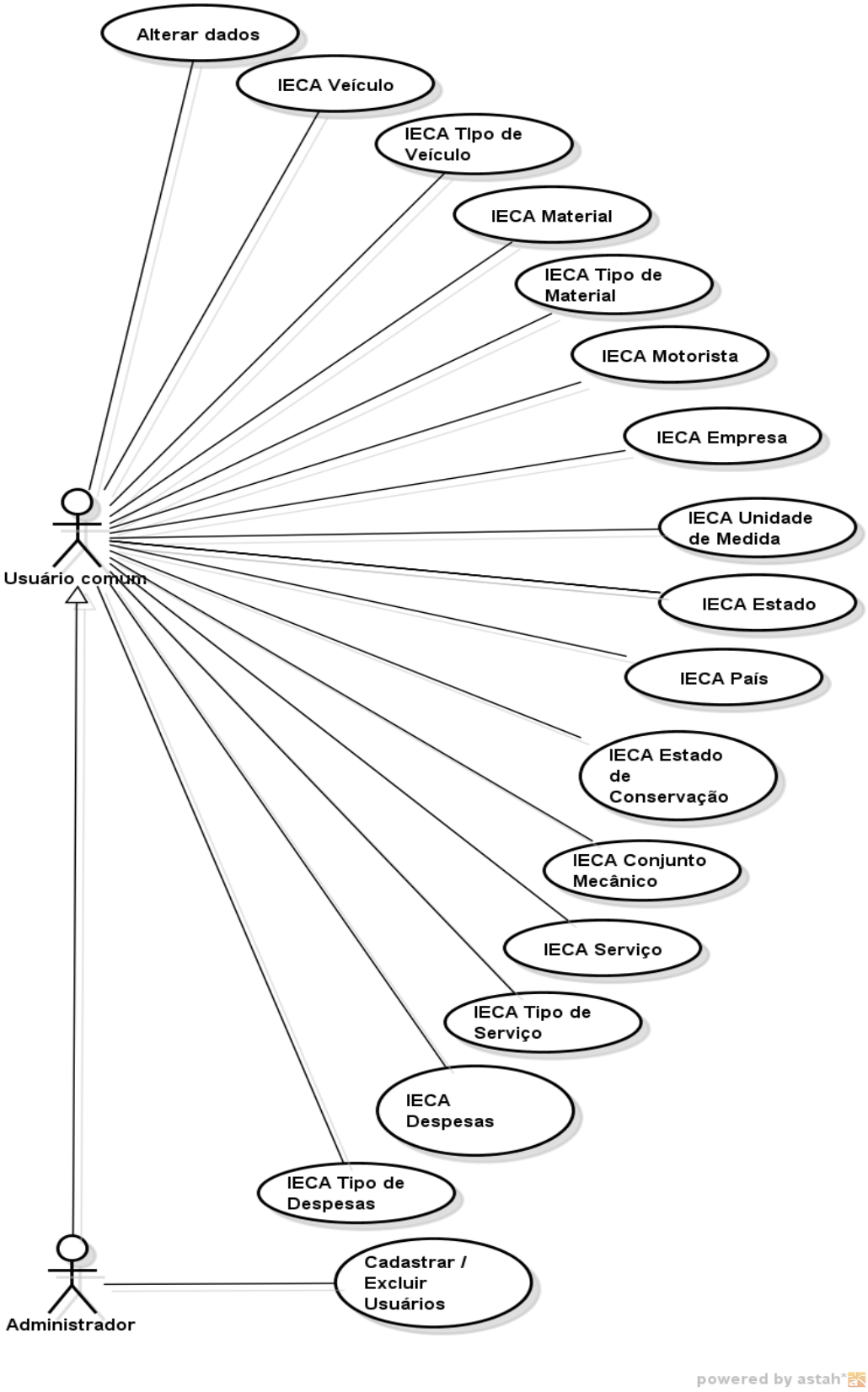

Figura 8 – Diagrama de casos de uso

Os diagramas de casos de uso da Figura 8 são descritos no Quadro 2.

Caso de uso	Descrição
Alterar dados pessoais	Alteração dos dados pessoais de cadastro no sistema.
Cadastro de usuários	Inclusão – cadastrar usuários no sistema. Exclusão – excluir usuários do sistema. Consulta – consultar os usuários cadastrados no sistema. Alteração – alterar dados de usuários cadastrados no sistema. Relatório – listagem dos usuários cadastrados no sistema.
Cadastro de veículo	Inclusão – cadastrar veículos no sistema. Exclusão – excluir veículos do sistema. Alteração – alterar dados de veículos cadastrados no sistema. Consulta – consultar veículos cadastrados no sistema. Relatório – listagem dos veículos cadastrados no sistema.
Cadastro de tipo de veículo	Inclusão – cadastrar o tipo de veículo, os tipos podem ser caminhão (diversos tipos), máquina (diversos tipos), veículos leves (diversos tipos), etc. Exclusão – excluir o tipo de veículo. Alteração – alterar dados de tipo de veículos cadastrados no sistema. Consulta – consultar tipo de veículo cadastrado no sistema. Relatório – listagem dos tipos de veículos cadastrados no sistema.
Cadastro de material	Inclusão – cadastrar material utilizado no veículo, por exemplo: gasolina, diesel, peças, etc. Exclusão – excluir material do sistema. Alteração – alterar dados de materiais cadastrados no sistema. Consulta – consultar materiais cadastrados no sistema. Relatório – listagem de materiais cadastrados no sistema.
Cadastro de tipo de Material	Inclusão – cadastrar o tipo de material que será utilizado pelos veículos, por exemplo: combustível, lubrificante, peças, etc. Exclusão – excluir tipo de material do sistema. Alteração – alterar dados de tipo de material cadastrados no sistema. Consulta – consultar os tipos de materiais desejados cadastrados no sistema. Relatório – listagem de tipos de materiais cadastrados no sistema.
Cadastro de motorista	Inclusão – cadastrar motoristas no sistema. Exclusão – excluir motorista cadastrado no sistema. Alteração – alterar dados de motorista cadastrado no sistema. Consulta – consultar motoristas cadastrados no sistema. Relatório – listagem de motoristas cadastrados no sistema.
Cadastro de empresa	Inclusão – cadastro das empresas que fornecem o material para manutenção dos veículos, por exemplo: fornecedora de combustível, empresa onde foi comprada as peças, etc. Exclusão - exclusão da empresa no sistema. Alteração – alterar dados de empresa cadastrada no sistema. Consulta – consultar as empresas cadastradas no sistema. Relatório – listagem das empresas cadastradas no sistema.
Cadastro de unidade de medida	Inclusão – cadastro de unidades de medida no sistema, por exemplo: m, ud, sc, kg, lt, etc. Exclusão - exclusão de unidade de medidas no sistema. Consulta – consultar as unidades de medida desejadas no sistema. Alteração – alterar dados de unidades de medida cadastradas no sistema. Relatório – listagem das unidades de medida cadastrados no sistema.
Cadastro de estado	Inclusão – cadastro dos estados (unidade federativa) utilizadas para cadastro de motorista. Exclusão – excluir Estados cadastrados no sistema.

	Consulta – consultar os Estados cadastrados no sistema. Alteração – alterar dados de estado cadastrado. Relatório - listagem de estados cadastrados no sistema.
Cadastro de país	Inclusão – cadastro no sistema dos países utilizados no cadastro de motoristas. Exclusão - exclusão de países do sistema. Consulta – consultar os países cadastrados no sistema. Alteração – alterar dados de países cadastrados. Relatório – listagem dos países cadastrados no sistema.
Cadastro de estado de conservação	Inclusão – cadastro no sistema dos estados de conservação dos veículos, que podem ser do tipo: péssimo, regular, bom, ótimo, novo. Exclusão – excluir estado de conservação do sistema. Consulta – consultar os estados de conservação cadastrados no sistema. Alteração – alterar dados de estados de conservação cadastrados. Relatório – listagem dos estados de conservação cadastrados no sistema.
Cadastro de conjunto mecânico	Inclusão – cadastro no sistema dos conjuntos mecânicos dos veículos ou máquinas. Por exemplo: motor, torque, transmissão, etc. Esse cadastro é importante para o lançamento da troca de óleo / lubrificação dos veículos. Exclusão - exclusão do conjunto mecânico desejado no sistema. Consulta – consultar os conjuntos mecânicos cadastrados no sistema. Alteração – alterar dados de conjuntos mecânico cadastrado no sistema. Relatório – listagem dos conjuntos mecânicos cadastrados no sistema.
Cadastro de despesas do veículo	Inclusão – cadastro no sistema das despesas do veículo. Exclusão – exclusão de alguma despesa do veículo. Consulta – consultar as despesas lançadas no sistema. Poderá ser feita de uma forma geral como também uma consulta por veículos em separado, ou seja, despesas por veículo. Alteração – alterar dados de despesas lançadas no sistema para um determinado veículo. Relatório – listagem das despesas por veículo.
Cadastro de abastecimento	Inclusão – cadastro no sistema dos abastecimentos feitos nos veículos. Exclusão – exclusão das abastecidas desejadas no sistema. Consulta – consultar as abastecidas feitas pelos veículos, poderá ser feita de um modo geral como também de forma individual por veículo. Alteração – alterar dados das abastecidas cadastradas no sistema. Relatório – listagem das abastecidas por veículo.
Cadastro de troca ou complemento de óleo	Inclusão – cadastro no sistema das trocas de óleo feitos nos veículos. Exclusão – exclusão das trocas de óleo desejadas no sistema. Consulta – consultar as trocas de óleo feitas pelos veículos, poderá ser feita de um modo geral como também de forma individual por veículo. Alteração – alterar dados das trocas de óleo desejadas no sistema. Relatório – listagem das trocas de óleo por veículo. Na tela haverá duas opções para escolher: troca ou complemento.
Cadastro de troca ou conserto de pneu	Inclusão – cadastro no sistema das trocas de pneus feitos nos veículos. Exclusão – exclusão das trocas de pneu desejadas no sistema. Consulta – consultar as trocas de pneu feitas pelos veículos, poderá ser feita de um modo geral como também de forma individual por veículo. Alteração – alterar dados das trocas de pneu. Relatório – listagem das trocas de pneu por veículo. Na tela haverá uma opção para escolher: conserto ou troca.
Cadastro de IPVA	Inclusão – cadastro no sistema das quitações de IPVA do veículo. Exclusão – exclusão de algum cadastro de IPVA no sistema.

	Consulta – consultar os cadastros das quitações de IPVA feitos pelos veículos, poderá ser feita de um modo geral como também de forma individual por veículo. Alteração – alterar dado das quitações de IPVA no sistema. Relatório – listagem das quitações de IPVA por veículo.
Cadastro de seguro obrigatório	Inclusão – cadastro no sistema das quitações de seguro obrigatório do veículo. Exclusão – exclusão de dados de cadastro na quitação do seguro obrigatório no sistema. Consulta – consultar os cadastros das quitações de seguro obrigatório feito pelos veículos, poderá ser feita de um modo geral como também de forma individual por veículo. Alteração – alterar dados das quitações de seguro obrigatório cadastrados no sistema. Relatório – listagem das quitações de seguro obrigatório por veículo.

Quadro 2 – Requisitos do sistema

A Figura 9 apresenta o diagrama de classes do sistema. Estão representadas apenas as classes, sem atributos ou métodos, porque os atributos são, com pequenas diferenças, os campos das tabelas descritos a seguir e os principais métodos são representados pela descrição dos casos de uso do Quadro 2.

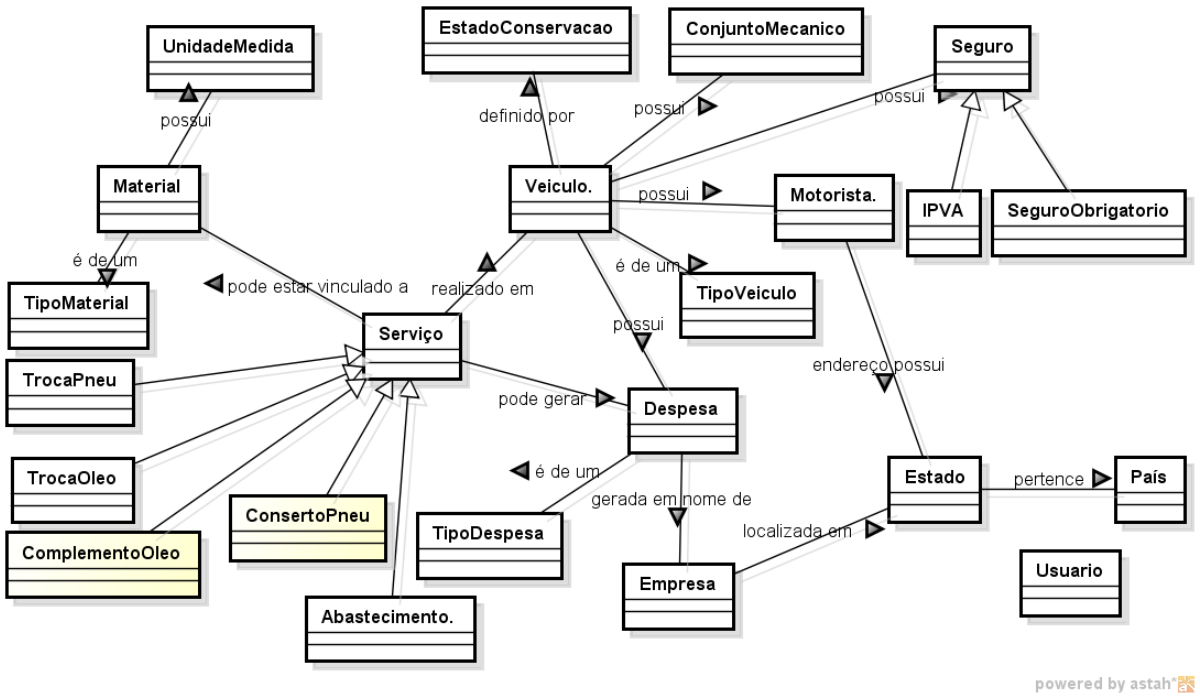

Figura 9 – Diagrama de classes

Na Figura 9 a classe usuário se refere aos usuários do sistema e não está vinculada a outras classes. Ela somente valida o acesso do usuário ao sistema. As tabelas definidas para o

banco de dados estão listadas a seguir e se referem as classes da Figura 11 porque essas representam unidades de persistência. Elas representam as classes persistentes do sistema. Os tipos de dados definidos para as tabelas são tipos não vinculados ao banco de dados. A modelagem do banco foi realizada independentemente da tecnologia a ser utilizada para a implementação do mesmo.

Os tipos de dados definidos para as tabelas são tipos não vinculados ao banco de dados. A modelagem do banco foi realizada independentemente da tecnologia a ser utilizada para a implementação do mesmo.

ABASTECIMENTO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodAbastecimento	AutoNumeração	Não	Sim	Não
DataAbastec	Data	Não	Não	Não
CodEmpresa	Número	Não	Não	Sim
ValorAbastec	Moeda	Não	Não	Não
CodMaterial	Número	Não	Não	Sim
Hora	Hora	Sim	Não	Não
Quantidade	Número	Não	Não	Não
CodVeículo	Número	Não	Não	Sim
Hodômetro	Número	Não	Não	Não

COMPLEMENTO DE ÓLEO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodCompOleo	AutoNumeração	Não	Sim	Não
DataComp	Texto	Não	Não	Não
CodEmpresa	Número	Não	Não	Sim
ValorComp	Moeda	Não	Não	Não
HoraComp	Hora	Não	Não	Não
CodConjMecanico	Número	Não	Não	Sim
CodMaterial	Número	Não	Não	Sim
QuantMat	Número	Não	Não	Não
CodVeiculo	Número	Não	Não	Sim
HodometroVeic	Número	Não	Não	Não

CONJUNTO MECÂNICO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodConjMecanico	AutoNumeração	Não	Sim	Não
DescConjMecanico	Texto	Não	Não	Não

CONSERTO DE PNEU

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodConPneu	AutoNumeração	Não	Sim	Não
DataPneu	Data	Não	Não	Não
CodEmpresa	Número	Não	Não	Sim

ValorPneu	Moeda	Não	Não	Não
HoraPneu	Hora	Não	Não	Não
CodMaterial	Número	Não	Não	Sim
QuantPneu	Número	Não	Não	Não
CodVeículo	Texto	Não	Não	Sim
HodomVeic	Número	Não	Não	Não
Obs	Texto	Sim	Não	Não

DESPESAS

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodDespesa	AutoNumeração	Não	Sim	Não
DataDesp	Texto	Não	Não	Não
ValorDesp	Moeda	Não	Não	Não
CodEmpresa	Número	Não	Não	Sim
Observações	Texto	Não	Não	Não
CodVeículo	Número	Não	Não	Sim
NumNotaF	Número	Sim	Não	Não

EMPRESA

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodEmpresa	AutoNumeração	Não	Sim	Não
NomeEmpresa	Texto	Não	Não	Não
Endereço	Texto	Não	Não	Não
Cidade	Texto	Não	Não	Não
CodEstado	Número	Não	Não	Sim
CodPaís	Texto	Não	Não	Sim
TelEmpresa	Texto	Não	Não	Não

ESTADO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodEstado	AutoNumeração	Não	Sim	Não
NomeEstado	Texto	Não	Não	Não

ESTADO DE CONSERVAÇÃO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodEstConservacao	AutoNumeração	Não	Sim	Não
DescEstConservacao	Texto	Não	Não	Não

IPVA

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodIPVA	AutoNumeração	Não	Sim	Não
CodVeículo	Número	Não	Não	Sim
ValorIPVA	Moeda	Não	Não	Não
DataVencIPVA	Data	Não	Não	Não
DataPagtoIPVA	Data	Não	Não	Não
ObserIPVA	Texto	Sim	Não	Não

MATERIAL

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodMaterial	AutoNumeração	Não	Sim	Não
DescricaoMaterial	Texto	Não	Não	Não
CodTipoMaterial	Número	Não	Não	Sim

CodUdMedida	Número	Não	Não	Sim

MOTORISTA

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodMotorista	AutoNumeração	Não	Sim	Não
NomeMotorista	Texto	Não	Não	Não
DataNasc	Data	Não	Não	Não
NumCNH	Número	Não	Não	Não
Categoria	Texto	Não	Não	Não
CidMotorista	Texto	Não	Não	Não
CodEstado	Número	Não	Não	Sim
CodPaís	Número	Não	Não	Sim
Telefone	Texto	Não	Não	Não

TROCA DE ÓLEO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodTrocaOleo	AutoNumeração	Não	Sim	Não
DataTroca	Data	Não	Não	Não
CodEmpresa	Número	Não	Não	Sim
ValorTroca	Moeda	Não	Não	Não
HoraTroca	Hora	Não	Não	Não
CodConjMecanico	Número	Não	Não	Sim
CodMaterial	Número	Não	Não	Sim
QuantMat	Número	Não	Não	Não
CodVeículo	Número	Não	Não	Sim
HodômetroVeic	Número	Não	Não	Não
HodômetroProxTroca	Número	Sim	Não	Não

PAÍS

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodPais	AutoNumeração	Não	Sim	Não
NomePais	Texto	Não	Não	Não

TROCA DE PNEU

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodPneu	AutoNumeração	Não	Sim	Não
DataPneu	Data	Não	Não	Não
CodEmpresa	Número	Não	Não	Sim
ValorPneu	Moeda	Não	Não	Não
HoraPneu	Hora	Não	Não	Não
CodMaterial	Número	Não	Não	Sim
QuantPneu	Número	Não	Não	Não
CodVeículo	Texto	Não	Não	Sim
HodomVeic	Número	Não	Não	Não
HodomProxTrocaPn	Número	Sim	Não	Não

SEGURO OBRIGATÓRIO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodSegObrigatorio	AutoNumeração	Não	Sim	Não
CodVeículo	Número	Não	Não	Sim
ValorSeg	Moeda	Não	Não	Não

DataVencSeg	Data	Não	Não	Não
DataPgtoSeg	Data	Não	Não	Não
ObserSeg	Texto	Sim	Não	Não

TIPO MATERIAL

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodTipoMaterial	AutoNumeração	Não	Sim	Não
TipoMaterial	Texto	Não	Não	Não

TIPO VEÍCULO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodTipoVeiculo	AutoNumeração	Não	Sim	Não
DescricaoTipoVeiculo	Texto	Não	Não	Não

UNIDADE MEDIDA

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodUdMedida	AutoNumeração	Não	Sim	Não
DescricaoUdMedida	Texto	Não	Não	Não
Sigla	Texto	Não	Não	Não

USUÁRIO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodUsuario	AutoNumeração	Não	Sim	Não
NomeUsuario	Texto	Não	Não	Não
Senha	Texto	Não	Não	Não

VEÍCULO

Campo	Tipo	Nulo	Chave primária	Chave estrangeira
CodVeiculo	AutoNumeração	Não	Sim	Não
DescVeiculo	Texto	Não	Não	Não
CodEstConservacao	Número	Não	Não	Sim
CodTipoVeiculo	Número	Não	Não	Sim
CodMaterial	Número	Não	Não	Sim
CodMotorista	Número	Não	Não	Sim
PlacaVeiculo	Texto	Não	Não	Não
Chassi	Texto	Não	Não	Não
Renavam	Texto	Não	Não	Não
AnoFab	Número	Não	Não	Não
AnoModelo	Número	Não	Não	Não
UdHod	Número	Não	Não	Não

4.3 DESCRIÇÃO DO SISTEMA

A tela principal do sistema de gerenciamento de um parque de máquinas contém o menu com as opções: Cadastros, Movimentos, Consultas e Ajuda. Esses menus fornecem o acesso às funcionalidades do sistema.

Por meio da opção “Cadastros” no menu da tela principal (Figura 10) é possível fazer cadastros de: Usuários, Localidade (Estado e País), Motorista, Veículos (Veículo, Conjunto Mecânico, Tipo de Veículo, Estado de Conservação), Unidade Medida, Materiais (Material, Tipo de Material) e Fornecedor.

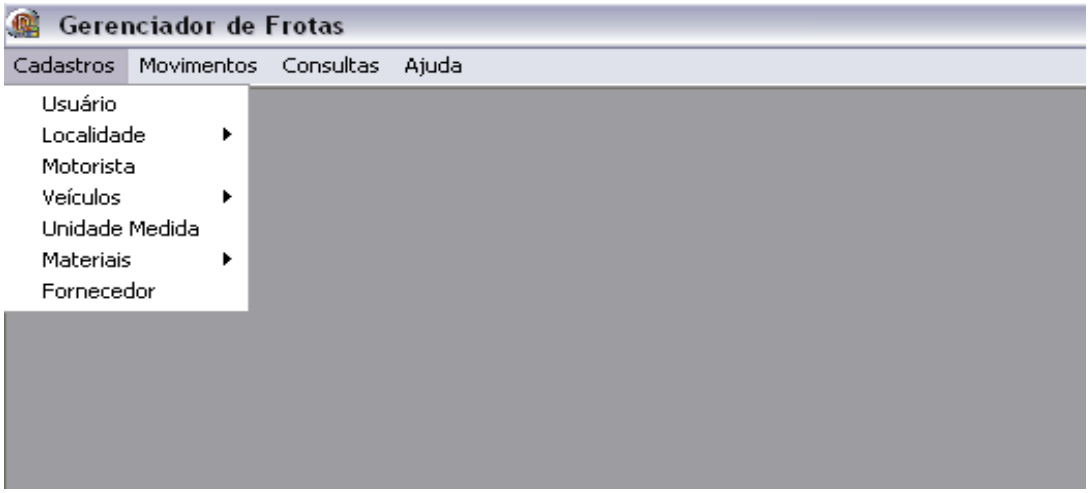

Figura 10 – Opções do menu

A seguir está exemplificada a forma de acesso e de manipulação do sistema, além de mostrar o padrão de interface utilizado. A apresentação dessas funcionalidades está centrada em três cadastros, que são de países (denominado IECA País, ou seja, inclusão, exclusão, consulta e alteração), Unidades de Federação (IECA Estado) e motoristas (IECA Motorista).

Para acessar a tela de manutenção de Estado, basta acessar o menu Cadastros, opção Localidade e em seguida Estado (Figura 11).

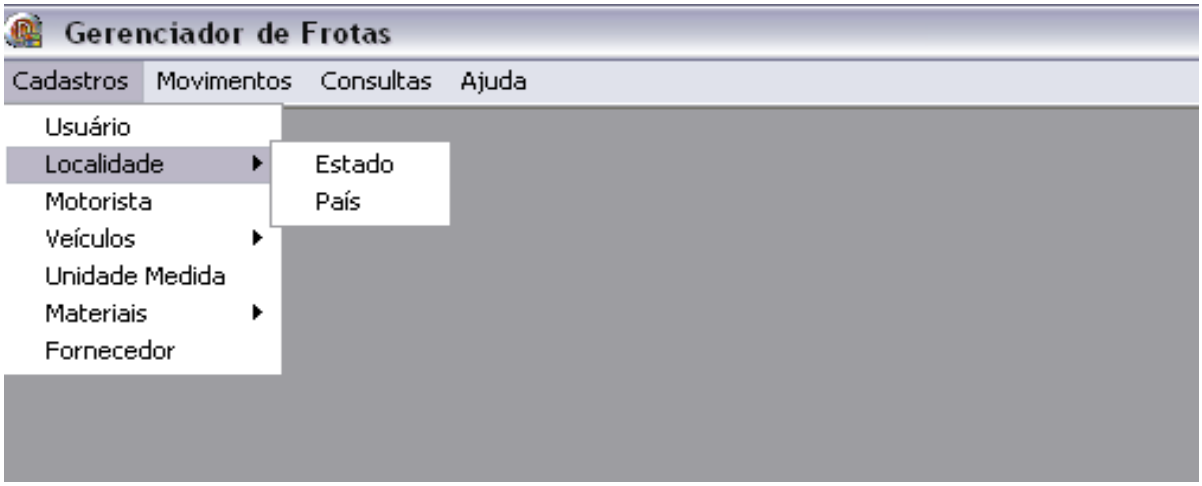

Figura 11 – Menu de acesso a manutenção de Estado

Acessando a tela de manutenção de Estados será apresentada a tela da Figura 12. Essa tela de cadastro possui as opções código e nome do Estado nos dados cadastrais e os

comandos: Incluir, Alterar, Salvar, Excluir, Cancelar e Fechar. E a opção para consulta de dados, apresentada na listagem na parte inferior da tela.

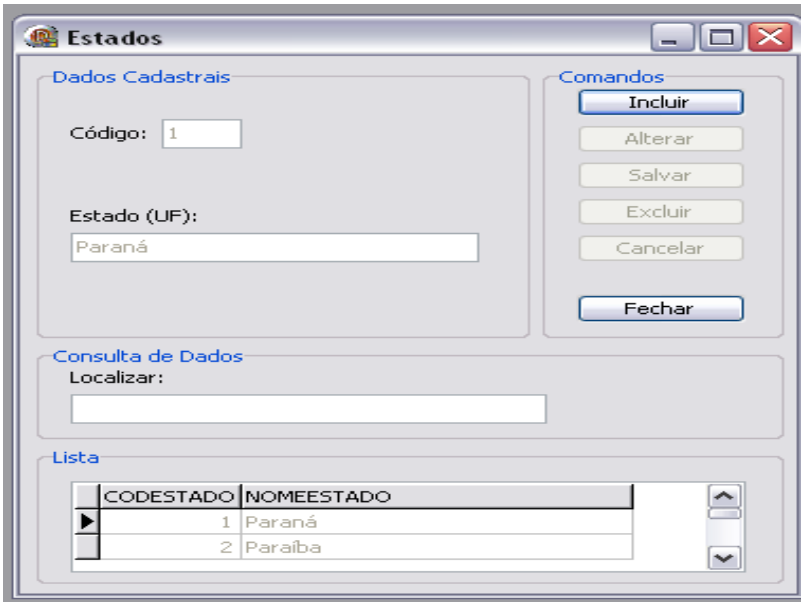

Figura 12 - Tela de manutenção de Estados

A tela de manutenção de países é semelhante à tela de Estado (Figura 13). A opção de consulta de dados permite pesquisar se um determinado país já está cadastrado. É uma forma mais rápida de pesquisa do que percorrer a lista apresentada na parte inferior da tela.

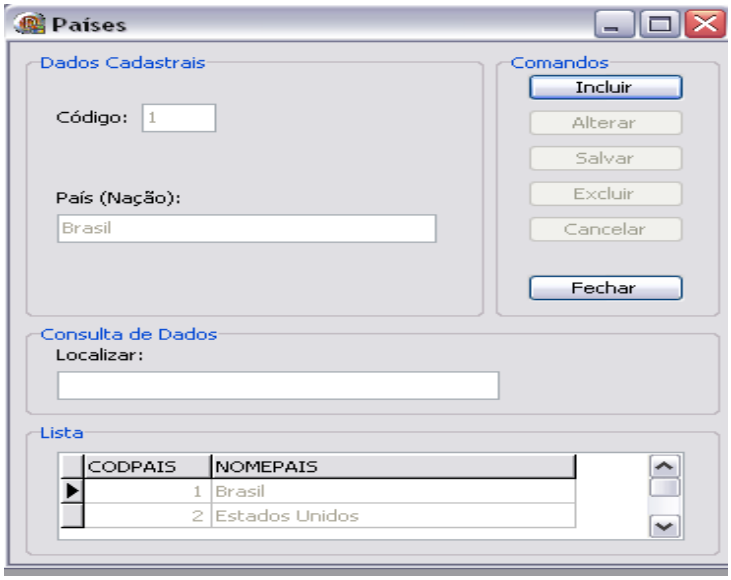

Figura 13 – Tela de manutenção de país

Considerando que a forma de manutenção de cadastro de Países e Estados é semelhante a seguir é apresentada a forma de inclusão, exclusão, consulta e alteração do cadastro de Estados.

Quando a tela é acessada estão habilitados apenas os botões Incluir e Fechar e a listagem na parte inferior da tela que contém os dados armazenados na respectiva tabela no banco de dados. As demais opções representadas pelos respectivos botões ficam desabilitadas. (Figura 14).

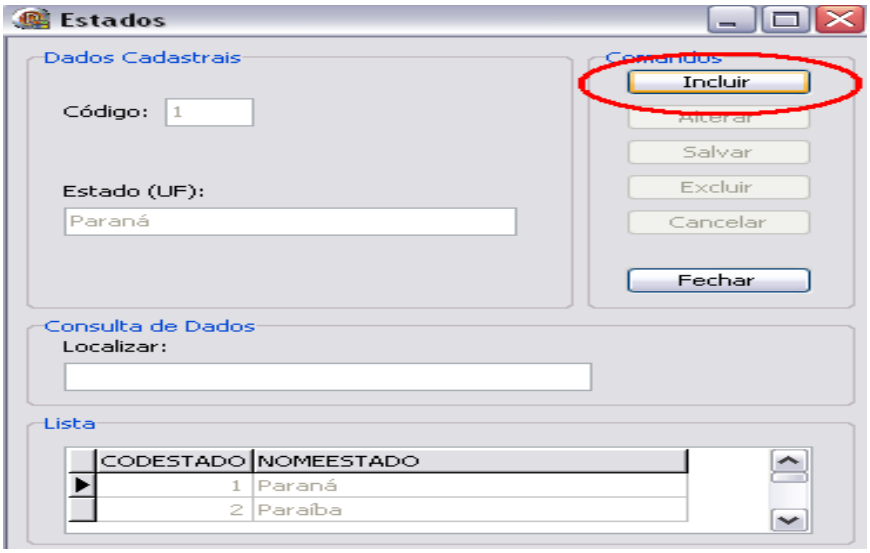

Figura 14 – Tela de manutenção de Estado - incluir

Após isso, os componentes incluídos no *GroupBox* Dados Cadastrais são habilitados juntamente com os botões Salvar e Cancelar do *GroupBox* Comandos. O botão Fechar continua habilitado, enquanto o botão Incluir agora fica desabilitado (Figura 15).

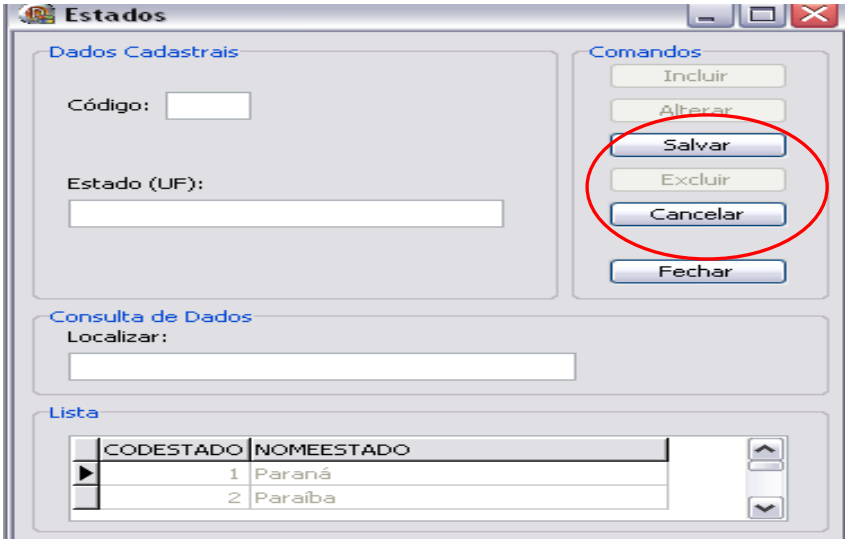

Figura 15 – Tela de manutenção de Estado – salvar e cancelar

A Figura 16 ilustra o preenchimento dos dados cadastrais que serão salvos por meio do botão Salvar ou desconsiderados por meio do botão Cancelar.

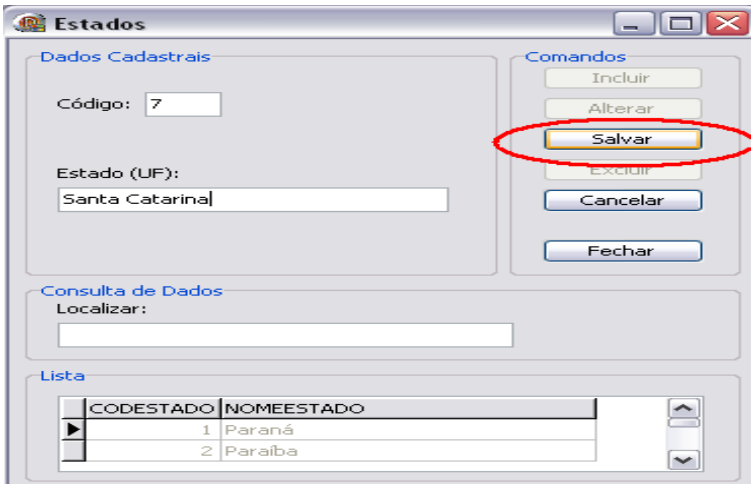

Figura 16 – Tela de manutenção de Estado - salvar

Quando um novo Estado é cadastrado no sistema, os botões de Incluir, Alterar e Excluir são habilitados, o botão Fechar continua habilitado, enquanto os demais ficam desabilitados. A Figura 17 mostra o estado incluído "Santa Catarina" na listagem apresentada na parte inferior da tela.

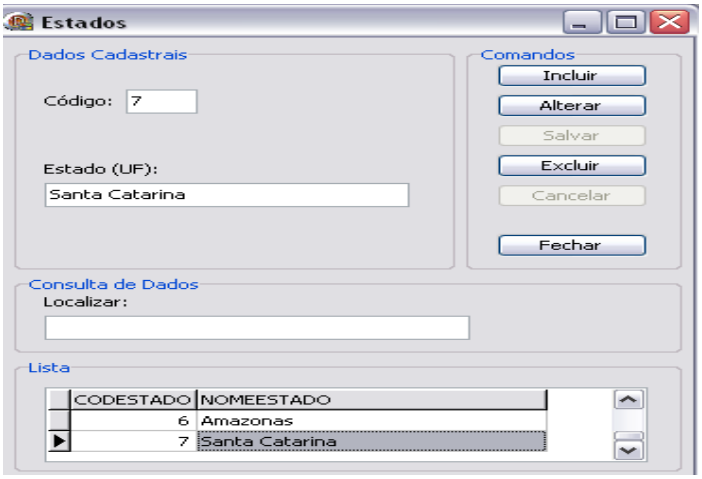

Figura 17 – Tela de manutenção de Estado – Estado incluído

Para excluir um lançamento é necessário escolher no *GroupBox* Lista um dos lançamentos desejados. Automaticamente esse lançamento é carregado no *GroupBox* Dados Cadastrais. Em seguida, clicando no botão Excluir o lançamento desejado será excluído da base de dados do sistema.

Para cadastrar um motorista é necessário selecionar a respectiva opção no menu cadastro. Após isso, a tela de manutenção de motoristas será acessada (Figura 18). Os comandos de habilitar e desabilitar os botões e os dados cadastrais funcionam de maneira semelhante à tela de manutenção de Estado e País.

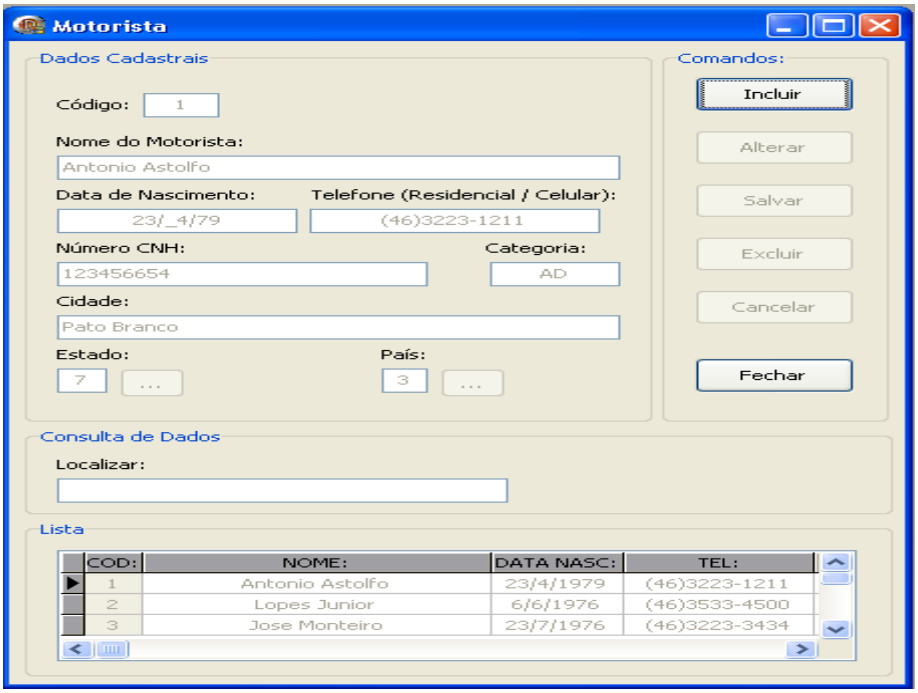

Figura 18 – Tela de manutenção de motorista

A tela de manutenção Motorista possui dois botões ao lado dos campos de preenchimento de Estado e País. Em cada botão serão carregados os dados cadastrados de Estado (Figura 19) e de País.

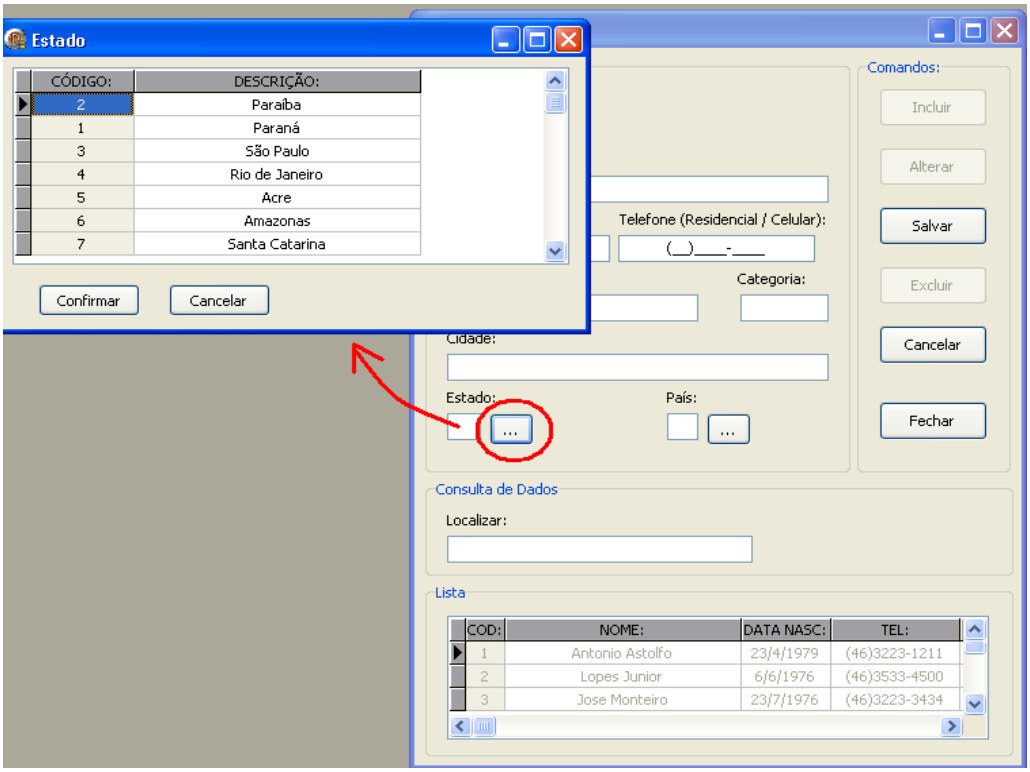

Figura 19 – Tela de manutenção de motorista com o botão de Estado acionado

A seguir está exemplificada a forma de acesso e de consulta dos relatórios de cadastro de motoristas e abastecimentos dos veículos. A tela de consultas é acessada pelo menu Consultas (Figura 20). As consultas estão divididas em Tabelas (com listagens das tabelas do banco de dados) e Movimentações (com composição de tabelas). Para exemplificar as consultas será apresentada a consulta de Motoristas (Figura 20). Ao selecionar essa consulta será aberta uma tela com um *grid* carregado com os dados cadastrados no sistema. Para visualizar o relatório é necessário clicar no botão Gerar Relatório (área circulada na Figura 20).

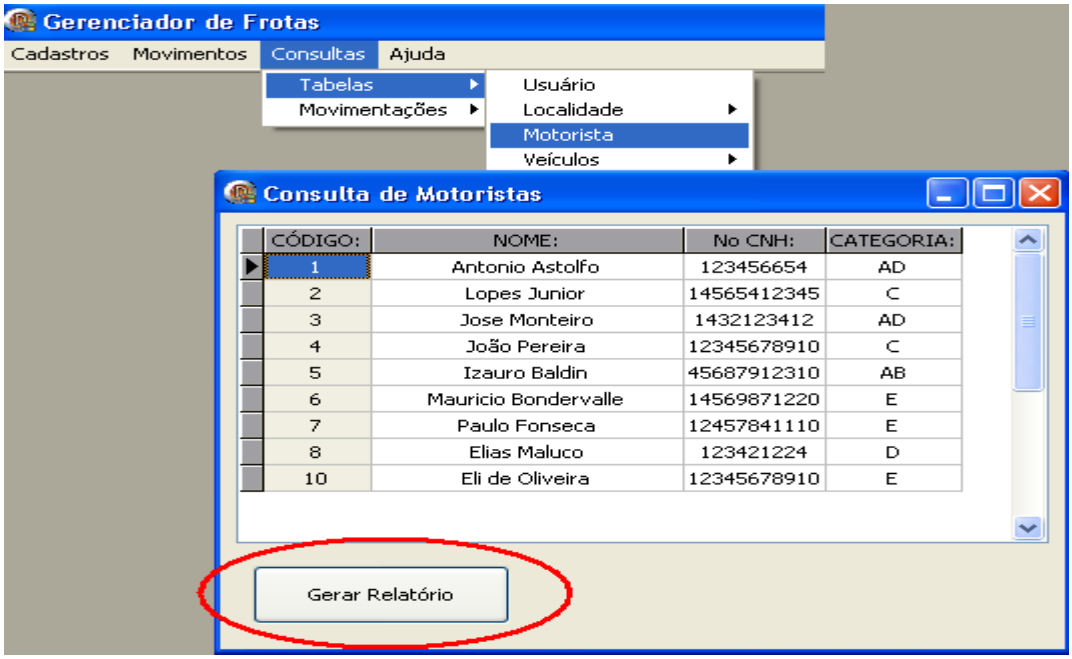

Figura 20 – Tela de consulta de motoristas

Clicando no botão Gerar Relatório, o relatório de motoristas é gerado. No cabeçalho aparecerá o logotipo do sistema à esquerda e o título do relatório juntamente com a data de emissão do mesmo à direita (Figura 21).

Figura 21 – Relatório de motoristas

O relatório da opção Movimentações, Abastecimentos, Geral apresenta todos os abastecimentos cadastrados no sistema. Ao clicar o botão Gerar Relatório será gerado o respectivo relatório (Figura 22).

Figura 22 – Relatório de abastecimentos

4.4 IMPLEMENTAÇÃO DO SISTEMA

Para exemplificar o código foi escolhido o cadastro de Motorista, pois este possui mais opções, mais dados a serem cadastrados, ou seja, é mais relevante em termos de implementação (código) do que IECA Estado e IECA País.

O código do botão "Incluir" é apresentado na Listagem 1. A sua implementação foi realizada no evento OnClick do botão "Incluir". As variáveis "RegSel" (do tipo *integer* armazena a o registro selecionado) foram inicializadas com zero e "Alterando" (do tipo *boolean*) com falso porque esse botão tem o objetivo de implementar o inserção de um novo registro no banco de dados. As propriedades dos botões da tela de manutenção Motorista são marcadas para ativo ou inativo, apenas para organizar melhor o leiaute da tela e seus funcionamentos. Após isso é chamado o procedimento "Limpar_Campos", que ao ser clicado no botão incluir limpa todos os campos, ou seja, estarão livres para serem preenchidos pelo usuário.

```
procedure TfrmMotorista.btIncluirClick(Sender: TObject);
begin
  RegSel := 0;
    Alterando := false;

    AtivaCampos(true);

    btIncluir.Enabled := false;
    btAlterar.Enabled := false;
    btExcluir.Enabled := false;
    btSalvar.Enabled := true;
    btCancelar.Enabled := true;
    DBGridMotorista.Enabled := false;

  edtCodigo.SetFocus;

  Limpar_Campos;
end;
```

Listagem 1 – Implementação botão Incluir – manutenção de motorista

A Listagem 2 apresenta a implementação do botão "Alterar". Quando o botão "Alterar" é acionado, a variável Alterando ficará como "*true*", indicando que será feita uma alteração. E a variável RegSel (que armazena o registro selecionado) será inicializada com o valor do campo código. Em seguida será chamado o procedimento AtivaCampos que ativa todos os campos para serem alterados pelo usuário. Ao mesmo tempo as propriedades dos botões são modificadas novamente para ativo ou inativo, apenas para organizar o leiaute da tela.

```
procedure TfrmMotorista.btAlterarClick(Sender: TObject);
begin
  RegSel := StrToInt(edtCodigo.Text);
  Alterando := true;

  AtivaCampos(true);

  btIncluir.Enabled := false;
  btAlterar.Enabled := false;
  btExcluir.Enabled := false;
  btSalvar.Enabled := true;
  btCancelar.Enabled := true;
  DBGridMotorista.Enabled := false;
end;
```

Listagem 2 – Implementação botão Alterar – manutenção de motorista

O botão “Salvar”, cujo código é apresentado na Listagem 3, é o mais importante dessa tela, pois é por meio dele que será possível inserir um novo registro no banco de dados. Foi declarada uma variável do tipo string: cSql. A variável cSql vai receber os comandos SQL para inserção e atualização de registros no banco de dados. Uma condição foi estabelecida para verificar se os campos código e nome do motorista foram preenchidos, caso eles estejam em branco será mostrada uma mensagem de alerta: “Por favor, preencha todos os campos”, ou seja, esses campos não podem ser deixados em branco.

```
procedure TfrmMotorista.btSalvarClick(Sender: TObject);
var
  cSql,       :: String;
begin
  try
      if (edtCodigo.Text = '') or (edtMotorista.Text = '') then
        begin
          MessageDlg('Por favor, preencha todos os campos.', mtInformation, [mbOk], 0);
          edtCodigo.SetFocus;
          Exit;
        end;

        if Alterando = False then
        begin
          cSql := 'Insert into motorista(codmotorista, nomemotorista, datanasc,'+
           'telefone, numcnh, categoria, cidmotorista, codestado, codpais)'+
            'values (:_cod, :_nome, :_data, :_telefone, :_numcnh, :_categoria, :_cidade, :_est, :_pais)';

                IbQryMot.SQL.Clear;
                IbQryMot.SQL.Add(cSql);
                IbQryMot.ParamByName('_cod').AsInteger := StrToInt(edtCodigo.Text);
                IbQryMot.ParamByName('_nome').AsString := edtMotorista.Text;
                IbQryMot.ParamByName('_data').AsString := edtData.Text;
                IbQryMot.ParamByName('_telefone').AsString := edtTelefone.Text;
                IbQryMot.ParamByName('_numcnh').AsString := edtNumCNH.Text;
                IbQryMot.ParamByName('_categoria').AsString := edtCategoria.Text;
                IbQryMot.ParamByName('_cidade').AsString := edtCidade.Text;
                IbQryMot.ParamByName('_est').AsInteger  := cbEstado.Items.Add(cbEstado.text);
                IbQryMot.ParamByName('_pais').AsInteger := cbPais.Items.Add(cbPais.text);

        end
```

Listagem 3 – Implementação botão Salvar – manutenção motorista (parte 1)

Para inclusão são executados os principais comandos desse botão, que se refere à inserção dos registros no banco de dados. Em seguida, por meio de uma condição, será verificado se “Alterando” é igual a “falso”: se o usuário está alterando um registro existente no banco ou inserindo um novo registro no banco. Caso essa condição seja verdadeira, o sistema entenderá que está sendo inserido um registro e serão executados os comandos de inserção no banco. Primeiro, o comando será atribuído a variável cSql. A TIBQuery do cadastro de motoristas recebeu o nome de IbQryMot, nela estará a database, o código SQL que será enviado para o servidor e a qual transação ela está relacionada. Após a variável cSql receber o código SQL, serão feitas algumas operações com a TIBQuery. Primeiro será adicionado na *query* o código SQL que será enviado para o servidor pela variável cSql. Em

seguida é passado o valor de cada parâmetro (palavras iniciadas com ":", ex: ":_cod") para a *query* através da propriedade "ParamByName".

Ainda no botão Salvar (Figura 4), caso "Alterando" seja verdadeiro, o sistema entenderá que naquele momento estará sendo feita uma alteração em algum registro existente e em seguida executará os mesmos comandos. A variável cSql vai receber o código SQL, em seguida a variável cSql será adicionada na *query* para que o código SQL contido em cSql possa ser enviado para o servidor. Em seguida são passados os valores dos parâmetros para a *query* por meio da propriedade "ParamByName".

Após isso o banco de dados será preparado reservando recursos para otimização da *query* pelo comando: IbQryMot.Prepare. Por fim, as instruções da propriedade SQL serão executadas com o comando: IBQryMot.ExecSQL e as transações são realizadas.

```
      else
      begin
        cSql := 'update motorista set nomemotorista = :_nome, datanasc = :_data,' +
              ' telefone = :_telefone, numcnh = :_numcnh,' +
              'categoria = :_categoria, cidade = :_cidade, codestado = :_est, codpais = :_pais' +
              'where codmotorista = :_cod';

              IbQryMot.SQL.Clear;
              IbQryMot.SQL.Add(cSql);
              IbQryMot.ParamByName('_cod').AsInteger := StrToInt(edtCodigo.Text);
              IbQryMot.ParamByName('_nome').AsString := edtMotorista.Text;
              IbQryMot.ParamByName('_data').AsString := edtData.Text;
              IbQryMot.ParamByName('_telefone').AsString := edtTelefone.Text;
              IbQryMot.ParamByName('_numcnh').AsString := edtNumCNH.Text;
              IbQryMot.ParamByName('_categoria').AsString := edtCategoria.Text;
              IbQryMot.ParamByName('_cidade').AsString := edtCidade.Text;
              IbQryMot.ParamByName('_est').AsInteger  := cbEstado.Items.Add(cbEstado.Text);
              IbQryMot.ParamByName('_pais').AsInteger := cbPais.Items.Add(cbPais.text);
      end;


      IbQryMot.Prepare;
      IbQryMot.ExecSQL;
      Dm.IBTrans.Commit;
  except
    on e: exception do
    begin
      MessageDlg('Erro ao gravar o registro! '+e.Message, mtError, [mbOk], 0);
      Dm.IBTrans.Rollback;
    end;
end;
```

Listagem 4 – Implementação botão Salvar – manutenção motorista (parte 2)

Na implementação do botão Excluir (Listagem 5) são declaradas duas variáveis do tipo string: cSql e RetExec. O processo de salvamento inicia com uma condição, caso a opção escolhida para a MessageBox que aparecerá na tela depois que clicar no botão Excluir for afirmativa serão executados alguns comandos. Na variável cSql será atribuído o código SQL

para exclusão dos registros do banco de dados e o campo código da tela de cadastros de motoristas. Em seguida, a variável RetExec receberá o valor de ExecQry, que por sua vez possuirá a propriedade SQL contida em cSql: RetExec := ExecQry(cSql, ‘E’). Por fim todos os dados são carregados novamente na tela de cadastros de motoristas por meio do procedimento Carregar_Dados.

```
procedure TfrmMotorista.btExcluirClick(Sender: TObject);
var
  cSql, RetExec: String;
begin
  if Application.MessageBox('Deseja Excluir o Registro Selecionado?',
    'Confirmação', MB_YESNO+MB_ICONQUESTION+MB_DEFBUTTON2)= idYes then

    cSql := 'delete from motorista where codmotorista = ' +QuotedStr(edtCodigo.Text);
    RetExec := ExecQry(cSql, 'E');
    if RetExec <> 'Ok' then
      MessageDlg(RetExec, mtError, [mbOk], 0);

      Carregar_Dados;

end;
```

Listagem 5 – Implementação botão Excluir – manutenção motorista

Na implementação do botão Cancelar (código na Listagem 6) serão executadas apenas algumas mudanças nas propriedades dos botões e dos campos *text* do formulário. Alguns botões ficarão ativos e outros inativos. Os dados inseridos no banco de dados serão carregados na tela de cadastro por meio do procedimento Carregar_Dados, porém todos os campos ficarão inativos pelo procedimento AtivaCampos que receberá “falso” como parâmetro.

```
procedure TfrmMotorista.btCancelarClick(Sender: TObject);
begin
  btIncluir.Enabled := true;
  btAlterar.Enabled := true;
  btExcluir.Enabled := true;
  btSalvar.Enabled := false;
  btCancelar.Enabled := false;
  DBGridMotorista.Enabled := true;

  Carregar_Dados;

  AtivaCampos(false);
end;
```

Listagem 6 – Implementação botão Cancelar – manutenção motorista

Na Listagem 7 está sendo mostrado como foi implementado o procedimento Carregar_Dados. Primeiro foi declarado a variável cSql do tipo String. Essa variável receberá a instrução SQL que posteriormente será enviada ao servidor. Em seguida uma condição será avaliada. Caso a *query* esteja ativa ela executará a instrução SQL contida em cSql e localizará

o registro contigo na variável RegSel pelo comando “Locate”, senão haverá uma mensagem de erro e nada será executado.

```
procedure TfrmMotorista.Carregar_Dados;
var cSql : String;
begin
  try
    cSql := 'select codmotorista, nomemotorista, datanasc, numcnh, categoria,'+
    'cidmotorista, estado. codestado, pais.codpais, estado.nomeestado, pais.nomepais,'+
    'telefone  from motorista inner join estado on (estado.codestado = motorista.codestado)'+
    'inner join pais on (pais.codpais = motorista.codpais) order by codmotorista';

    if IbQryMot.Active then
      IbQryMot.Close;
      IbQryMot.SQL.Clear;
      IbQryMot.SQL.Add(cSql);
      IbQryMot.Open;

      IbQryMot.Locate('CODMOTORISTA', RegSel, [loPartialKey, loCaseInsensitive]);
    except
      on e: exception do
        MessageDlg('Erro! '+e.Message, mtError, [mbOK], 0);
  end;
end;
```

Listagem 7 – Implementação procedimento Carregar_Dados

Na listagem 8 está a implementação do botão que é utilizado para acessar a tabela de veículos cadastrados no sistema no qual o usuário pode escolher o veículo desejado. Lembrando que a implementação para os outros botões com essa funcionalidade é igual.

Primeiro foram declaradas quatro variáveis do tipo String. Em seguida as variáveis receberam valores da tabela de veículos, por exemplo, a variável cCodVeic está recebendo o valor código da tabela de veículos: “veiculo.codveiculo” e assim por diante. Por fim a variável cSql irá receber um comando SQL, que será o select para selecionar os campos desejados da tabela de veículos, no caso: Código, Descrição e Hodômetro.

Na próxima linha está o comando para criação do Form que possibilita a visualização do *grid* com os dados selecionados da tabela Veículos. Em seguida, por meio do condicional *if*, serão passados os valores armazenados nas variáveis cCodVeic, cNomeVeic e cHodVeic para os campos TEdit e TLabel do *form* de Abastecimentos. O campo TEdit chamado edtVeic receberá pela sua propriedade *text* o valor da variável cCodVeic. O valor da variável cNomeVeic será carregada no *label* chamado lblVeic pela propriedade Caption. O mesmo acontecerá com o valor da variável cHodVeic.

```
190 procedure TfrmAbastecimento.btLocVeicClick(Sender: TObject);
    var
      cSql, cCodVeic, cNomeVeic, cHodVeic: String;
    begin
      cCodVeic := 'veiculo.codveiculo';
      cNomeVeic := 'veiculo.descveiculo';
      cHodVeic := 'veiculo.udhod';
      cSql := 'select '+cCodVeic+','+cNomeVeic+','+cHodVeic+' from veiculo';
      FrmVeicSel := TFrmVeicSel.Create(Self);

200   if FrmVeicSel.Retorno(cSql, cCodVeic, cNomeVeic, cHodVeic) <> '' then
        begin
          edtVeic.Text := cCodVeic;
          lblVeic.Caption := cNomeVeic;
          lblHodAtual.Caption := cHodVeic;
        end;
```

Listagem 8 – Implementação procedimento Acessar tabela de veículos

Na listagem 9 está sendo mostrado uma condicional criada no evento OnExit do campo TEdit chamado edtHodom. Essa condicional mostra que caso o hodômetro passado nesse campo (edtHodom) seja menor ou igual ao hodômetro atual do veículo (o hodômetro atual será mostrado por meio de um *label* chamado lblHodAtual) uma mensagem na tela informando 'O valor do hodômetro passado é menor do que o cadastrado.'.

```
  procedure TfrmAbastecimento.edtHodomExit(Sender: TObject);
  begin
358 if edtHodom.Text <= lblHodAtual.Caption then
      begin
360     MessageDlg('O valor do hodômetro passado é menor do que o cadastrado.', mtInformation, [mbOk], 0);
        edtHodom.SetFocus;
        Exit;
      end;

  end;
```

Listagem 9 – Implementação do procedimento OnExit do campo TEdit do hodômetro

5 CONCLUSÃO

Este trabalho teve como objetivo principal apresentar a implementação de um sistema para controle ou gerenciamento de um parque de máquinas de uma prefeitura e exemplificar o uso da linguagem Delphi e do banco de dados Firebird. Os objetivos foram alcançados. O sistema foi implementado conforme planejado e a exemplificação de uso da linguagem foi realizada. Considerando que a linguagem Delphi é orientada a objetos, o referencial teórico do trabalho centrou-se em orientação a objetos, apresentado uma visão simplificada da modelagem de um sistema utilizando a UML.

A essência da modelagem e da implementação de sistemas orientados a objetos é identificar as entidades básicas ou essenciais de um sistema, os objetos, definir os atributos e as ações (operações implementadas como métodos em uma linguagem de programação) que essas entidades realizam e a comunicação (troca de mensagens entre elas). As trocas de mensagens são baseadas em parâmetros que se referem aos seus atributos.

Mesmo que não seja utilizado um banco de dados orientado a objetos, geralmente há um relacionamento bastante direto entre os atributos das classes e os campos das tabelas, além de entre classes e tabelas. Essas são as classes denominadas persistentes porque elas serão armazenadas (persistidas) em banco de dados.

Modelar e implementar um sistema como o apresentado neste trabalho não é tão fácil, são diversas tabelas, com muitos campos e relacionamentos envolvidos. Por isso exige muito cuidado, pois algum item esquecido no levantamento dos requisitos ou algum item colocado desnecessariamente pode prejudicar a sequência do trabalho.

Gerenciar um parque de máquinas de uma prefeitura não é uma tarefa trivial, inclui vários itens e aspectos de controle. Assim, um sistema para auxílio nesse gerenciamento apresenta várias vantagens, incluindo a agilidade nos serviços e a facilidade de controle. Mesmo porque, um controle feito por anotações manuais ou planilhas eletrônicas, especialmente se a frota contar com muitos veículos, certamente não é a alternativa mais adequada.

REFERÊNCIAS

AGARWAL, R.; SINHA, A. P. **Object-oriented modeling with UML: a study of developers' perceptions.** COMMUNICATIONS OF THE ACM, set. 2003, v. 46, n. 9, p. 248-253.

AMBLER, S.W. **The object primer 3rd edition: agile model driven development with UML 2**. New York: Cambridge University Press, 2004.

BERLITZ, I. H. **Gerador gráfico de relatórios utilizando a classe fpdf**. Trabalho de conclusão de curso. Centro Universitário Feevale. Instituto de Ciências Exatas e Tecnológicas. Curso de Ciência da Computação, 2007. Disponível em: <http://tconline.feevale.br/tc/files/1253.pdf>. Acesso em: 20 abr. 2011.

BOOCH, G. **Object oriented design with applications**. Benjamin/Cummings, 1991.

BOOCH, G.; RAMBAUGH, J.; JACOBSON, I. **UML guia do usuário**, Rio de Janeiro: Elsevier, 2000.

BOOCH, G.; RAMBAUGH, J.; JACOBSON, I. **The unified software development process**, Addison-Wesley, New York, 2001.

CAMPOS, A. **A IDE para modelagem de dados JUDE**. Disponível em: <http://br-linux.org/linux/node/333>. Acesso em: 25 set. 2010.

CANTU, C. H. **Get to know Firebird in 2 minutes (Conheça o Firebird em dois minutos)**. Disponivel em: <http://www.firebirdnews.org/docs/f>. Acesso em: 26 set. 2010.

CANTU, M. **Dominando o Delphi – a bíblia.** Rio de Janeiro: Makron Books Editora Ltda., 2002.

CANTU, M. **Dominando o Delphi 7 – a bíblia**. Rio de Janeiro: Makron Books Ltda, 2005.

CONRADI, R. et al. **EPOS: object-oriented and cooperative process modeling**, In: Software Process Modeling and Technology, 1994, p. 33-70.

FOWLER, M. **Analysis ptterns**, Reusable Objects Models. Addison-Wesley, 2000.

HIGHSMITH, J. **Extreme programming. Agile project management advisor service**, p. 1-19. Disponível em <http://rockfish-cs.cs.unc.edu/COMP290-agile/xp-highsmith.pdf>. Acesso em: 02 out. 2010.

HUMPHREY, W. S. **Managing the software process**. Reading, MA: Addison-Wesley, 1989.

IBEXPERT. **IBExpert dveloper sudio**. Disponível em: <http://www.ibexpert.com/>. Acesso em: 12 fev. 2010.

LARMAN, C. **Utilizando UML e padrões**. Bookman, 2000.

LI, W. Software product metrics. **IEEE Potentials**. v. 18, n. 5, dez.1999-jan.2000, p.24-27.

LYCETT, M., MACREDIE, R. D., PATEL C., Paul R. J., **Migrating agile methods to standardized development practice**, IEEE Computer Society, p. 79-85, Junho 2003.

MENASCÉ, D. A.; GOMAA, H. A method for design and performance modeling of client/server systems. **IEEE Transactions on Software Engineering,** v. 26, n. 11, p.1066-1085, November, 2000.

POTOK T.; MLADEN V. **Development productivity for commercial software using object-oriented methods.** The 1995 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada, p. 52-59, 1995.

PRESSMAN, **Engenharia de software**, 5ª ed., Rio de Janeiro: McGraw-Hill, 2002.

RUMBAUGH, J. et al. **Modelagem e projeto baseado em objeto**. Rio de Janeiro: Campus, 1997.

SCHMIETENDORF, E. DIMITROV, R. DUMKE. **Process models for the software development and performance engineering tasks**. Workshop on Software and Performance (WOSP '02), 2002, ACM Press, p. 211-218.

SOMMERVILLE, I. **Software engineering**. Addison-Wesley, 2003.

WAGENHALS, L. W.; HAIDER, S.; LEVIS A. H. **Synthesizing executable models of object oriented architectures.** Conference on Application and Theory of Petri Nets: Formal Methods in Software Engineering and Defense Systems, 2002, p. 85 - 93.