UNIVERSIDADE ESTADUAL DO NORTE DO PARANÁ

CAMPUS LUIZ MENEGHEL

LUCAS JOSÉ CORRÊA CHAGAS

DESENVOLVIMENTO EM TRÊS CAMADAS DE UM PROTÓTIPO DE SISTEMA GERENCIAL PARA AUTOPEÇAS E AUTO MECÂNICA.

Bandeirantes.

Junho – 2009.

LUCAS JOSÉ CORRÊA CHAGAS

DESENVOLVIMENTO EM TRÊS CAMADAS DE UM PROTÓTIPO DE SISTEMA GERENCIAL PARA AUTOPEÇAS E AUTO MECÂNICA.

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, Campus Luiz Meneghel, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Ms. Glauco Carlos Silva

Bandeirantes.

Junho – 2009.

LUCAS JOSÉ CORRÊA CHAGAS

DESENVOLVIMENTO EM TRÊS CAMADAS DE UM PROTÓTIPO DE SISTEMA GERENCIAL PARA AUTOPEÇAS E AUTO MECÂNICA

Trabalho de Conclusão de Curso submetido à Universidade Estadual do Norte do Paraná, Campus Luiz Meneghel, como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

COMISSÃO EXAMINADORA

_____

Prof. Ms. Glauco Carlos Silva
UENP – Campus Luiz Meneghel

_____

Prof. Ma. Cristiane Yanase Hirabara de Castro
UENP – Campus Luiz Meneghel

_____

Prof. Ms. Ricardo Gonçalves Coelho
UENP – Campus Luiz Meneghel

Bandeirantes, __ de_____ de 2009.

A Deus, por mais esta etapa.

Aos meus pais e irmã.

A minha amada esposa Solange.

A minha amada filha Alice.

AGRADECIMENTOS

Agradeço primeiramente a Deus e a Nossa Senhora Aparecida.

A meus pais razão e cumplicidade da minha vida. A minha esposa, pela compreensão, cumplicidade e apoio nos momentos difíceis.

Ao meu orientador Glauco, pela paciência e determinação durante o desenvolvimento do trabalho. Aos professores Ricardo e Cristiane, pelas dicas e por se mostrarem sempre dispostos a ajudar. A todos os professores do Departamento de Informática, que contribuíram para o meu aprendizado e dinâmica.

Aos verdadeiros amigos Kenion, Lucas e Kalil que sempre estiveram ao meu lado me apoiando e me incentivando nos melhores e piores momentos vividos nestes quatro anos e em toda a minha vida. Aos grandes amigos da VIII e IX Turmas do Curso de Sistema de Informação.

Enfim, agradeço a todos aqueles que direta ou indiretamente participaram e contribuíram para transformar dificuldade em objetividade e dinâmica.

"Quero melhorar em tudo. Sempre.”

Ayrton Senna da Silva

CHAGAS, Lucas. **Desenvolvimento em três camadas de um Protótipo de Sistema Gerencial para autopeças e auto mecânica.** 2009. 99 p. Monografia (Graduação em Sistemas de Informação) – Universidade Estadual do Norte do Paraná.

RESUMO

Este trabalho visa estudar e aplicar a arquitetura de software em multicamadas através da especificação e implementação de um protótipo de sistema gerencial para autopeças e auto mecânica. Foi utilizado na implementação a linguagem Object Pascal, explorando os recursos do *Code Gear Rad Studio* 2009 – *Trial Version*, utilizando como servidor de aplicação a tecnologia *MIDAS* e acessando o banco de dados *Firebird* 2.0.3.12981-1.

Palavras-chave: arquitetura de três camadas, desenvolvimento incremental, documentação UML.

CHAGAS, Lucas. **Development in three layers of a Prototype Management System for automotive and auto mechanics**. 2009. 99 p. Monograph (Graduation in Information Systems) - University of Northern Paraná.

ABSTRACT

This work aims to study and implement the software architecture in multilayer through specification and implementation of a prototype management system for auto and auto mechanics. Was used to implement the Object Pascal language, exploiting the resources of the Code Gear Rad Studio 2009 - Trial Version, using application server technology MIDAS and accessing the database Firebird 2.0.3.12981-1.

Key-words: architecture of three layers, incremental development, UML documentation.

LISTA DE ILUSTRAÇÕES

LISTA DE QUADROS

LISTA DE SIGLAS

ASP – *Active Server Pages.*

UML – *Unified Model Language*.

OOSE - *Object Oriented Software Engeneering*.

OMT – *Object Modeling Technique*.

SGBD – Sistema Gerenciador de Banco de Dados.

PHP – *Hypertext Preprocessor.*

P2P – *Peer to Peer*.

MIDAS – *Middle – tier Distributed Application Services*.

CPF – Cadastro de Pessoas Físicas.

CNPJ – Cadastro Nacional de Pessoas Jurídicas.

DLL – *Dynamic Library Link.*

EXE – *Directly Executable Program.*

.NET – Dot.Net.

GRASP - *General Responsibility Assignment Software Patterns.*

TCP/IP – *Transmission Control Protocol / Internet Protocol*.

CORBA – *Common Object Request Broker Architecture*.

SQL - *Structure Query Language.*

IP - *Internet Protocol*.

LAN – *Local Area Network.*

WAN – *Wide Area Network.*

SUMÁRIO

1. INTRODUÇÃO

Segundo Pacheco (2005), a *Internet* apesar de representar um ganho expressivo em informações de diversos tipos, ainda sofre com a falta de interatividade entre os serviços oferecidos pelas corporações.

Na tentativa de solucionar este problema, os gerentes de sistemas de informação enfrentam com um dilema: criar uma vantagem competitiva para a organização, desenvolvendo, distribuindo e gerenciando aplicações distribuídas, que sejam escaláveis através de uma *Local Area Network* (LAN), *Wide Area Network* (WAN) e Internet, preservando investimentos da corporação.

Dentro deste contexto, uma solução seria a implementação de uma arquitetura de computação distribuída em multicamadas (*multi-tier*), pois esta tem o potencial de prover melhores resultados para a organização a um custo mais baixo do que a combinação de um sistema cliente/servidor.

O modelo multicamadas divide a aplicação nas seguintes camadas, porém, pode existir qualquer número de qualquer das camadas em uma aplicação:

- lógica de apresentação: apresenta a informação a uma fonte externa e obtém entradas daquela fonte;
- lógica de negócio: contém a lógica da aplicação que governa as funções do negócio e os processos executados pela aplicação;
- lógica de acesso a dados: contém a lógica que fornece à interface um sistema de armazenamento de dados ou com algum outro tipo de fonte de dados externos, como uma aplicação externa.

A abordagem arquitetural em três camadas significa a criação de uma arquitetura de software em níveis, na qual há a completa separação dos serviços de dados e dos serviços de negócios (modelo do domínio) dos serviços dos usuários (interface do usuário), onde cada camada possui regras de negócio dentro de classes e operações em objetos de negócio.

O Delphi assim como outras linguagens, oferece suporte para este tipo de desenvolvimento, possibilitando a criação de uma aplicação completa nesta arquitetura.

Neste projeto será abordado como desenvolver um aplicativo na arquitetura de multicamadas utilizando o Delphi como ferramenta de desenvolvimento.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral é desenvolver um protótipo de um Sistema Gerencial para Autopeças e Auto Mecânica sob o aspecto de uma aplicação de múltiplas camadas. Para isso o será utilizado o modelo de desenvolvimento incremental descrito por Sommerville (2003) e a UML descrita por Booch (2005).

1.1.2 Objetivos Específicos

Os objetivos específicos do sistema são:

- Gerar um sistema confiável onde os erros dos usuários sejam minimizados;
- A comunicação entre as três camadas deve ser total, onde a camada de dados jamais poderá ser acessada diretamente pela camada de visualização;
- Acesso restrito ao sistema, onde somente um usuário cadastrado poderá acessar suas funções;
- O processo de desenvolvimento e a documentação entregue deverão estar de acordo com o modelo de desenvolvimento incremental descrito por Sommerville (2003) e os artefatos da UML descritos por Booch (2005);
- Facilitar a manutenção e distribuição de atualizações futuras do sistema, sendo que vários métodos serão implementados no servidor de aplicações;
- O sistema deverá possuir controles intuitivos, facilitando sua utilização para usuários com qualquer nível de experiência na informática.

1.2 Justificativa

O desenvolvimento deste protótipo deve ser realizado para aplicação de um método de desenvolvimento descrito pelos autores Pacheco, com o auxílio das técnicas de modelagem da UML (BOOCH, 2005), a fim de gerar um protótipo de aplicação com capacidade para grande escalabilidade, flexibilidade para futuras atualizações e fácil manutenção.

1.3 Metodologia

O desenvolvimento do sistema será feito de no Modelo incremental, ou seja, cada parte do sistema irá gerar um incremento, que adicionará uma nova funcionalidade ao sistema como um todo.

A UML será usada como linguagem de modelagem e tem por função auxiliar na visualização dos requisitos do sistema, bem como relacionamentos entre classes e objetos, como descrição passo a passo dos processos utilizados nos sistemas.

O protótipo será desenvolvido primeiramente na camada de dados com a criação do banco de dados. Ao se finalizar a criação do banco de dados a camada de acesso ao banco, o servidor remoto será criado, juntamente com o servidor remoto se dará o início do desenvolvimento da aplicação cliente, sendo que ao final de cada incremento irá gerar uma nova funcionalidade para o cliente e atualizará o servidor de aplicações para que as regras de acesso a dados e regras de validação sejam implementadas.

Após o término de toda a implementação, uma documentação do sistema será criada, para que o estudo chegue a uma conclusão sobre o método de desenvolvimento, bem como questões de abstração de requisitos como também na aplicação dos mesmos.

Para gerar a documentação, o uso da UML será imprescindível, pois todos os processos criados durante a implementação deverão ser documentados para um

maior entendimento de como o protótipo irá funcionar, e o porquê de seu funcionamento ocorrer daquela maneira.

1.4 Organização do Trabalho

No capítulo 2 é apresentada a revisão de literatura, abordando a UML e métodos de desenvolvimento. No capítulo 3 é apresentado o desenvolvimento do protótipo, especificando os requisitos do sistema e descrevendo o processo de desenvolvimento. No capitulo 4 é apresentada a conclusão do trabalho, descrevendo acertos e dificuldades encontradas durante o desenvolvimento. No capítulo 5 são apresentadas as referências do trabalho. O último capítulo contém somente os apêndices do trabalho.

2. REVISÃO DE LITERATURA

A revisão de literatura irá abordar alguns tópicos para que o processo de desenvolvimento do protótipo ocorra de acordo com padrões de processos e documentos necessários para a criação de um *software* que atenda seus requisitos de forma plena.

A revisão de literatura está organizada na seguinte seqüência: Engenharia de S*oftware*, UML – Linguagem Unificada de Modelagem, *Design Patterns* – Padrões de Projeto de *Software* e Aplicações de Múltiplas Camadas.

2.1 Engenharia de *Software*

A Engenharia de *software* é uma área da engenharia e tem como principal objetivo dar suporte para a especificação, desenvolvimento e manutenção de sistemas com o melhor custo benefício.

Para isso se ocupa de todos os aspectos da produção de *software*, desde o levantamento de dados até o *software* completo. Segundo Sommerville (2003), no desenvolvimento são usados os processos de *softwares*. Cada processo é um conjunto de atividades e resultados que geram um produto de *software*.

Entre eles possuem quatro processos básicos:

1. Especificação de *software*;
2. Desenvolvimento de *software*;
3. Validação do *software*;
4. Evolução do *software*.

Na engenharia de *software*, também existem alguns modelos que são considerados modelos gerais, estes são:

1. Modelo em cascata;
2. Modelo evolucionário;
3. Desenvolvimento formal de sistema;
4. Desenvolvimento orientado a reuso;
5. Desenvolvimento incremental;

6. Desenvolvimento em espiral;
7. Transformação formal;
8. Montagem de um sistema a partir de componentes reutilizáveis.

Mesmo existindo inúmeros processos de *softwares* utilizáveis, todos eles possuem atividades que são consideradas comuns a todos (SANTOS 2009). A especificação de *software* tem como objetivo definir a funcionalidade de um *software* e suas restrições. O projeto e a implementação são as atividades que produzem um software, e este deve seguir as especificações coletadas no levantamento de requisitos.

O projeto deve ser implementado para que possa assegurar que ele atenda os requisitos que o cliente requisitou. Um software precisa evoluir, pois os requisitos do cliente não são imutáveis, e nem seu software. Alguns processos podem ser considerados paradigmas de processos, pois são modelos genéricos e não descrições definitivas, mas sim abstrações úteis que podem ser usadas para explicar diferentes abordagens do desenvolvimento do software. Quando um sistema é muito complexo, podem ser usados vários processos para desenvolver várias partes do sistema.

Sommerville (2003) afirma que:

> "*um modelo de processo de software é uma representação abstrata de um processo de software. Cada modelo de processo representa um processo a partir de uma perspectiva partícula, de uma maneira que proporciona apenas informações parciais sobre o processo."*

Quando um sistema é muito complexo podem ser usados vários modelos de processos para desenvolver várias partes do sistema.

2.1.1 Modelo cascata

O modelo em cascata é um modelo seqüencial, onde uma fase é subseqüente a outra, e em uma determinada ordem.

As fases do modelo em cascata são:

1. Análise e definição de requisitos;
2. Projeto de sistemas e de software;
3. Implementação e teste de unidades;
4. Integração e teste de sistemas;
5. Operação e manutenção.

Ao final de cada fase, um ou mais documento é gerado e este documento precisa ser aprovado, sendo que uma fase não pode ser iniciada sem que a outra seja concluída com sucesso.

O modelo em cascata gera muito retrabalho, pois um erro encontrado durante a codificação de uma fase, sendo que os problemas encontrados podem ser resolvidos com atraso ou podem fazer com que uma iteração não seja corretamente desenvolvida, deixando que a solução do problema seja programada para um momento mais a frente, possibilitando que o novo software não atenda os requisitos do cliente.

Sendo assim, o modelo em cascata se torna um tanto inflexível, considerando que os requisitos do cliente sempre mudam, tornando o processo mais difícil.

2.1.2 Desenvolvimento Evolucionário

Neste modelo, o desenvolvimento se faz através da criação e exposição ao cliente de várias versões do software, cada versão gerando uma melhoria requisitada pelo cliente, eliminando etapas como levantamento de dados em separado e usando um *feedback* por meio dessas atividades de evolução. Sendo assim o desenvolvimento evolucionário se faz de duas maneiras, com o desenvolvimento exploratório e com protótipos descartáveis.

No desenvolvimento exploratório o desenvolvedor trabalha diretamente expondo o sistema ao cliente, assim listando os requisitos compreendidos e gerando uma nova versão do sistema.

Com protótipos descartáveis o objetivo é compreender e gerar um sistema que melhor se encaixe nos requisitos. Os experimentos são com partes mal compreendidos do sistema.

2.1.3 Desenvolvimento Formal de Sistemas

O desenvolvimento formal de sistemas é uma abordagem que possui pontos em comum com o modelo em cascata, mas seu processo tem como base a matemática, ou seja, a transformação matemática formal de uma especificação de sistema em um executável.

A representação é transformada gradualmente em um sistema, sendo que quanto mais a transformação formal avança, mais detalhado o sistema irá se tornar. Uma vantagem é que a distância entre as transformações é menor do que a distância entre a especificação e o programa. Sommerville (2003) ainda afirma que quanto menores forem as etapas, mais viável o processo de transformação será.

Esta abordagem é recomendada para sistemas que requerem alto nível de segurança, também é necessário que quem utilizar este modelo tenha conhecimento especializado, sendo que este modelo de processo não é largamente utilizado fora dos domínios da engenharia de *software*.

2.1.4 Desenvolvimento Orientado a Reuso

Esse método consiste em reaproveitamento de códigos, com adaptações para o nosso sistema que esteja sendo criado.

Muitas vezes, algumas funções requeridas já se encontram criadas e planejadas para que possam ser reaproveitadas por qualquer sistema a qualquer hora, são os chamados sistemas de prateleiras, que adicionam funcionalidades completas, como edição de texto, planilhas eletrônicas.

Além disso, o reuso é utilizado informalmente em outros métodos, em casos em que o desenvolvedor conhece algum código semelhante e o adapta ao novo código.

É um método que propicia velocidade de desenvolvimento, mas em casos de adequações de código, os requisitos do cliente podem não ser atendidos, e outro aspecto importante é que a evolução dos componentes usados não está sob controle da equipe de desenvolvimento, o que pode gerar problemas de evolução para o cliente.

2.1.5 Desenvolvimento Incremental

Segundo Sommerville (2003), mesmo os modelos apresentando vantagens e desvantagens, em alguns casos, há a necessidade de utilizar diferentes abordagens para diferentes partes do sistema, assim gerando um modelo híbrido para utilização.

O desenvolvimento incremental apóia a iteração de processo, onde partes do processo de software são repetidas na medida em que os requisitos do cliente evoluem, gerando um retrabalho em alguns casos.

Segundo Sommerville (2003), o desenvolvimento incremental é uma abordagem intermediária, entre o modelo cascata e o desenvolvimento evolucionário. Este modelo foi sugerido por Mills, e combina as vantagens de ambos os modelos.

Neste modelo, as fases são desdobradas em vários estágios, que são desenvolvidos um por vez. Assim, o modelo incremental exige que os clientes tenham fortes requisitos pré - definidos e que os desenvolvedores se atenham a estratégias em particular, assim quando um requisito é alterado, ele gera um retrabalho em toda a iteração, ou seja, uma nova iteração é programada para que o sistema atenda as novas requisições do sistema.

Assim, no modelo incremental, o sistema é entregue em partes, da mais prioritária a menos prioritária, podendo assim fazer com que o cliente possa ter uma prévia experiência com o sistema assim modificando os requisitos para um novo incremento futuro.

Um incremento por ser considerado como as partes do software a serem entregues no decorrer do desenvolvimento do sistema.

Da mesma forma, quando mais crucial é o incremento a ser desenvolvido, mais tempo em fase de testes este incremento possuirá.

2.1.6 Desenvolvimento em Espiral

Assim como no desenvolvimento incremental, o desenvolvimento em espiral também oferece apoio às iterações no desenvolvimento de um software.

Este processo é representado como uma espiral, onde cada volta da espiral representa uma fase do processo de software. Neste modelo a preocupação com o tratamento de riscos de projetos é evidente, também sendo ordenado por objetivos que o sistema deve atingir.

O modelo em espiral não possui fases fixas, mas ele abrange alguns outros modelos, podendo seguir um desenvolvimento em cascata convencional para cada um de seus requisitos, assim como a transformação formal para requisitos de segurança.

2.2 UML – Linguagem Unificada de Modelagem

Segundo Booch (2005), a UML é uma linguagem gráfica que tem por função padronizar a criação de planos de arquiteturas de projetos, usando para isso conceitos visuais que podem ser aplicados em qualquer linguagem de programação posteriormente.

Para a criação do modelo para este sistema, será usada a UML 2.0, como ponto de partida para a arquitetura.

2.2.1 História da UML

Segundo Booch (2005), a UML atual tem como base três metodologias anteriores, que são:

- Booch;
- OOSE (*Object Oriented Software Engeneering*);
- OMT (*Object Modeling Technique*).

Estas eram as metodologias para criação se softwares orientados a objetos. O Booch era mais expressivo, principalmente nas fases de projeto e na construção do sistema. Enquanto que o OOSE fornecia suporte para casas como uma maneira de controlar os requisitos, análise e projeto em alto nível. E o OMT era mais útil para análise e sistemas de informações com uso intenso de dados. Essas metodologias eram reconhecidas como as principais na metade da década de 90.

Devido à proximidade cada vez maior dos três métodos, Grady Booch, Ivar Jacobson e James Rumbaugh decidiram que, em vez de desenvolverem três métodos distintos e que, poderiam gerar diferenças e confusões entre os usuários, era mais interessante um único modelo que trouxesse estabilidade ao mercado de softwares orientados a objetos, permitindo que vários projetos possuam a mesma base metodológica em sua criação.

Segundo Booch (2005), três objetivos foram estabelecidos na unificação:

1. Fazer a modelagem, do conceito ao artefato executável, com a utilização de técnicas orientadas a objetos;
2. Incluir questões de escala inerentes a sistemas complexos de tarefas críticas;
3. Criar uma linguagem de modelagem a ser utilizada por humanos e máquinas.

A UML teve seu início em Outubro de 1994 quando os esforços para sua criação se iniciaram. Em 1996, um consórcio da UML com várias empresas incluindo Microsoft e IBM, foi responsável pelo desenvolvimento da UML em sua versão 1.0. A UML 2.0 é uma importante atualização, trazendo novos recursos capazes de facilitar mais a criação de softwares.

2.2.2 Visão geral sobe UML

A UML sendo uma linguagem de visualização, especificação, construção e documentação, tem como objetivo gerar artefatos de um sistema de software.

Segundo Booch (2005), um artefato na UML representa pacotes físicos de bits, onde são identificados três tipos de artefatos: os de implementação, do produto do trabalho e de execução.

Os artefatos de implementação são aqueles necessários para formar um programa executável, como DLL – *Dynamic Library Link e* EXE – *Directly Executable Program.* Os artefatos do produto do trabalho são arquivos como arquivos de código-fonte e arquivos de dados. Os artefatos de execução são aqueles criados como conseqüências de um sistema, podem ser objetos .Net que são instanciados em uma DLL.

A UML tem como objetivo sistemas mais complexos, mesmo assim ela não é restrita somente na modelagem de softwares (DEBONI, 2009), a UML também pode abranger fluxos de trabalhos e outras áreas, como administração por exemplo.

2.2.3 Principais Diagramas da UML 2.0

Os diagramas da UML incluem três divisões de diagramas básicos (BOOCH, 2005):

1. Diagramas de comportamento;
2. Diagramas de interação;
3. Diagramas de estrutura.

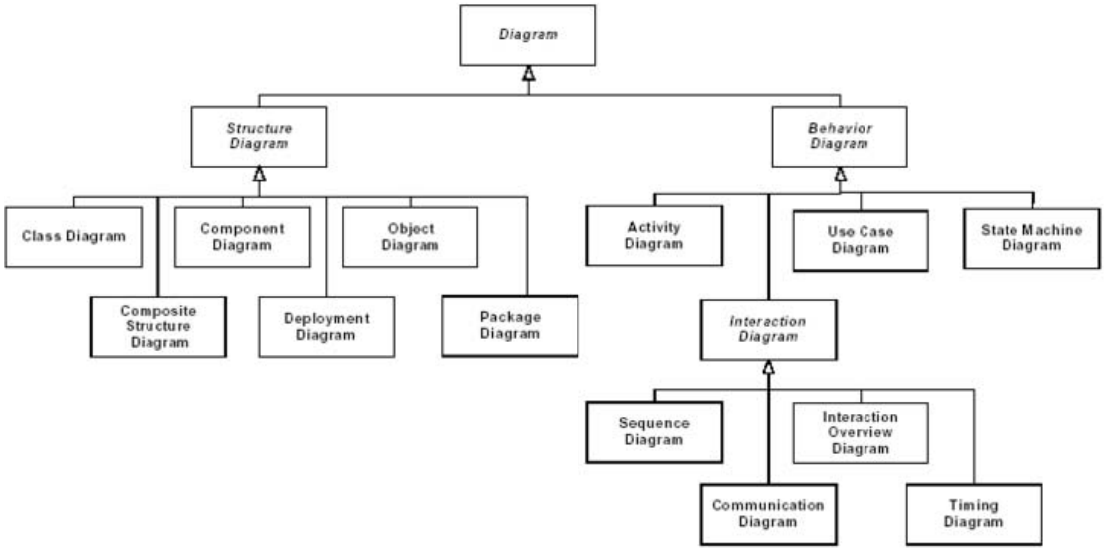

Figura 1 - Hierarquia dos Diagramas da UML

Segundo Amaral (2009), os diagramas de comportamento representam características de um sistema ou de um processo de um negócio, que são:

- Diagrama de caso de uso: descreve a funcionalidade proposta para o novo sistema, representando a interação entre um usuário com o sistema;

- Diagrama de transição de estados: é a representação do estado ou situação que um objeto pode se encontrar no decorrer da execução de processos do sistema;
- Diagrama de atividade: representa fluxos conduzidos por processamentos. Como é um fluxo gráfico, mostra o fluxo de uma atividade para outra, sendo comum em desenvolvimento de sistemas computadorizados.

Um exemplo de generalização de atores de casos de uso pode ser vista na figura abaixo.

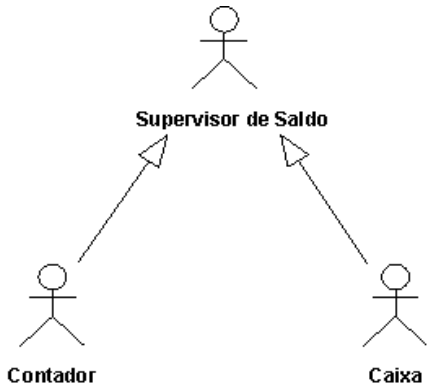

Figura 2 - Exemplo de Generalização de Atores de Casos de Uso

Segundo Booch (2005) os diagramas de interação são por sua vez uma subdivisão dos diagramas de comportamento, que tem por objetivo enfatizar as interações de objetos dentro do sistema.

- Diagrama de seqüência: representa a seqüência de processos, sendo mais específico em mensagens passadas entre objetos dentro de um programa de computador;
- Diagrama de interatividade: normalmente são extensões do diagrama de atividades, formando um fluxo de atividades, mostrando seu trabalho em uma seqüência de eventos;
- Diagrama de colaboração ou comunicação: mostra de maneira semelhante ao diagrama de seqüência a colaboração dinâmica entre objetos;
- Diagrama de tempo: apresenta o comportamento dos objetos e sua interação em escala temporal, focalizando as mudanças ocorridas no período de tempo.

Os diagramas de estrutura representam elementos não temporais dentro de um sistema (MACORATTI, 2009).

- Diagrama de objetos: é uma variação do diagrama de classes, utilizando quase a mesma conotação, sendo que a diferença é que o diagrama de objetos mostra os objetos que foram instanciados das classes.
- Diagrama de classes: representação da estrutura e relações das classes que servem de modelo para os objetos. Aqui ficam definidas todas as classes que o sistema necessita possuir e é base para construção dos diagramas de comunicação, seqüência e estados;
- Diagrama de componentes: ilustra como as classes deverão se encontrar organizadas através da noção de componentes de trabalho;
- Diagrama de instalação: tem por função descrever componentes de hardware e software e sua interação com outros elementos de suporte ao processamento, representando a configuração e a arquitetura de um sistema em que estarão ligados seus respectivos componentes, sendo representado pela arquitetura física de hardware;
- Diagrama de pacotes: descreve pedaços ou pacotes do sistema divididos em agrupamentos lógicos, mostrando também a dependência entre estes, ou seja, pacotes podem depender de outros pacotes;
- Diagrama de estrutura: descreve os relacionamentos entre os elementos. Sendo utilizadas para descrever a colaboração interna de classes, interfaces ou componentes para especificar uma funcionalidade.

2.3 Aplicações de Múltiplas Camadas

Quando desenvolvemos aplicações em múltiplas camadas, as camadas funcionam como um separador de responsabilidades.

Segundo Pacheco (2005), no desenvolvimento de aplicações orientadas a objetos vêm ocorrendo uma evolução na maneira de desenvolver sistemas, onde grande parte dessas mudanças se deve ao fato de que a web está cada dia crescendo, e a necessidade de serviços desenvolvidos com maior agilidade, eficiência e com menor tempo de desenvolvimento são crescentes.

Devido ao aumento da demanda, os métodos de desenvolvimento têm passado por grandes mudanças desde o surgimento da *Internet*, pois os serviços que antes estavam centralizados em um único local, agora podem ser distribuídos em vários locais, devido a facilidade de acesso proporcionada pela *Internet*.

As aplicações eram desenvolvidas em apenas uma camada, que ficava responsável por armazenar, verificar, implementar lógica e acesso ao banco de dados, o que tornava este tipo de sistema caro, e de difícil manutenção devido ao grande número de linhas implementadas. Estes sistemas também eram conhecidos como aplicações monolíticas ou aplicações de uma camada.

Devido à necessidade de compartilhamento de lógica de acesso aos dados, sugiram as aplicações de duas camadas ou aplicações cliente/servidor. Neste tipo de sistema, um SGBD (Sistema Gerenciador de Banco de Dados) é instalado em uma máquina servidora, sendo esta camada também conhecida como camada de dados. Já do lado do cliente, temos a outra camada da aplicação, que é composta das camadas de lógica de negócios ou regras de negócio e da camada de apresentação ou interface com o usuário. Esta camada é chamada de cliente rico ou cliente gordo, pois todas as verificações do sistema, validações e visualizações de dados do sistema estão presentes nesta camada.

Segundo Pacheco (2005), com o surgimento da Internet e grande necessidade de serviços que possam garantir o acesso as mesmas informações sem que o cliente necessite instalar as aplicações em sua máquina, um movimento surgiu para que as camadas de lógica de negócios e apresentação fossem separadas. Deste modo, a lógica de negócios é separada, deixando todo o processamento sendo feito em outro local, o cliente tornou-se muito mais leve, pois agora só possui a camada de apresentação. Este novo cliente recebeu o nome de cliente magro ou *thin-client*.

Neste modelo o aplicativo onde reside todo o processamento é movido para um Servidor e um navegador *web* ou um *software* de apresentação se conecta a ele, recebendo as informações que são processadas no servidor. Este modelo recebeu o nome de aplicação de três camadas ou aplicação de múltiplas camadas. Quando este método é utilizado, são geradas no mínimo duas aplicações, uma que reside no servidor e tem acesso completo ao sistema de banco de dados, e outra que reside

na máquina cliente, sendo somente uma interface do sistema, sem acesso direto ao banco de dados, como mostrado na figura abaixo.

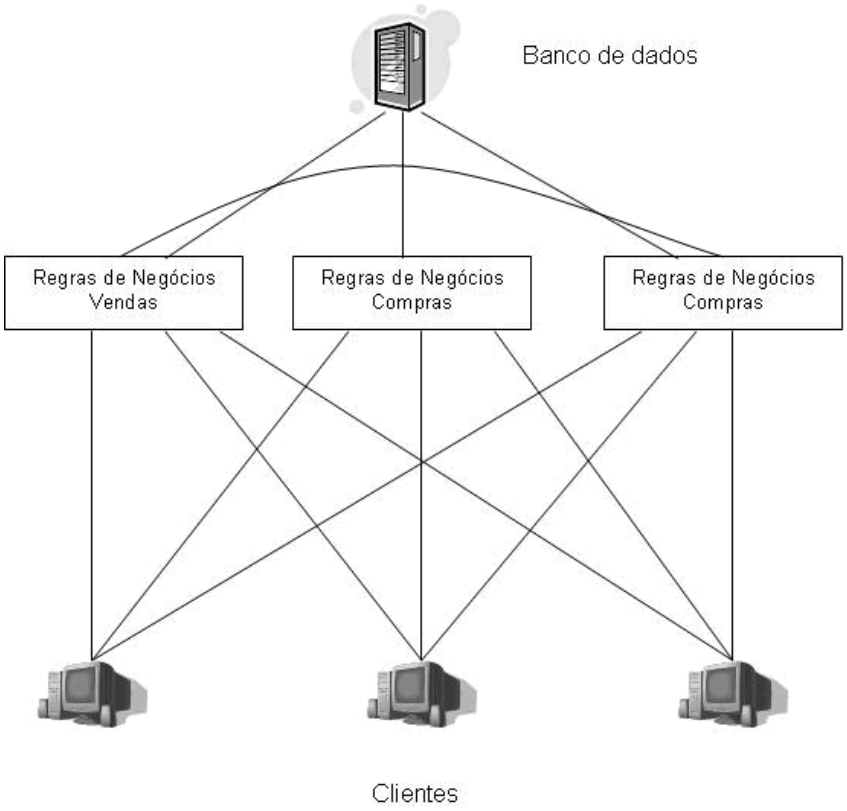

Figura 3 - Exemplo Múltiplas Camadas

Esta arquitetura possibilita dividir as funcionalidades envolvidas na manutenção e na apresentação dos dados da aplicação. Esta arquitetura não é nova. Ela foi desenvolvida para mapear tarefas de entrada, processamento e saída para interação com o usuário. Abaixo um exemplo de um sistema em três camadas, que utiliza um browser para acesso do cliente (LIMA, 2009).

2.3.1 Camada de Apresentação ou Visualização

Esta camada somente exibe as informações para o usuário, sem a preocupação de como elas são obtidas. Em um sistema de três ou mais camadas, um sistema pode possuir várias interfaces. Exemplos disso são sistemas que possuem interfaces de controles feitas em Delphi, Java ou Visual Studio e também

possuem interfaces para acesso do cliente via internet como o PHP (*Hypertext Preprocessor)* ou ASP (*Active Server Pages*).

2.3.2 Camada de Controle ou Intermediária

Camada que tem por objetivo servir de intermédio entre a camada lógica (banco de dados) e a camada de apresentação. Nesta camada é onde as regras de negócio devem residir como validação, cálculos de porcentagem, acesso ao SGBD, etc.

2.3.3 Camada Lógica da Aplicação ou Dados

Esta camada é responsável pelo armazenamento das informações recebidas das camadas superiores. Também é onde residem sistemas de Bancos de Dados, como Interbase e Oracle.

Estas camadas ficam dispostas de modo que o cliente nunca acesse diretamente o SGBD, como mostrado na figura abaixo.

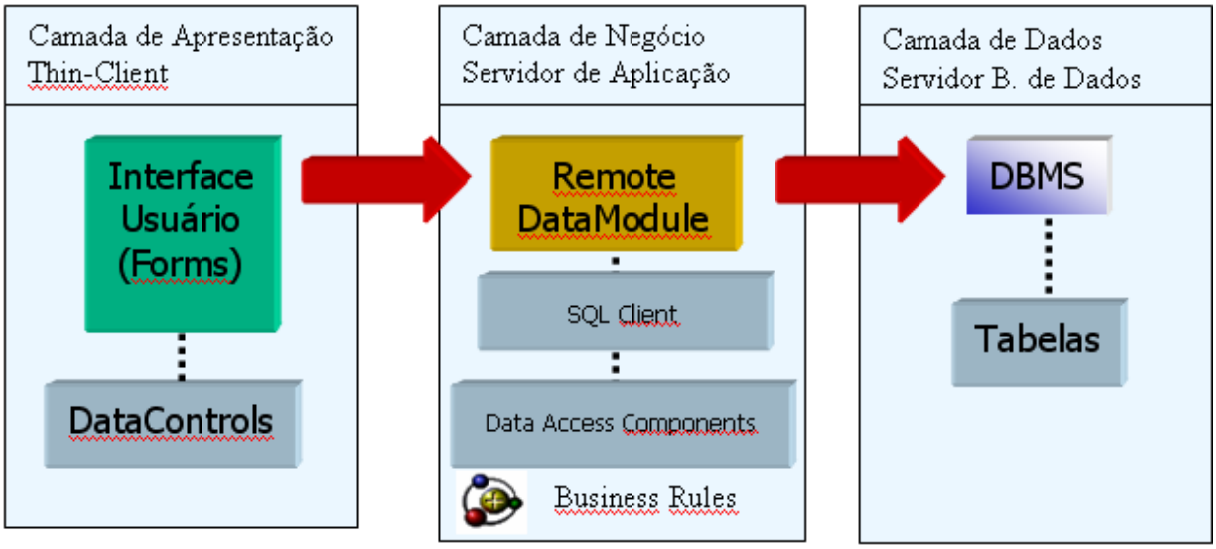

Figura 4 - Camada de Apresentação, Camada de Negócios e Camada de Dados

2.3.4 Vantagens em Desenvolver em Múltiplas Camadas

O desenvolvimento em multicamadas possui algumas vantagens em relação ao modelo cliente/servidor. Segundo Pacheco (2005) um projeto desenvolvido sob o modelo de múltiplas camadas possui maior escalabilidade, maior facilidade de atualização do sistema, ou alteração de funções que estão localizadas na aplicação servidora.

Outro aspecto, é que as estações clientes não serão configuradas para acesso direto ao banco de dados, e sim configurado para acessar a camada de negócios.

Também deve ser levada em consideração a abstração para a criação do sistema. Sendo uma aplicação de múltiplas camadas, ela possui uma camada de negócios, que tem por finalidade trabalhar as regras de negócio implícitas no sistema.

Para isso, algumas regras que ficavam implícitas no próprio banco de dados ou na camada de apresentação, podem ficar “alojadas” na camada intermediária, como verificação de campos vazios em um formulário, ou campos auto – incrementáveis. Ao utilizar este método de desenvolvimento, a utilização de recursos do banco de dados pode ser reduzida, sendo que checagens ou validações podem ser feitas fora do banco.

Outra vantagem se encontra em uma possível migração de base de dados, pois o código da camada intermediária fica praticamente intacto, sendo que somente a camada de dados é alterada.

Vamos usar como exemplo uma validação de CPF (Cadastro de Pessoas Físicas), onde ao digitar o número do CPF, o sistema cliente envia o número para verificação no servidor, o qual possui sua rotina de verificação, caso necessite de uma atualização na rotina de verificação, a atualização é feita diretamente no servidor, em vez de ser feita no sistema do cliente.

Deste modo, em vez de se atualizar a rotina em todos os clientes, e depois redistribuir uma nova versão do sistema, esta atualização é feita na aplicação servidora (PACHECO, 2005).

2.3.5 Desvantagens em Desenvolver em Múltiplas Camadas

Do mesmo modo que existem vantagens sensíveis para o desenvolvimento em múltiplas camadas, também existem desvantagens que devem ser levadas em consideração antes de se escolher este método.

O custo de licenças para a distribuição das aplicações. Se usarmos como exemplo o Delphi, que utiliza a tecnologia DataSnap, é necessário que se adquira a licença do MIDAS (*Middle – tier Distributed Application Services*) para que a aplicação seja distribuída. Mesmo assim, é mais vantajoso em relação à compra de licenças de conexão com o banco de dados.

Outro aspecto é com relação à auditoria por usuário. Para fazer a auditoria neste tipo de aplicação, o desenvolvedor deve criar uma aplicação interna, sendo que o controle de auditoria do banco de dados não poderá ser usado.

Também pode ser considerado como desvantagem, que para o desenvolvimento é necessário investimento em aprendizagem. Muitas vezes, por mais que um desenvolvedor ou um grupo de desenvolvedores possuam experiência, nem todos tiveram a oportunidade de desenvolver em múltiplas camadas, tornando o processo de desenvolvimento mais lento.

A velocidade na produção da primeira versão do sistema é mais lenta do que em outras abordagens, devido à programação de não um único sistema, mas sim de dois ou mais, dependendo da aplicação.

2.3.6 Outras linguagens

Da mesma forma que o Delphi possui a capacidade para o desenvolvimento em múltiplas camadas, outras linguagens como Java, Visual Basic, entre outras também possuem esta capacidade.

Em teoria, os aplicativos podem ser desenvolvidos em linguagens distintas. Um exemplo seria um servidor desenvolvido com o Object Pascal do Delphi, sendo acessado por clientes ASP, Java e Visual Basic, assim possibilitando que diferentes equipes de desenvolvimento com conhecimentos diferentes possam desenvolver um único sistema integrado.

2.3.7 MIDAS

PACHECO (2005) explica que no ambiente Delphi, a biblioteca responsável pela implementação é a MIDAS, ou Datasnap, que é como a bibiloteca MIDAS se apresenta a partir da versão 6 do Delphi.

Em uma aplicação multicamadas, a camada de cliente não difere visualmente de um software feito sob o aspecto de cliente/servidor. No entanto, ele deve possuir meios para se conectar a aplicação remota.

Segundo CANTU (2005) o DataSnap possui vários protocolos de conexão, que possuem características únicas, como o *Windows Sockets (TCP/IP (Transmission Control Protocol / Internet Protocol))* e o *CORBA (Common Object Request Broker Architecture)*.

Quase todos os protocolos possuem um executável em tempo de execução que deve ser acionado no servidor, para que a conexão seja orientada para a porta correta do servidor.

CANTU (2006) ainda explica que a aplicação servidora pode ser tanto um executável como um arquivo DLL, que deve estar corretamente instalado no servidor. Apesar de ser basicamente a mesma coisa, o Delphi diferencia os dois. Quando o servidor é feito com base em um executável, o objeto a ser usado é o *Remote Data Module*.

O *Remote Data Module* possui uma interface que é por onde a aplicação cliente se conecta, esta interface é chamada *IAppServer*. Dentro desta interface podem ser definidos métodos que serão executados remotamente pelos clientes.

Da mesma forma, quanto o servidor é feito com base na DLL, o objeto a ser usado é o *Transactional Data Module*. Assim como o *Remote Data Module*, o *Transactional Data Module* possui uma interface de conexão com o mesmo nome.

Um SGBD deverá ser escolhido e instalado em uma estação servidora remota, possibilitando o acesso por parte dos servidores de aplicação. Para que o servidor de aplicações se conecte ao banco de dados remoto, componentes comuns podem ser utilizados dentro do servidor de aplicações.

Para implementar a aplicação servidora, uma aplicação comum deve ser iniciada no Delphi, após isso, deve-se acrescentar um *Remote Data Module*, que está disponível na guia *File > New > Other > Delphi Projects > Multtier* do Delphi (CANTU,2006) e um *Data Module*.

O *Data Module* ficará responsável pela conexão direta com o banco de dados remoto, podendo ser configurado pelo Gerente do Sistema, onde o endereço do arquivo da base de dados, o endereço IP, o nome de usuário do banco e a senha devem ser informadas.

> *"Esse DataModule é necessário para que, quando ele criar uma nova instância do Remote DataModule, ele não crie também uma nova instância de conexão no servidor. Pois toda vez que um cliente se conecta em nosso servidor de aplicação, será criada uma instância do Remote DataModule na memória e, consecutivamente, uma instância de todos os objetos que estiverem nele. Se o SQLConnection, que é responsável pela conectar com a base de dados, estiver no Remote DataModule, será criada também uma nova instância dele e criando uma nova conexão com o banco. Por isso, separamos a conexão do banco em um DataModule."* (GAFFO, 2009).

O *Remote Data Module* ficará responsável pela conexão remota da aplicação servidora e disponibilizar os componentes de acesso ao banco de dados. Estes componentes estarão ligados ao componente de conexão presente no *datamodule* local.

A tela para configuração é aberta pelo Delphi, onde as configurações do servidor podem ser feitas. A tela de configuração está disponível nas figuras abaixo (GAFFO, 2009).

Figura 5 - Configurando o Remote Data Module

Figura 6 - Inserindo o Nome da CoClass

Quando um *Remote Data Module* é criado, ele gera um número identificador que, ao compilar a aplicação pela primeira vez, este número é adicionado ao registro do Windows.

Para questões de compatibilidade entre bancos de dados, os componentes utilizados são os componentes da guia *DbExpress*, estes componentes possuem grande versatilidade entre bancos de dados diferentes.

Em caso da base de dados alcançar o limite de espaço permitido pelo SGBD *Firebird* e a empresa necessitar migrar para um banco de dados *Oracle*, por exemplo, somente as configurações de acesso definidas no *Data module* precisarão ser alteradas, como a definição do banco de dados, *driver* fornecedor entre outras opções.

Na guia *DbExpress* do Delphi podemos encontrar vários componentes, entre eles o *TSQLConnection*, o *TSQLQuery* e o *TSQLDataset* (CANTU, 2006).

O *TSQLConnection* é o componente responsável por conectar a aplicação gerada pelo Delphi ao banco de dados. Este objeto é responsável pelas conexões ao banco de dados.

Para configurá-lo corretamente, um *driver* de conexão deve ser escolhido, cada *driver* de conexão se refere a um tipo de banco de dados. No caso do *Firebird*, o *driver* de conexão a ser selecionado é o *Interbase*. Os dois bancos de dados são similares, sendo que o *Interbase* é de propriedade da *Code Gear* e é pago. O *Firebird* é um banco de dados gratuito e pode ser baixado em sua versão completa de sites da *Internet*. Um exemplo de configuração de um objeto *TSQLConnection* esta disponível na figura abaixo.

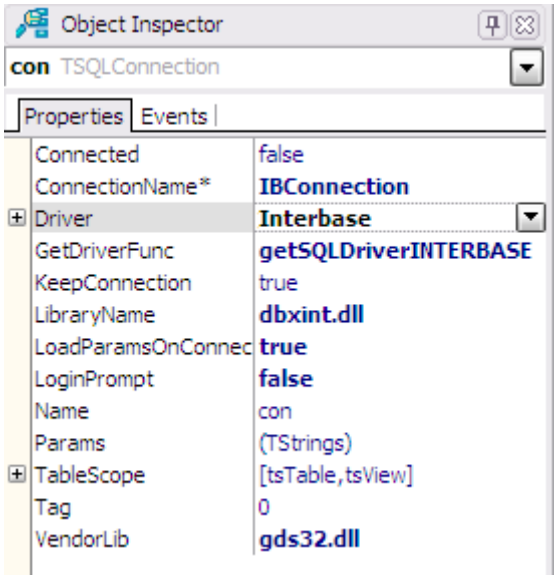

Figura 7 - Object Inspector do Objeto TSQLConnection

Os componentes *TSQLQuery* e *TSQLDataset* serão ligados ao componente *TSQLConnection* da aplicação, para assim poderem acessarem as tabelas da base de dados. Estes componentes são unidirecionais, ou seja, não suportam uma navegação nos registros selecionados pelo comando *select* do SQL.

Para pesquisas no banco de dados os componentes do tipo *TSQLQuery* serão utilizados. Estes componentes não suportam inserção, edição e exclusão através de comandos do Delphi e por este motivo serão utilizados na maioria das vezes somente para pesquisas.

Para as operações de inclusão, edição e exclusão de dados, serão utilizados componentes do tipo *TSQLDataset*, que permitem este tipo de operações através de comandos do Delphi.

Como o protótipo deve funcionar em rede, suas sentenças SQL possuem parâmetros para que somente o que a sentença SQL encontrar dentro destes parâmetros retorne ao cliente que requisitou os dados, evitando assim tráfego desnecessário na rede.

Na aplicação servidora também irão residir às validações de dados. Estas validações são as validações onde dados serão pesquisado no banco de dados, sendo que se um dado que deve ser único, como o CPF de um funcionário, por exemplo, encontrar uma referência no banco de dados, uma exceção será levantada no servidor. Esta exceção será retornada a aplicação cliente, onde será exibida ao usuário que efetuou a operação.

Para que esta troca de informações ocorra, cada componente de pesquisa e de manipulação de dados deverá estar ligado a um componente do tipo *TDataSetProvider* (GAFFO, 2009). Este componente é responsável por fornecer a aplicação cliente acesso ao seu componente de dados, seja um *TSQLQuery* ou um *TSQLDataset* (CANTU, 2006).

É neste componente que as regras de validação serão inseridas. Este componente também tem a função de passar quaisquer parâmetros definidos pelos componentes de acesso a dados, incluindo os campos selecionados por estes componentes.

A aplicação servidora também pode possuir métodos criados pelo desenvolvedor, sendo que estes métodos serão executados a partir da aplicação cliente. Para isso, um método deve ser declarado na *Type Library*, podendo ser uma *function* ou uma *procedure* e também possuindo parâmetros de entrada e/ou saída de dados. A *type library* da aplicação servidora possui os métodos mostrados na figura 12 do apêndice V.

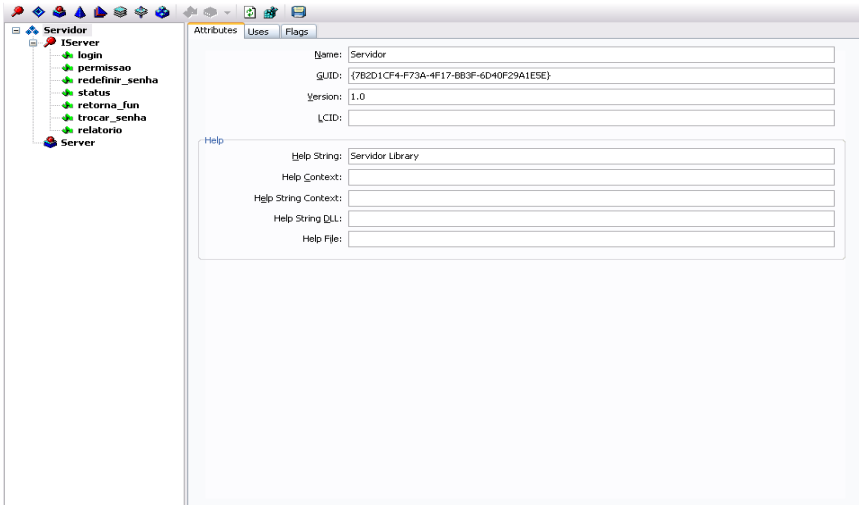

Figura 8 - Type Library da Aplicação Servidora

Para que as alterações realizadas na aplicação servidora sejam acessadas pela aplicação cliente, a aplicação servidora deve ser compilada, assim gerando uma atualização da aplicação servidora.

3. DESENVOLVENDO O PROTÓTIPO

3.1 Levantamento de Requisitos

Para criação do protótipo do sistema, regras e informações de um sistema realizado de forma manual serão usadas para desenvolvimento.

Para isso, as informações usadas no levantamento de requisitos serão fundamentais, tanto para o planejamento do software como na explanação da seqüência.

A empresa é uma mecânica de autos que vende autopeças e serviços a seus clientes. A empresa está localizada na Cidade de Jacarezinho e tem seu ramo de atividades voltado para a manutenção de veículos de pequeno porte.

A empresa requisitou um sistema que deve ter por função principal gerenciar uma mecânica de carros com autopeças onde, são vendidos ao consumidor os serviços e produtos de acordo com o diagnóstico feito previamente por um mecânico.

O sistema atual será abordado para uma maior compreensão de como sistema da empresa funciona em seu dia – a – dia, também serão abordados os problemas que levaram a empresa a requisitar um sistema. Também será apresentada uma proposta de criação de *software* para solucionar os problemas encontrados, mostrando também todos os requisitos que devem ser preenchidos para que o *software* atenda as necessidades da empresa.

3.1.1 O Sistema Atual

O sistema usado pela empresa não é um sistema automatizado, ou seja, todo o processo é feito manualmente.

Este sistema, além de ser ineficiente divide a empresa em pequenos serviços que encontram dificuldades de comunicação entre sim, tornando assim qualquer processo mais demorado e com informações imprecisas de alguma das partes.

Os serviços são:

- Controle de funcionários;
- Controle de estoque;
- Gerência / vendas;
- Gerenciamento de serviços;
- Controle de clientes.

3.1.2 Os Problemas Encontrados

Por ser um sistema manual, torna-se cansativo procurar arquivos que não ordenados ou não são atualizados com freqüência.

Da mesma maneira, o controle de estoque não gera uma requisição, isto é, o controle é feito de forma informal, não passando por nenhuma seqüência de eventos associados à organização de processos.

Foi constatado que a empresa não possui um controle de clientes gerenciado, o que é de grande importância no momento da venda e posterior cobrança em casos de falta de pagamento.

O sistema de controle de contas é feito através de um livro caixa no qual as contas de entrada e saída são cadastradas manualmente, anotando o número da nota fiscal gerada e os dados da conta.

O sistema de caixa, que é colocado juntamente com o sistema de contas é ineficaz, considerando que não existe um espaço físico onde as notas fiscais deveriam ficar armazenadas para futura consulta.

A empresa não possui uma listagem de seus serviços, nem descrição, nem valores dos mesmos, tornando assim o serviço incerto.

Também não foi encontrado um sistema sólido para gerenciamento dos funcionários da empresa, como função, carga horária e outros dados pertinentes para controle.

3.1.3 Proposta de Criação de Software

Para criação do sistema, foi gerado um planejamento de soluções baseado nos problemas encontrados e nas requisições do cliente.

O sistema seguirá o modelo incremental, ou seja, será desenvolvido por etapas bem definidas, sendo que ao final de cada etapa o sistema ganhará uma nova funcionalidade.

O sistema será dividido em seis incrementos, onde os dados do levantamento serão distribuídos, assim podendo ser trabalhados individualmente.

Estes incrementos são:

- Servidor de Aplicações;
- Controle de usuários;
- Controle de funcionários;
- Controle de estoque e serviços;
- Sistema de vendas;

Tendo por base o levantamento de dados, o sistema possuirá a capacidade de funcionamento em rede uma interna da empresa, assim deixando a possibilidade para uma possível expansão do sistema em uma subseqüente atualização.

3.1.4 Requisitos

Como descrito anteriormente, o sistema deve funcionar através de uma rede local, possibilitando assim setorizar os serviços da empresa.

O 1º incremento será o Servidor de Aplicações. Neste incremento, estarão às regras de negócios, como também será o sistema que terá a ligação com o banco de dados, assim formando as camadas de dados e regras de negócios.

O 2º incremento será o controle de usuários, esse sistema consiste em habilitar um funcionário já cadastrado no sistema a usar algum dos sistemas que a empresa possui. Esta função só será exercida pelo administrador do sistema.

O 3º incremento será o controle de funcionários, onde as informações sobre os funcionários da empresa ficaram armazenados, como nome, endereço e função associada para cada funcionário da empresa.

O 4º incremento será o controle de estoques e serviços. O controle de estoques consiste em cadastrar produtos que representarão os itens no estoque, seu tipo, seu fornecedor, valores de compra e venda, assim como os detalhes de cada item em separado.

Cada item em particular tem um número mínimo de quantidade que deve ser mantido em estoque, portanto, cada item em separado deve possuir a sua quantidade mínima em estoque, podendo assim oferecer ao cliente certa quantia.

Em caso do nível do estoque estar muito baixo, o sistema gera um alerta, este alerta é recebido pelo gerente, que ao constatar que um ou mais itens necessitam ser renovados, irá gerar uma requisição para cada um, indicando a quantidade a ser adquirida de cada item.

O controle de serviços tem como função cadastrar todos os serviços que a empresa oferece com detalhes, descrevendo o que cada um faz em específico, quantos mecânicos são necessários para o serviço e qual o custo do mesmo.

O 5º incremento é o controle de vendas e clientes, no qual haverá uma subdivisão.

Existem dois tipos de clientes básicos, o cliente físico e o cliente jurídico, para cada um será montado um tipo de cadastro, apenas discriminando os elementos comuns entre os dois, como endereço, bairro, entre outros.

No caso do cliente pessoas físicas serão abordados campos mais comuns, como nome e CPF, e no caso do cliente pessoa jurídica os elementos mudam, como exemplo a razão social, nome fantasia e o CNPJ (Cadastro Nacional de Pessoas Jurídicas). Como todos serão listados como clientes, todos poderão adquirir produtos e serviços a qualquer hora.

O sistema de vendas se valerá das informações contidas em sistemas anteriores, que são o de estoque, serviços, clientes e funcionários. Um cliente, nem sempre pode requisitar uma venda direta, mas sim um orçamento, onde as informações devem ser tratadas como as de uma venda, mesmo assim, um orçamento pode posteriormente se tornar uma venda, se solicitada pelo cliente.

Em caso de venda, direta ou indireta (através de orçamento), os dados da venda são salvos. Toda venda gera uma ou mais ordens de serviço.

A ordem de serviço gerada pela venda inicia o processo de execução dos pedidos do cliente, referente à venda. Uma venda também gera uma nota fiscal em que será impressa em duas vias, uma é armazenada em arquivo dentro da empresa e outra é entregue ao cliente, nesta nota fiscal, estão listados os itens e/ou serviços listados no diagnóstico do mecânico e aprovados pelo cliente.

Todos os incrementos irão gerar uma atualização no 1º incremento, o servidor de aplicações. Todas as chamadas remotas serão feitas sob demanda, para que o trafego na rede não seja tão intenso.

3.1.5 Casos de Uso

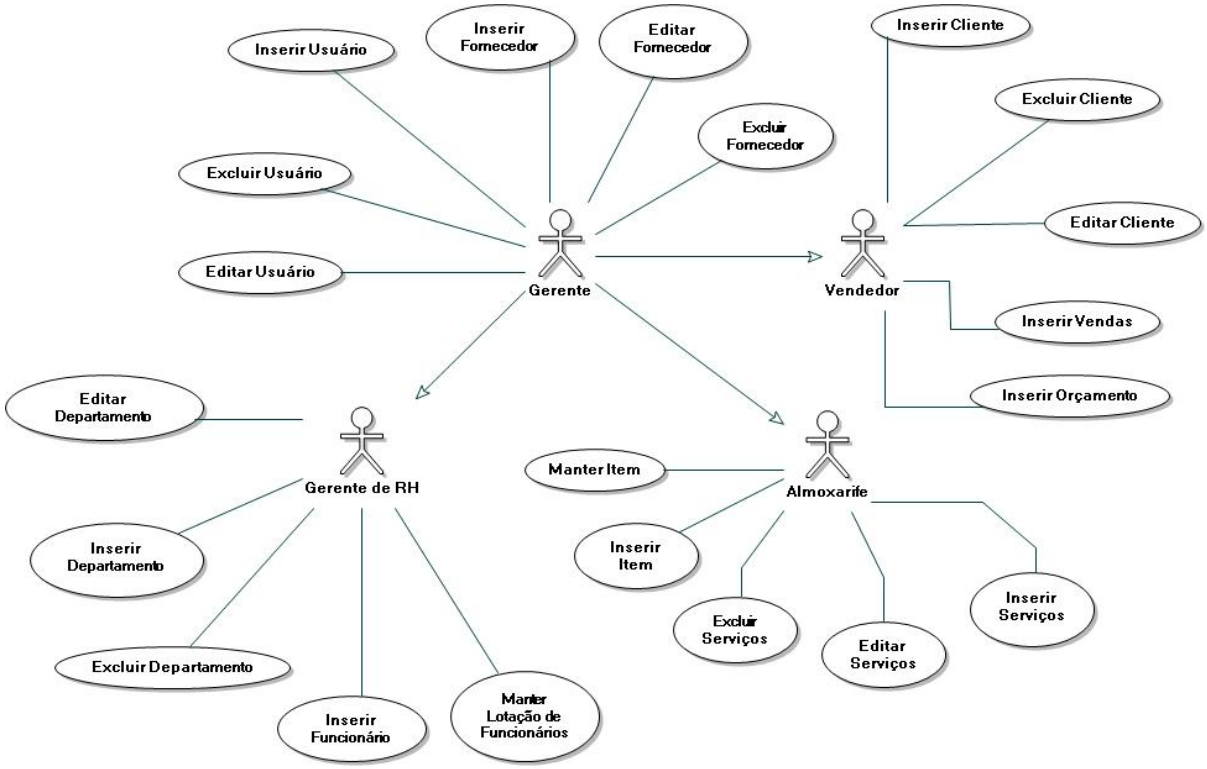

Figura 8 - Casos de Uso

A descrição dos casos de uso pode ser encontrada no Apêndice A do trabalho.

3.2 Desenvolvimento

O desenvolvimento do protótipo seguiu os incrementos definidos no levantamento de requisitos. Estes incrementos são:

- Servidor de Aplicações;
- Controle de Usuários;
- Controle de Funcionários;
- Controle de estoques e serviços;
- Sistema de Vendas;

O Servidor de Aplicações foi o primeiro incremento do sistema, nele residem todos os objetos de acesso a dados. Este incremento possui um *Datamodule*, um *RemoteDatamodule* e uma *Interface*.

O *Datamodule* é responsável pela conexão com o servidor de dados da aplicação, sendo que ele possui poucos métodos. O *RemoteDataModule* possui todos os componentes de acesso a dados e métodos remotos da aplicação, estes componentes estão ligados ao componente de conexão de dados do *DataModule*.

O *RemoteDatamodule* também é o responsável por implementar os métodos existentes na *Interface* da aplicação servidora, sendo que estes métodos serão utilizados pelos clientes remotos da aplicação.

É neste incremento que as tabelas do banco de dados são criadas e relacionadas, assim ficando disponíveis em todo o processo de desenvolvimento dos demais incrementos.

O controle de usuários foi o segundo incremento do sistema. Neste incremento a aplicação cliente é gerada. Neste incremento uma classe centralizadora é criada, onde as rotinas de manipulação de dados do lado cliente são geradas, assim como as rotinas de criação de formulários da aplicação cliente. Neste incremento também é implementado um *data module* para que os componentes de acesso remoto sejam configurados.

Este incremento implementa a pesquisa de usuários, permissões de usuários e perfis existentes para a aplicação, e sendo também responsável pelas implementações de inserção, edição e exclusão de usuários e permissões do sistema.

Para que a aplicação gere uma pesquisa, primeiro no Servidor de aplicações os componentes de pesquisa (*TSQLQuery)* e os componentes de acesso remoto (*TDataSetProvider*) devem ser adicionados ao *RemoteDataModule*, para que assim as sentenças sql's necessárias sejam configuradas e distribuídas para acesso das aplicações clientes. Também foram criados métodos remotos para que um usuário do sistema possa logar no sistema.

Este método de login é ativado a partir de cada aplicação cliente, sendo que, ao retornar algum erro, uma exceção é levantada e retornada para o usuário remoto. Caso contrário, outro método é ativado, para verificar se o usuário possui algum dos perfis pré – configurados no sistema, caso não possua nenhum, o sistema impedirá o login, caso contrário, as definições de usuário serão aplicadas na aplicação cliente, assim delimitando acesso a determinadas áreas do sistema.

Para que o incremento tenha todas as suas funcionalidades, na aplicação servidora dois componentes *de* manipulação de dados (*TSQLDataset*) foram inseridos, seguidos de dois componentes de acesso remoto. Estes componentes são os responsáveis por inserir, editar e excluir registros da base de dados.

As validações simples como campos vazios foram programadas na aplicação cliente, isso se deve ao fato que ao aplicar o comando *post* do *dataset*, ele executa de maneira local este comando, assim possibilitando que parte do processamento ainda resida na aplicação cliente. Quando a validação dos dados requeria pesquisa no banco de dados da aplicação, esta validação era programada na camada de negócios ou Servidor de aplicação.

Para isso, o comando usado pela aplicação cliente para enviar os dados para validação é o comando *ApplyUpdates*. Uma vez enviados os dados, eles passavam pelo processo de validação, diferenciando se a ação era uma inserção ou edição.

No caso da exclusão, todos os dados necessários para a exclusão eram enviados para o Servidor de Aplicações, sendo que uma validação em forma de questionamento ao usuário é feita na aplicação cliente e, em caso de resposta positiva a exclusão era efetivada.

O controle de funcionários foi o terceiro incremento do sistema. Este incremento implementa a pesquisa de departamentos, funcionários e lotação de funcionários, sendo também responsável pelas implementações de inserção, edição e exclusão das respectivas funcionalidades.

Assim como ocorreu no primeiro incremento, para que a aplicação gere uma pesquisa, primeiro no Servidor de aplicações os componentes de pesquisa (*TSQLQuery)* e os componentes de acesso remoto (*TDataSetProvider*) devem ser adicionados ao *RemoteDataModule*, para que assim as sentenças sql's necessárias sejam configuradas e distribuídas para acesso das aplicações clientes.

Para que o incremento tenha todas as suas funcionalidades, na aplicação servidora três componentes *de* manipulação de dados (*TSQLDataset*) foram inseridos, seguidos de três componentes de acesso remoto.

Estes componentes são os responsáveis por inserir, editar e excluir registros da base de dados. As validações simples como campos vazios foram programadas na aplicação cliente, isso se deve ao fato que ao aplicar o comando *post* do *dataset*, ele executa de maneira local este comando, assim possibilitando que parte do processamento ainda resida na aplicação cliente.

Quando a validação dos dados requeria pesquisa no banco de dados da aplicação, esta validação era programada na camada de negócios ou Servidor de aplicação. Para isso, o comando usado pela aplicação cliente para enviar os dados para validação é o comando *ApplyUpdates*.

Uma vez enviados os dados, eles passavam pelo processo de validação, diferenciando se a ação era uma inserção ou edição. No caso da exclusão, todos os dados necessários para a exclusão eram enviados para o Servidor de Aplicações, sendo que uma validação em forma de questionamento ao usuário é feita na aplicação cliente e, em caso de resposta positiva a exclusão era efetivada.

Os controles de estoques e serviços foram o quarto incremento do sistema. Este incremento implementa a pesquisa de itens, movimentação de itens, fornecedores e serviços, sendo também responsável pelas implementações de inserção, edição e exclusão das respectivas funcionalidades.

Assim como ocorreu em incrementos anteriores, para que a aplicação gere uma pesquisa, primeiro no Servidor de aplicações os componentes de pesquisa (*TSQLQuery)* e os componentes de acesso remoto (*TDataSetProvider*) devem ser

adicionados ao *RemoteDataModule*, para que assim as sentenças sql's necessárias sejam configuradas e distribuídas para acesso das aplicações clientes.

Para que o incremento tenha todas as suas funcionalidades, na aplicação servidora foram adicionados um componente de manipulação de dados (*TSQLDataset*) e um componente de acesso remoto para cada tabela.

Estes componentes são os responsáveis por inserir, editar e excluir registros da base de dados. As validações simples como campos vazios foram programadas na aplicação cliente, isso se deve ao fato que ao aplicar o comando *post* do *dataset*, ele executa de maneira local este comando, assim possibilitando que parte do processamento ainda resida na aplicação cliente.

Quando a validação dos dados requeria pesquisa no banco de dados da aplicação, esta validação era programada na camada de negócios ou Servidor de aplicação. Para isso, o comando usado pela aplicação cliente para enviar os dados para validação é o comando *ApplyUpdates*.

Uma vez enviados os dados, eles passavam pelo processo de validação, diferenciando se a ação era uma inserção ou edição. No caso da exclusão, todos os dados necessários para a exclusão eram enviados para o Servidor de Aplicações, sendo que uma validação em forma de questionamento ao usuário é feita na aplicação cliente e, em caso de resposta positiva a exclusão era efetivada.

O sistema de vendas foi o quinto incremento do sistema. Este incremento implementa a pesquisa de clientes físicos, clientes jurídicos, orçamentos e vendas, sendo também responsável pelas implementações de inserção, edição e exclusão das respectivas funcionalidades.

Assim como ocorreu em incrementos anteriores, para que a aplicação gere uma pesquisa, primeiro no Servidor de aplicações os componentes de pesquisa (*TSQLQuery)* e os componentes de acesso remoto (*TDataSetProvider*) devem ser adicionados ao *RemoteDataModule*, para que assim as sentenças sql's necessárias sejam configuradas e distribuídas para acesso das aplicações clientes.

Para que o incremento tenha todas as suas funcionalidades, na aplicação servidora foram adicionados um componente de manipulação de dados (*TSQLDataset*) e um componente de acesso remoto para cada tabela. Estes componentes são os responsáveis por inserir, editar e excluir registros da base de dados.

As validações simples como campos vazios foram programadas na aplicação cliente, isso se deve ao fato que ao aplicar o comando *post* do *dataset*, ele executa de maneira local este comando, assim possibilitando que parte do processamento ainda resida na aplicação cliente.

Quando a validação dos dados requeria pesquisa no banco de dados da aplicação, esta validação era programada na camada de negócios ou Servidor de aplicação.

Para isso, o comando usado pela aplicação cliente para enviar os dados para validação é o comando *ApplyUpdates*. Uma vez enviados os dados, eles passavam pelo processo de validação, diferenciando se a ação era uma inserção ou edição.

No caso da exclusão, todos os dados necessários para a exclusão eram enviados para o Servidor de Aplicações, sendo que uma validação em forma de questionamento ao usuário é feita na aplicação cliente e, em caso de resposta positiva a exclusão era efetivada.

O desenvolvimento caminhou de acordo com o cronograma pré – estabelecido para o trabalho, sem deixar de lado nenhum dos casos de usos identificados pelo levantamento de requisitos.

O sistema ficou disposto de acordo com as figuras 9,10, 11 e 12.

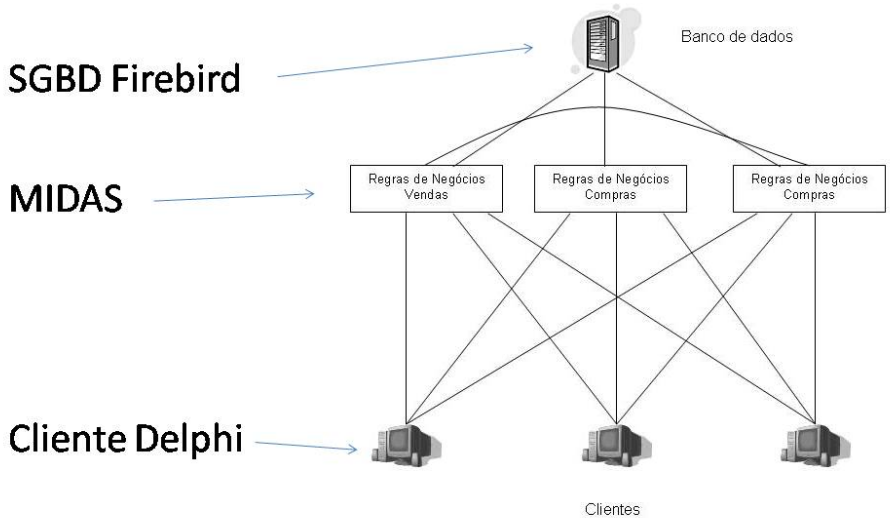

Figura 9 - Elementos Reais do Sistema x Arquitetura de 3 Camadas

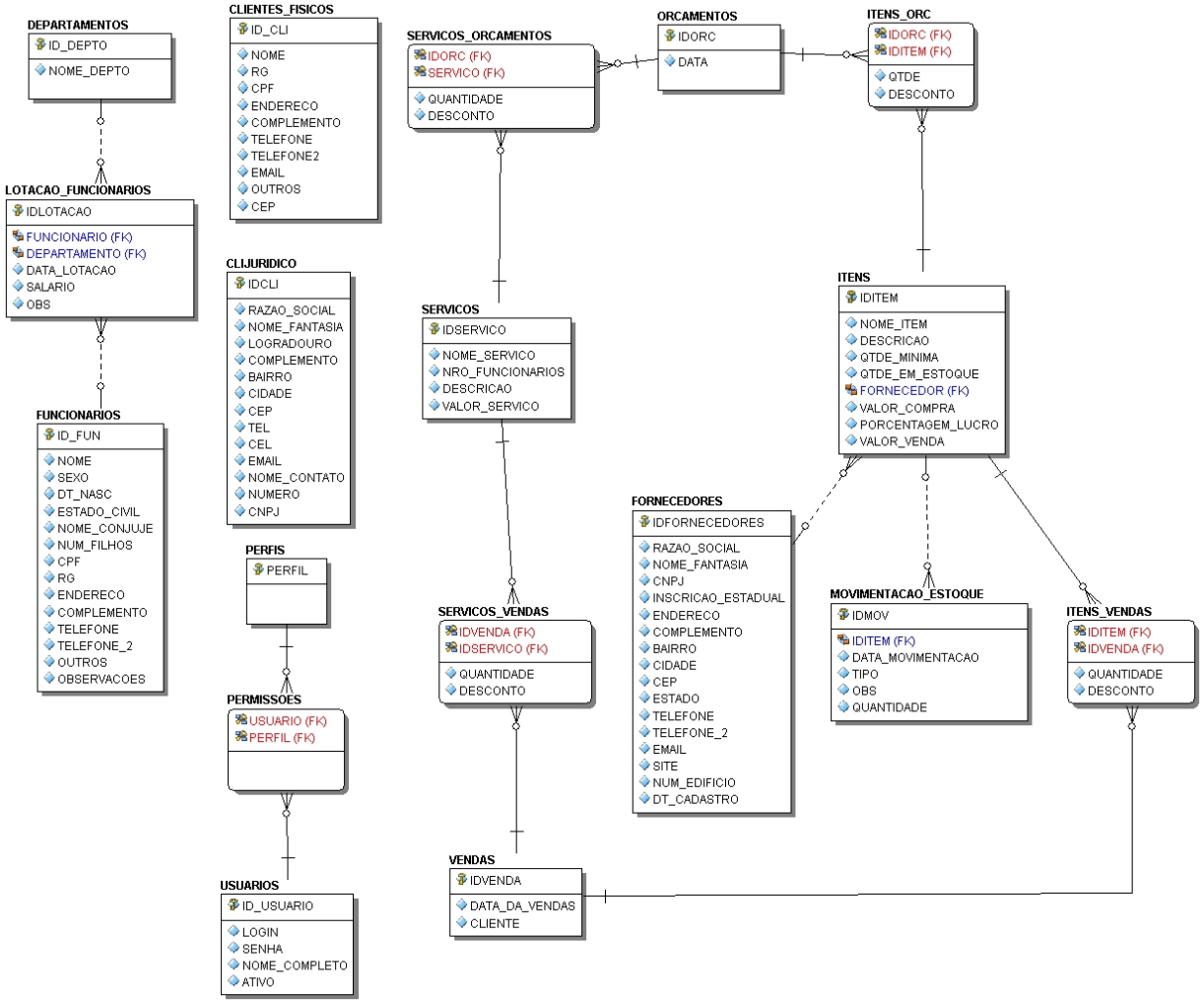

Figura 10 - Camada de Dados: Diagrama de Entidade / Relacionamento

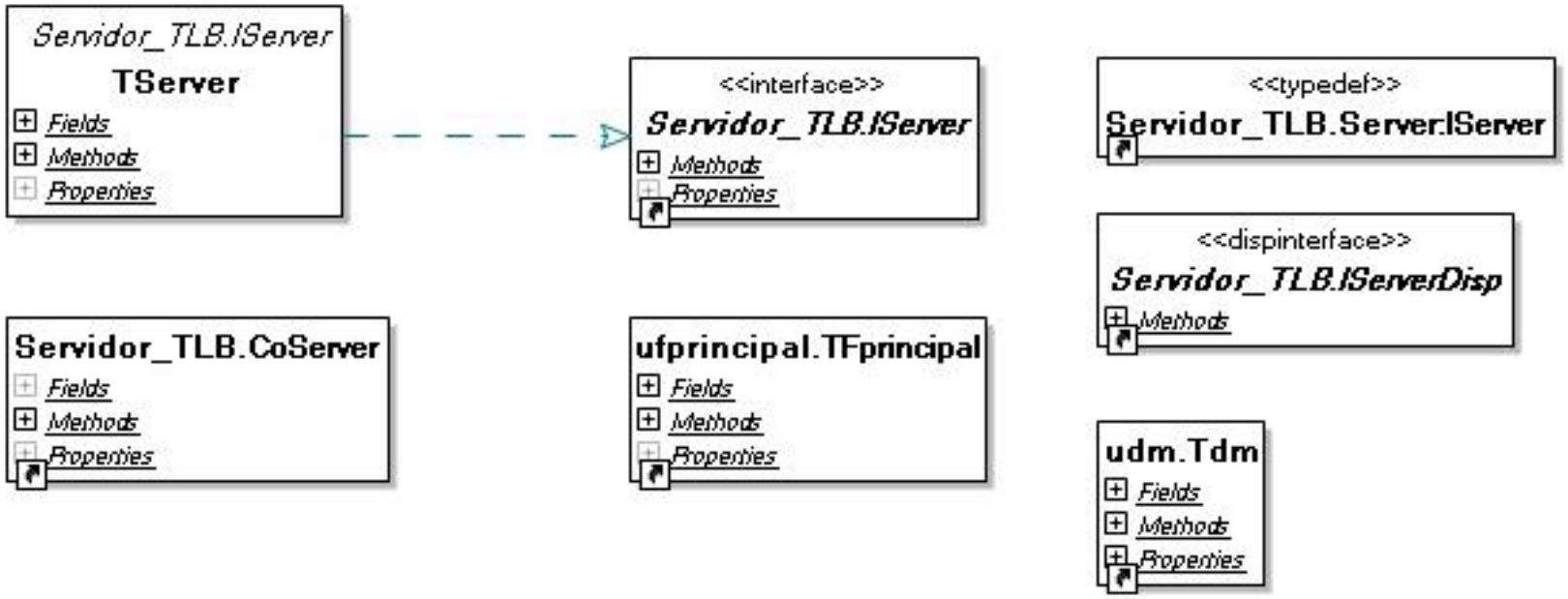

Figura 11 - Camada de Regras de Negócio: Classes da Aplicação Servidora

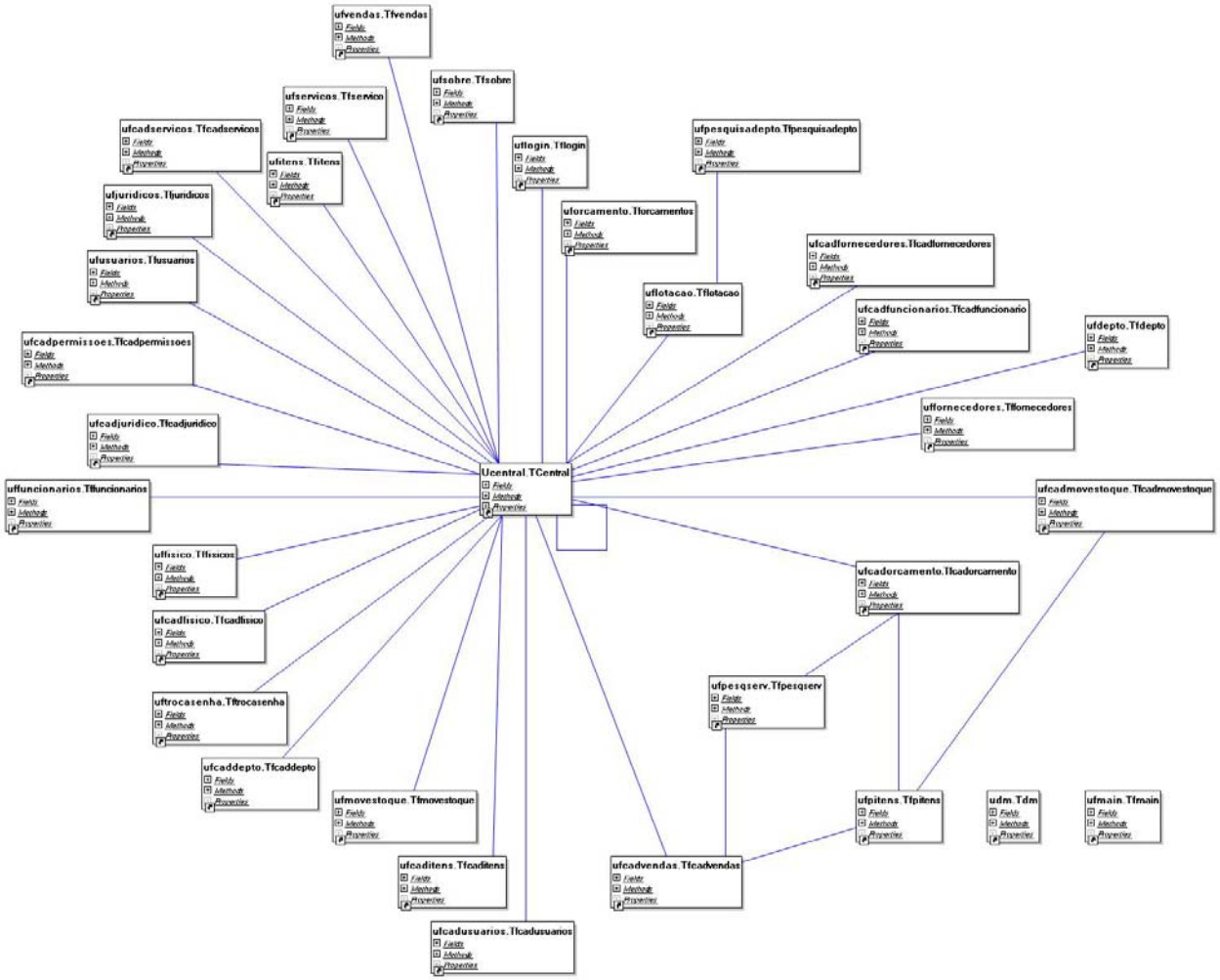

Figura 12 - Camada de Apresentação: Classes da Aplicação Cliente

4. CONCLUSÃO

Este trabalho teve como objetivo o desenvolvimento de um protótipo de um Sistema Gerencial para Autopeças e Auto Mecânica sob o aspecto de uma aplicação de múltiplas camadas. Através dele foi verificada a viabilidade de se criar um projeto para uma arquitetura de três camadas descrita por Pacheco (2005) e Cantu (2006).

A partir da proposta de criação de um sistema de arquitetura de três camadas, o trabalho mostrou bons resultados na implementação das três camadas da aplicação.

Os destaques estão na comunicação das aplicações em um ambiente de rede local, onde uma das máquinas conectadas a rede serviu como servidor de dados, outra como servidor de aplicações e outras duas máquinas como as máquinas clientes da rede, onde cada máquina cliente da rede se conectava somente a máquina servidora de aplicações e na atualização dos métodos declarados no servidor de aplicação para a visualização pelas aplicações clientes.

A cada atualização dos métodos e validações feitas na aplicação servidora, a mesma era compilada e automaticamente disponibilizava para a aplicação cliente a atualização, sem a necessidade de redistribuir a aplicação cliente como um todo.

Houve certa dificuldade na execução de relatórios na aplicação servidora. Os relatórios deveriam estar na aplicação servidora para um caso de atualizações futuras, as atualizações ocorressem no servidor de aplicações, alterando os relatórios presentes no servidor e não realizando uma nova compilação e distribuição do software para as estações clientes.

Verificou-se como principal dificuldade para o desenvolvimento deste projeto, o levantamento bibliográfico, devido à escassez de documentação especializada sobre como documentar diagramas da UML em um projeto de arquitetura de três camadas, bem como a forma correta para distribuição da aplicação, compartilhamento de arquivos e recursos de rede necessários para a utilização correta deste tipo de sistema.

Este trabalho apresentou como uma aplicação de três camadas pode ser desenvolvida, de forma viável, partindo de conhecimentos comuns a qualquer programador Delphi.

O aprimoramento deste tipo de *software* é necessário, pois cada vez mais empresas de médio a grande porte necessitam de sistemas distribuídos e modularizados para descentralizar ações em um mesmo sistema.

5. REFERÊNCIAS

BOOCH, Grady. UML: guia do Usuário / Grady Booch, James Rumbaugh, Ivar Jacobson - Tradução: Fábio Freitas da Silva e Cristina de Amorim Machado – 2ª Edição - Rio de Janeiro: Elsevier, 2005.

SOMMERVILLE, Ian. Engenharia de Software – 6ª Edição - São Paulo: Pearson Prentice Hall, 2003.

PACHECO, Xavier. Guia do desenvolvedor de Delphi for .NET – Tradução: Sandra Figueiredo e Carlos Schafranski – São Paulo: Pearson Makron Books, 2005.

MACORATTI, José Carlos. UML – Diagrama de Classes e Objetos – Disponível em: http://www.macoratti.net/net_uml1.htm - Acesso em: 10 de Janeiro de 2009.

DEBONI, José Eduardo Zindel – Introdução aos Diagramas da UML – Disponível em: http://www.voxxel.com.br/pages/introdiauml.html - Acesso em: 05 de Janeiro de 2009.

AMARAL, Fernando – Introdução a Diagramas UML – Disponível em: http://www.fernandoamaral.com.br/Default.aspx?Artigo=54 – Acesso em: 9 de Janeiro de 2009.

SANTOS, Misael – Padrões de Software – Disponível em: http://knol.google.com/k/misael-santos/padres-de-software/irtziatj2kf9/2# - Acesso em: Três de Fevereiro de 2009.

CANTU, Marco – Dominando o Delphi 2005 – A Bíblia - 1ª Edição - São Paulo: Pearson Prentice Hall, 2006.

GAFFO, Roberto – Aplicação Multi-Camadas – Parte 02 – Disponível em: http://imasters.uol.com.br/artigo/4360/delphi/aplicacao_multi-camadas_-_parte_02/ - Acesso em: 25 de maio de 2009.

APÊNDICES

APÊNDICE A – DESCRIÇÃO DOS CASOS DE USO

Na análise do sistema foram encontrados vinte e um casos de uso. O diagrama pode ser encontrado na Figura 5 do Apêndice I. Estes casos de uso são:

- Inserir Fornecedor;
- Inserir Usuário;
- Editar usuário;
- Excluir usuário;
- Excluir Fornecedor;
- Editar Fornecedor;
- Editar Cliente;
- Excluir Cliente;
- Inserir Cliente;
- Inserir Venda;
- Inserir Orçamento;
- Inserir Serviços;
- Editar Serviços;
- Excluir Serviços;
- Inserir Item;
- Manter item;
- Inserir Funcionário;
- Manter lotação de funcionários;

- Inserir departamento;
- Editar departamento;
- Excluir departamento.

Para estes casos de uso, alguns atores foram definidos no levantamento de requisitos, eles são:

- Vendedor;
- Almoxarife;
- Gerente de RH;
- Gerente.

Caso de Uso: Inserir Fornecedor

- Descrição: insere um fornecedor na base de dados;
- Fluxo de Eventos Principal: O Gerente insere os dados do fornecedor e salva a operação:
- Fluxo Excepcional de Eventos: Gerente cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Usuário

- Descrição: insere um usuário na base de dados;
- Fluxo de Eventos Principal: O Gerente insere os dados do usuário e salva a operação:

- Fluxo Excepcional de Eventos: Gerente cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Departamento

- Descrição: insere um departamento na base de dados;
- Fluxo de Eventos Principal: O Gerente de RH insere os dados do departamento e salva a operação:
- Fluxo Excepcional de Eventos: Gerente de RH cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Funcionário

- Descrição: insere um funcionário na base de dados;
- Fluxo de Eventos Principal: O Gerente de RH insere os dados do funcionário e salva a operação:
- Fluxo Excepcional de Eventos: Gerente de RH cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Item

- Descrição: insere um item na base de dados;
- Fluxo de Eventos Principal: O Almoxarife insere os dados do item e salva a operação:
- Fluxo Excepcional de Eventos: Almoxarife cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Serviços

- Descrição: insere um serviço na base de dados;
- Fluxo de Eventos Principal: O Almoxarife insere os dados do serviço e salva a operação:
- Fluxo Excepcional de Eventos: Almoxarife cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Orçamento

- Descrição: insere um orçamento na base de dados;
- Fluxo de Eventos Principal: O Vendedor insere os dados do orçamento e salva a operação:
- Fluxo Excepcional de Eventos: Vendedor cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Vendas

- Descrição: insere uma venda na base de dados;
- Fluxo de Eventos Principal: O Vendedor insere os dados da venda e salva a operação:
- Fluxo Excepcional de Eventos: Vendedor cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Inserir Cliente

- Descrição: cadastra um cliente na base de dados;
- Fluxo de Eventos Principal: O Vendedor insere os dados do cliente e salva a operação:
- Fluxo Excepcional de Eventos: Vendedor cancela a inserção.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Editar Fornecedor

- Pré – Condição: que ao menos um fornecedor seja retornado de uma pesquisa prévia.
- Descrição: Edita um fornecedor existente na base de dados.

- Fluxo de Eventos Principal: O Gerente edita os dados do fornecedor e salva a operação.
- Fluxo Excepcional de Eventos: Gerente cancela a edição.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Editar Usuário

- Pré – Condição: que ao menos um usuário seja retornado de uma pesquisa prévia.
- Descrição: Edita um usuário existente na base de dados.
- Fluxo de Eventos Principal: O Gerente edita os dados do usuário e salva a operação.
- Fluxo Excepcional de Eventos: Gerente cancela a edição.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Editar Departamento

- Pré – Condição: que ao menos um departamento seja retornado de uma pesquisa prévia.
- Descrição: Edita um departamento existente na base de dados.
- Fluxo de Eventos Principal: O Gerente de RH edita os dados do departamento e salva a operação.
- Fluxo Excepcional de Eventos: Gerente de RH cancela a edição.

- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Manter Lotação de Funcionários

- Pré – Condição: que ao menos um departamento e um funcionário estejam cadastrados na base de dados.
- Descrição: Mantém um funcionário lotado em um departamento ou lota o funcionário em outro departamento.
- Fluxo de Eventos Principal: O Gerente de RH edita os dados da lotação e salva a operação.
- Fluxo Excepcional de Eventos: Gerente de RH cancela a edição.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Editar Serviços

- Pré – Condição: que ao menos um serviço seja retornado de uma pesquisa prévia.
- Descrição: Edita um serviço existente na base de dados.
- Fluxo de Eventos Principal: O Almoxarife edita os dados do serviço e salva a operação.
- Fluxo Excepcional de Eventos: Almoxarife cancela a edição.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Editar Cliente

- Pré – Condição: que ao menos um cliente seja retornado de uma pesquisa prévia.
- Descrição: Edita um cliente existente na base de dados.
- Fluxo de Eventos Principal: O Vendedor edita os dados do cliente e salva a operação.
- Fluxo Excepcional de Eventos: Vendedor cancela a edição.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Que os dados passem sem erros pela validação.

Caso de Uso: Excluir Cliente

- Pré – Condição: que ao menos um cliente seja retornado de uma pesquisa prévia.
- Descrição: Exclui um cliente existente na base de dados.
- Fluxo de Eventos Principal: O Vendedor exclui um cliente da base de dados.
- Fluxo Excepcional de Eventos: Vendedor cancela a edição.
- Pós - Condição: Vendedor confirme a exclusão.

Caso de Uso: Excluir Serviços

- Pré – Condição: que ao menos um serviço seja retornado de uma pesquisa prévia.
- Descrição: Exclui um serviço existente na base de dados.
- Fluxo de Eventos Principal: O Almoxarife exclui um cliente da base de dados.
- Fluxo Excepcional de Eventos: Almoxarife cancela a edição.
- Pós - Condição: Almoxarife confirme a exclusão.

Caso de Uso: Excluir Departamento

- Pré – Condição: que ao menos um departamento seja retornado de uma pesquisa prévia.
- Descrição: Exclui um departamento existente na base de dados.
- Fluxo de Eventos Principal: O Gerente de RH exclui um cliente da base de dados.
- Fluxo Excepcional de Eventos: Gerente de RH cancela a edição.
- Pós - Condição: Gerente de RH confirme a exclusão.

Caso de Uso: Excluir Fornecedor

- Pré – Condição: que ao menos um fornecedor seja retornado de uma pesquisa prévia.
- Descrição: Exclui um fornecedor existente na base de dados.
- Fluxo de Eventos Principal: O Gerente exclui um fornecedor da base de dados.

- Fluxo Excepcional de Eventos: Gerente cancela a edição.
- Pós - Condição: Gerente confirme a exclusão.

Caso de Uso: Excluir Usuário

- Pré – Condição: que ao menos um usuário seja retornado de uma pesquisa prévia.
- Descrição: Exclui um usuário existente na base de dados.
- Fluxo de Eventos Principal: O Gerente exclui um usuário da base de dados.
- Fluxo Excepcional de Eventos: Gerente cancela a edição.
- Pós - Condição: Gerente confirme a exclusão.

Caso de Uso: Manter Item

- Pré – Condição: que ao menos um item esteja cadastrado.
- Descrição: Almoxarife mantém dados do item atualizado.
- Fluxo de Eventos Principal: O Almoxarife mantém os dados atualizados.
- Fluxo Excepcional de Eventos: Almoxarife cancela a edição.
- Fluxo Excepcional de Eventos: Dados não são válidos.
- Pós - Condição: Passe sem erros pela validação.

Descrição dos Atores

Ator: Gerente de RH

O Gerente de RH é a pessoa responsável pelo controle de departamentos e o controle de funcionários da empresa. Todas as operações destes incrementos devem ser executadas por este ator.

Ator: Almoxarife

O Almoxarife é a pessoa responsável pelo controle de estoques e o controle de serviços da empresa. Todas as operações destes incrementos devem ser executadas por este ator.

Ator: Vendedor

O Vendedor é a pessoa responsável pelo controle de clientes e o controle de vendas da empresa. Todas as operações destes incrementos devem ser executadas por este ator.

Ator: Gerente

O Gerente é a pessoa responsável pelo controle de fornecedores, controle de usuários e por todas as ações realizadas pelos outros atores. Este ator tem acesso a todas as áreas do sistema de forma irrestrita.

APÊNDICE B – QUADROS COM CODIFICAÇÃO DA APLICAÇÃO CLIENTE

```
function TCentral.Salvar(cds: tclientdataset): boolean;
begin
  if cds.State in [dsinsert,dsedit] then
    cds.post;
  if cds.ApplyUpdates(0) = 0 then
    begin
      result:=true;
      inserindo:=false;
    end else
      begin
        result:=false;
        cds.edit;
      end;
  dm.con.close;
end;
```

Quadro 1 - Chamada Remota de um Dataset

```
procedure TCentral.Login(const login,senha: string);
begin
  dm.con.open;
  try
    dm.con.AppServer.login(login,senha);
    permissoes(login);
    PUsername:=login;
  finally
    dm.con.close;
  end;
end;
```

Quadro 2 - Chamada de Método Remoto