



Centro Universitário do Sul de Minas – UNIS-MG

Bacharelado em Ciência da Computação

Aplicação Comercial com Ênfase em Usabilidade

Hélio Lemes Costa Jr.

**Guilherme Lourenção Tempesta
Júlio César Oliveira Flório**

Varginha, 2008



Centro Universitário do Sul de Minas – UNIS-MG

Bacharelado em Ciência da Computação

Aplicação Comercial com Ênfase em Usabilidade

Projeto de Conclusão de Curso
apresentado ao programa do curso
de Bacharelado em Ciência da
Computação do Centro Universitário do
Sul de Minas, como requisito parcial
para a obtenção do título de Bacharel
em Ciência da Computação

Varginha, 2008

FOLHA DE APROVAÇÃO

**Guilherme Lourenção Tempesta
Júlio César Oliveira Flório**

Aplicação Comercial com Ênfase em Usabilidade

Monografia apresentada ao curso de Ciência da Computação do Centro Universitário do Sul de Minas – UNIS/MG, como pré-requisito para obtenção do grau de bacharel pela Banca Examinadora composta pelos membros:

() Aprovado

() Reprovado

Data 25/11/2008

Profº Especialista Hélio Lemes Costa Jr.

Profº. Especialista Fabrício Pelloso Piurcosky

Profº. Especialista Lázaro Eduardo da Silva

OBS.:

Dedicamos este projeto a todos os que nos ajudaram direta e indiretamente durante todo o seu desenvolvimento, e a nós mesmos, que nos esforçamos ao máximo para concluir mais esta etapa.

Agradecimentos

Agradecemos primeiramente a Deus, por ter nos dado força e perseverança para concluir este projeto, aos nossos familiares, pelo incentivo, compreensão e companheirismo e ao nosso orientador, coordenador e colegas de trabalho por tirar dúvidas e procurar nos ajudar nas etapas mais críticas.

RESUMO

Nesta monografia é descrita a criação de um protótipo de uma aplicação comercial, cujo foco principal é o conceito de Usabilidade, onde a interação entre usuário e sistema é tratada como o ponto mais importante do desenvolvimento.

O projeto consiste na utilização deste protótipo especificamente em uma vídeo-locadora, sendo que deve ser disponibilizado em quiosques dentro do próprio estabelecimento, para que os clientes da locadora possam ter acesso a ele, entrando com sua identificação e efetuando vários tipos de consultas de filmes, podendo eles próprios efetuar locações e reservas, com a utilização de recursos multimídia como a apresentação de trailers. Tudo isso de uma forma independente, sem necessitar de ajuda do atendente para lhe apresentar os filmes.

No desenvolvimento do projeto, foram utilizados conhecimentos adquiridos nas áreas de Engenharia de Software, que auxiliou na organização da parte estrutural, UML, que possibilitou uma visão panorâmica de como modelar o sistema; programação, utilizando ferramentas para a implementação do software; Banco de Dados, utilizando a linguagem SQL através de comandos que facilitam o gerenciamento das informações; e Interface Homem-Máquina, que é o principal foco, procurando interagir com o usuário da melhor forma possível.

Lista de Figuras

FIGURA 1 – DIVERGÊNCIAS ENTRE ENGENHARIA DE SOFTWARE E IHM	17
FIGURA 2 - CICLO DE VIDA CLÁSSICO (FONTE: DCA, 2008).....	19
FIGURA 3 - ORIENTAÇÃO À OBJETO (FONTE: AUTOR).....	24
FIGURA 4 - APLICAÇÃO DE UM DIAGRAMA E-R (FONTE: PONTES, 2008).....	25
FIGURA 5 - DIAGRAMA DE CASO DE USO (FONTE: DSC, 2008).....	27
FIGURA 6 - DIAGRAMA DE CLASSE (FONTE: VOXXEL, 2008).....	28
FIGURA 7 - ASSOCIAÇÕES (FONTE: AUTOR).....	29
FIGURA 8 - DIAGRAMA DE ATIVIDADE (FONTE: LINHA, 2008).....	31
FIGURA 9 - MODELO EM REDE (FONTE: KORTH E SILBERSCHATZ, 1995)	32
FIGURA 10 - MODELO HIERÁRQUICO (FONTE: DEVMEDIA, 2008)	32
FIGURA 11 - NÍVEIS DE ABSTRAÇÃO (FONTE: KORTH E SILBERSCHATZ, 1995)	33
FIGURA 12 - TIPOS DE CARDINALIDADE (FONTE: AUTOR)	34
FIGURA 13 - DIAGRAMA E-R (FONTE: KORTH E SILBERSCHATZ, 1995).....	35
FIGURA 14 - TABELAS DE UM BD RELACIONAL (FONTE: ENG, 2008)	35
FIGURA 15 - <i>FORM</i> CRIADO NO DELPHI (FONTE: AUTOR).....	45
FIGURA 16 - CODE EDITOR (FONTE: AUTOR)	46
FIGURA 17 - COMPONENT PALETTE (FONTE: AUTOR)	46
FIGURA 18 - OBJECT TREEVIEW E OBJECT INSPECTOR (FONTE: AUTOR).....	47
FIGURA 19 - CONEXÃO NO SQLCONNECTION (FONTE: AUTOR)	48
FIGURA 20 - DATAMODULE E OS COMPONENTES DE CONEXÃO (FONTE: AUTOR)	48
FIGURA 21 - ADICIONANDO CAMPOS NO CLIENTDATASET (FONTE: AUTOR).....	49
FIGURA 22 - LIGAÇÃO NO DATASOURCE (FONTE: AUTOR)	49
FIGURA 23 - MODELO DE PROTOTIPAGEM (FONTE: PRESSMAN, 1995).....	50
FIGURA 24 - VISUAL PARADIGM COMMUNITY EDITION.	52
FIGURA 25 - DIAGRAMA DE CASO DE USO	53
FIGURA 26 - DIAGRAMA DE CLASSE.....	54
FIGURA 27 - DIAGRAMA DE ATIVIDADE.....	55
FIGURA 28 - INTERFACE DO IBEXPERT	56
FIGURA 29 - DOMÍNIOS CRIADOS NO BANCO	57
FIGURA 30 - TABELAS E RELACIONAMENTOS DO BD	58
FIGURA 31 – DATAMODULE E DEMAIS COMPONENTES DE CONEXÃO	62
FIGURA 32 - TELA DE BUSCA.....	64
FIGURA 33 - TELA DE DETALHES DO FILME	66
FIGURA 34 - EXIBIÇÃO DA SINOPSE DO FILME	67
FIGURA 35 - PLAYER QUE EXECUTA OS TRAILERS	68
FIGURA 36 - RESERVA DO FILME	69
FIGURA 37 - TELA DE LOCAÇÃO.....	70

Sumário

RESUMO	6
1 INTRODUÇÃO.....	10
1.1 OBJETIVOS	10
1.1.1 Geral.....	10
1.1.2 Específico	10
1.2 JUSTIFICATIVA	11
1.3 ORGANIZAÇÃO DA MONOGRAFIA	11
2 REFERENCIAL TEÓRICO.....	12
2.1 AUTOMAÇÃO COMERCIAL.....	12
2.1.1 Benefícios e Dificuldades	12
2.1.2 Contexto Atual.....	12
2.2 INTERFACE HOMEM-MÁQUINA	14
2.2.1 O Computador na Relação homem-máquina	14
2.2.2 Definição e importância das interfaces	14
2.2.3 Fatores que influenciam a construção de interfaces	15
2.2.4 Objetivos de IHM.....	15
2.2.5 Desafios de IHM.....	15
2.2.6 Usabilidade.....	16
2.2.7 Ergonomia	16
2.2.8 Principais diferenças entre IHM e Engenharia de software	17
2.3 ENGENHARIA DE SOFTWARE	18
2.3.1 Importância do software.....	18
2.3.2 Aplicações do Software	19
2.3.3 Ciclo de Vida Clássico da Engenharia de Software.....	19
2.3.3.1 Análise e Engenharia de Sistemas	20
2.3.3.2 Análise de Requisitos de Software	20
2.3.3.3 Projeto	20
2.3.3.4 Codificação	20
2.3.3.5 Testes	20
2.3.3.6 Manutenção	20
2.3.4 O Processo de Gerencia de Projetos.....	21
2.3.5 Administração de Projetos	21
2.3.6 Objetivos do Planejamento de Projetos.....	22
2.3.7 Modelos de Processo	22
2.3.8 Escopo do Software	22
2.3.9 Software Comercial	22
2.3.10 Importância do Software em Aplicação Comercial.....	23
2.3.11 Análise Orientada a objeto.....	23
2.3.11.1 Conceitos da Orientação a Objetos.....	23
2.3.11.2 Identificação de Objetos.....	24
2.3.11.3 Especificação de atributos.....	25
2.3.12 Modelagem de Dados	25
2.3.12.1 Diagramas Entidade-Relacionamento	25
2.4 UNIFIED MODELING LANGUAGE	26
2.4.1 Diagramas da UML.....	26
2.4.1.1 Diagramas de caso de uso	27
2.4.1.2 Diagramas de Classe	28
2.4.1.3 Diagrama de atividade.....	30
2.5 BANCO DE DADOS	32
2.5.1 Principais Características de um BD.....	33
2.5.2 Abstração de dados	33
2.5.3 Modelagem de Dados	34
2.5.4 Modelagem Lógica.....	35
2.5.4.1 Banco de dados relacional.....	35
2.5.4.2 Álgebra Relacional.....	36
2.5.5 Structured Query Language (SQL).....	36
2.5.5.1 Views	37

2.5.5.2	Generators	37
2.5.5.3	Triggers	38
2.5.5.4	Stored Procedures	39
2.5.6	<i>Arquitetura Cliente/ Servidor</i>	40
2.5.7	<i>Sistema Gerenciador de Banco de Dados (SGDB)</i>	40
2.5.8	<i>Firebird</i>	40
2.5.8.1	Principais Características	41
2.5.8.2	Usuários de peso	41
2.6	PROGRAMAÇÃO	42
2.6.1	<i>Algoritmo</i>	42
2.6.2	<i>Linguagens de Programação</i>	42
2.6.2.1	Linguagem de Máquina	42
2.6.2.2	Tradução	43
2.6.2.3	Níveis de Abstração	43
2.6.2.4	Paradigmas de Programação	43
2.6.3	<i>Linguagem Delphi</i>	44
2.6.3.1	Object Pascal	44
2.6.3.2	Borland Delphi 7	45
2.6.3.3	Conexão com o Banco de Dados	47
3	DESENVOLVIMENTO	50
3.1	MODELOS DE PROCESSO DE ENGENHARIA DE SOFTWARE	50
3.1.1	<i>Comunicação</i>	51
3.1.2	<i>Criação e modelagem do projeto rápido</i>	51
3.1.3	<i>Construção do protótipo</i>	51
3.1.4	<i>Implantação, Entrega e Feedback</i>	51
3.2	MODELAGEM UML	52
3.2.1	<i>Ferramenta utilizada</i>	52
3.2.2	<i>Construção dos diagramas</i>	53
3.2.2.1	Diagrama de caso de uso	53
3.2.2.2	Diagrama de classe	53
3.2.2.3	Diagrama de atividades	54
3.3	CONSTRUÇÃO DO BANCO DE DADOS	56
3.3.1	<i>Ferramentas utilizadas</i>	56
3.3.2	<i>Criação de domínios e tabelas</i>	57
3.3.3	<i>Geração de autonumeração</i>	58
3.3.4	<i>Utilização de Stored Procedures</i>	59
3.3.5	<i>Consultas à partir de Views</i>	60
3.4	IMPLEMENTAÇÃO DA APLICAÇÃO	61
3.4.1	<i>Ferramenta Utilizada</i>	61
3.4.2	<i>Conexão com o BD</i>	61
3.4.3	<i>Interface da Aplicação</i>	63
3.4.4	<i>Tela de Busca</i>	64
3.4.5	<i>Tela de Detalhes do Filme</i>	66
4	CONCLUSÃO	71
4.1	DIFICULDADES ENCONTRADAS	71
4.2	TRABALHOS FUTUROS	71
5	REFERÊNCIAS BIBLIOGRÁFICAS	72
6	ANEXOS	74
6.1	ANEXO 1 – FEEDBACK DO CLIENTE	74
6.2	ANEXO 2 – IMPLEMENTAÇÃO DO TECLADO VIRTUAL	76
6.3	ANEXO 3 – CONSTRUÇÃO DE UM PLAYER DE VÍDEO	78

1 INTRODUÇÃO

Atualmente, a automação comercial vem apresentando um crescimento considerável, juntamente com ela a complexidade das interfaces, onde vários indivíduos têm acessos aos terminais comerciais, sendo que cada um possui um perfil distinto e também apresenta dificuldades diferentes. Pois, enquanto um grupo de usuário consegue interagir com uma interface mais complexa, outros usuários encontram dificuldades na interação, causando insatisfação e desconforto.

Com esta necessidade de acompanhar a evolução da tecnologia sem deixar de lado a preocupação sobre o modo de interação entre usuário e sistema, o desenvolvimento de aplicações voltadas para um público com características variadas precisa cada vez mais ser focado na Usabilidade e na Ergonomia.

Desenvolver aplicações utilizando estes conceitos como base, resulta na construção de uma interface simples, clara e objetiva, fazendo com que a interação com o usuário aconteça de uma forma natural, cômoda e satisfatória, diminuindo a ocorrência de erros, otimizando a utilização e fazendo com que o sistema seja bem aceito pelos seus usuários.

1.1 *Objetivos*

1.1.1 Geral

Desenvolver uma interface clara e objetiva, que faça com que o usuário a utilize de forma independente e interaja com a aplicação de uma forma natural e intuitiva, com clareza e objetividade, buscando sempre a comodidade e a satisfação do usuário ao ter contato com esta interface.

1.1.2 Específico

- Utilizar os conceitos de Interface Homem-Máquina, no desenvolvimento da interface da aplicação, principalmente usabilidade, otimizando a interação com o usuário;
- Aplicar modelagem UML, construindo diagramas e estabelecendo uma visão do sistema para que possa ser implementado corretamente;

- Utilização da Engenharia de Software, onde através de modelos de processo, testar, avaliar e estruturar o protótipo, tendo metas de organização.

1.2 Justificativa

Em algumas aplicações voltadas para atendimento a um público variado, como por exemplo, de caixas eletrônicos, é possível notar a grande insatisfação dos usuários, que nem sempre conseguem interagir corretamente com o sistema, onde são induzidos a fazer escolhas erradas, pela falta de clareza, causando transtorno e fazendo com que o usuário evite utilizar o sistema.

Devido a essas dificuldades encontradas por alguns usuários, foi desenvolvido este projeto, onde se preza pela qualidade da interação entre o usuário e o sistema, construindo uma interface planejada para que esta interação exista de forma natural e objetiva.

1.3 Organização da Monografia

A divisão e organização da monografia foram feitas da seguinte forma:

- Capítulo 1: Introdução e escopo do projeto;
- Capítulo 2: Referencial teórico;
- Capítulo 3: Desenvolvimento do projeto;
- Capítulo 4: Conclusão;
- Capítulo 5: Referências bibliográficas;
- Capítulo 6: Anexos.

2 REFERENCIAL TEÓRICO

2.1 AUTOMAÇÃO COMERCIAL

O termo automação comercial pode ser definido como a transformação de tarefas manuais repetitivas em processos automáticos executados por uma máquina, com intuito de tomar tais tarefas mais ágeis e eficientes, na área comercial. (AUTOMAÇÃO, 2008)

A automação comercial faz parte na vida da sociedade moderna devido ao grande número de empresas que utilizam esta tecnologia, contudo nem todos estão cientes dos seus benefícios. Um estabelecimento automatizado estará preparado para atender com mais qualidade o exigente consumidor, o atendendo com mais agilidade e precisão, e possibilitando uma maior comodidade, tanto para a empresa quanto para o cliente.

A missão da automação comercial é proporcionar à empresa um gerenciamento mais eficiente, tornando possível uma análise geral da empresa, auxiliando na tomada de decisões com o objetivo de fazer com que a empresa cresça.

2.1.1 Benefícios e Dificuldades

A utilização da automação comercial traz inúmeros benefícios às empresas em geral. Podendo citar como exemplos a redução de custos; o aumento da eficiência nos processos administrativos, obtendo uma melhor gestão do negócio; a agilidade no atendimento ao cliente, atendendo suas necessidades mais rapidamente; e reduzindo a ocorrência de erros e falhas.

Algumas dificuldades também são identificadas na implantação da automação comercial em um estabelecimento, como insegurança e resistência às mudanças e a questão do treinamento dos usuários. (AUTOMAÇÃO, 2008)

2.1.2 Contexto Atual

A automação comercial deixou de ser uma vantagem competitiva e se tornou um fator de sobrevivência, exigindo um planejamento adequado e uma implementação

correta para que se torne uma ferramenta poderosa de gestão do negocio. O número de estabelecimentos comerciais automatizados vem praticamente dobrando a cada ano, e a um ritmo acelerado.

Atualmente, é um dos setores mais promissores para o mercado de tecnologia, movimentando em torno de um bilhão de reais por ano no país, sendo 60% do mercado de negócios específicos para automação comercial representados pelas empresas IBM e Itautec Philco. (AUTOMAÇÃO, 2008).

2.2 INTERFACE HOMEM-MÁQUINA

A interface homem máquina (IHM) é um elemento indispensável para aceitação de um sistema interativo pelo usuário. Estudos mostram que o usuário prefere uma interface mais simples e agradável a uma que lhe forneça maiores funcionalidade e se tornando uma interface pobre deixando a eficiência do usuário afetada. Se os fatores humanos são levados em consideração, o diálogo é amigável e um ritmo é estabelecido entre o homem e a máquina.

A IHM é uma preocupação da indústria muito antes de se falar em interface computacional. Existe uma experiência adquirida sobre o assunto, como por exemplo, a utilização de mais de um perfil do operador e níveis de conhecimento.

2.2.1 O Computador na Relação homem-máquina

A dependência da espécie humana de computador é enorme, tanto que empresas e até mesmo pessoas não conseguiriam viver sem a ajuda dessa máquina.

Várias pesquisas vêm acompanhando a relação que o homem e o computador, tanto que IHM foi constituída para estudar essa relação. Tendo em vista estes conceitos os computadores do futuro já estão sendo projetados para atender as necessidades dos usuários e coletando a sua linguagem verbal e não-verbal, já que a linguagem usual não é apropriada para o relacionamento do homem com o computador.

O computador é uma máquina que ajuda ao homem algum tipo de função com maior facilidade e foi criado para desempenhar tarefas que o homem não podia

2.2.2 Definição e importância das interfaces

Segundo Rocha e Baranauskas (2003), no surgimento do conceito de interface, ela era considerada como hardware e software que interagem com o homem, entretanto este conceito evolui, incluindo aspectos emocionais e cognitivos na interação.

Sobre essas condições foi que surgiu a prioridade de se desenvolver uma interface devidamente adequada para o usuário fazer essa interação da melhor maneira. O conceito de interface está ligado a algo discreto e tangível uma coisa que se pode desenhar, mapear, projetar e implementar.

A interface é responsável por fazer o usuário ter condições de interagir com a funcionalidade, fazer um entendimento das dificuldades da comunidade dos usuários e

tarefas. Análise dos requisitos, dos usuários, das tarefas, concepção, especificação e prototipação da interface e avaliação da utilização do protótipo.

Estamos cada vez mais dependentes de computadores e dispositivos eletrônicos seja pra trabalho ou até mesmo para nossas atividades pessoais. Essas atividades necessitam de uma facilidade de uso destes dispositivos, juntamente com interfaces difíceis não são desenvolvidas como se deveria, sendo desperdiçado o tempo e necessitando de um treinamento. Um sério problema está ligado a pessoas que investem muito com tecnologia da informação e não obtendo resultados agradáveis devido ao fato de não ter focado em uma interface amigável.

2.2.3 Fatores que influenciam a construção de interfaces

A tecnologia de construção de interfaces tem sido influenciada por diversos fatores como disseminação do uso de sistemas, aumento da complexidade e a preocupação com a qualidade do software dentro da característica de usabilidade.

2.2.4 Objetivos de IHM

Os objetivos principais objetivos do IHM são o de produzir sistemas usáveis, seguro, funcionais dando ao usuário maior comodidade e naturalidade para manipular dados de uma interface, melhorando sua usabilidade.

De acordo com Rocha e Baranauskas (2003), esses objetivos podem ser resumidos como a necessidade de desenvolver interfaces abrangendo totalmente os aspectos relacionados à interação entre usuários e sistemas.

2.2.5 Desafios de IHM

Devido ao desenvolvimento contínuo da tecnologia, com máquinas mais potentes, melhorias de hardware e software, abrem-se inúmeras possibilidades para IHM, o que a torna uma área cheia de desafios. (ROCHA e BARANAUSKAS, 2003)

Alguns dos principais desafios da área de IHM, é acompanhar a rápida evolução tecnológica e garantir *designs* com uma boa interface que explorem suas funcionalidades, possibilitando a interação do usuário com o computador de uma maneira fácil e amigável sem deixar de ser otimizada.

2.2.6 Usabilidade

Segundo Nielsen (apud. Rocha e Baranauskas, 2003, p. 28), nos sistemas computacionais, a Usabilidade é um dos critérios que definem a aceitabilidade de um sistema. Não se constrói uma interface ótima apenas com idéias próprias, os usuários muitas vezes interpretam mal os elementos da interface, fazendo uso das tarefas de um modo diferente do que foi projetado. Para isso deve-se sempre trabalhar baseado no entendimento do usuário e de suas tarefas.

A definição da usabilidade de um sistema computacional pode ser identificada de diversas maneiras, como por exemplo, a facilidade de aprendizagem, pois o sistema necessita de uma aprendizagem fácil, para que ocorra uma rápida interação. A facilidade de memorização de suas funcionalidades é uma consequência da facilidade de aprendizagem. Outros exemplos de usabilidade são: a eficiência no seu uso, fazendo com que o usuário desenvolva sua tarefa com bastante produtividade e uma pequena taxa de erros durante a utilização de um sistema.

Outro aspecto que identifica a Usabilidade é a satisfação subjetiva do usuário com o sistema, isto é, se a forma com que o usuário interage com o sistema é agradável e faz com que ele goste de utilizá-lo.

2.2.7 Ergonomia

Ergonomia é a qualidade de como um dispositivo é adaptável ao indivíduo que o opera e a tarefa que realiza, podendo considerar que a ergonomia está na origem da usabilidade.

A ergonomia tem como objetivo fazer com que diversas ferramentas tenham seu *design* adequado às necessidades do usuário, maximizando a eficiência e confiabilidade, e aumentando a satisfação e conforto do usuário perante esta ferramenta. (ROCHA e BARANAUSKAS, 2003).

2.2.8 Principais diferenças entre IHM e Engenharia de software

Muitos fatores aproximam as áreas de IHM e Engenharia de Software, como a padronização, confiabilidade, consistência, portabilidade e integração do sistema. Entretanto, ainda é possível identificar algumas divergências entre os métodos de Engenharia de Software e IHM. A Engenharia de Software trata dos aspectos funcionais do sistema, enquanto IHM trata dos aspectos de interação com o usuário. (RETONDARO, 2008).

A figura 1 ilustra a diferença entre as duas áreas:

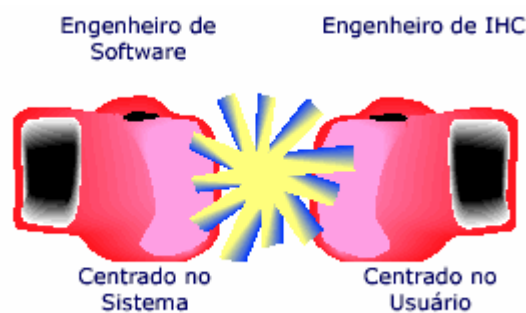


Figura 1 – Divergências entre Engenharia de Software e IHM (Fonte: RETONDARO, 2008)

2.3 ENGENHARIA DE SOFTWARE

A engenharia de software é um conjunto de três elementos fundamentais como: métodos, ferramentas e procedimentos – que facilitam ao agente e aos engenheiros uma base estruturada para a construção de software com alta qualidade e produtividade, para estarem aptos a entrar no mercado. (PRESSMAN, 1995).

Os métodos oferecem à engenharia de software a proporção nos detalhes de como construir o software. Possuem um conjunto de tarefas que incluem: planejamento, estimativa de projeto, análise de requisitos de software e de sistemas, projetos da estrutura de dados, arquitetura de programa e algoritmo de processamento codificação teste e manutenção.

As ferramentas auxiliam os métodos na automatização e sustentação, quando as ferramentas são integradas de forma que a informação criada por uma ferramenta possa ser usada por outra.

Os procedimentos fazem a ligação aproximando os métodos das ferramentas e possibilita o desenvolvimento racional e oportuno do software no seu computador.

2.3.1 Importância do software

Segundo Pressman, (1995), durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um hardware que reduzisse o custo de processamento e armazenamento. Ao longo da década de 1980, avanços na microeletrônica resultam em maior poder de computação a um custo cada vez mais baixo. Hoje o problema é diferente.

O principal desafio durante a década de 1990 é melhorar a qualidade (e reduzir o custo) de soluções baseadas em computador, soluções que são implementadas com os softwares. Para o desenvolvimento das tecnologias com enormes capacidades de armazenamento e processamento de dados nos nossos hardwares, existe o software de extrema importância para fazer a ligação dos dados com o usuário de maneira proveitosa e objetiva dando comodidade para o usuário.

2.3.2 Aplicações do Software

O Software pode ser aplicado onde são desenvolvidos algoritmo com procedimentos de passos previamente especificados. As informações coletadas do cliente e a determinância são fatores importantes no desenvolvimento de um aplicativo. (PRESSMAN, 1995).

Determinância de informação refere-se a previsibilidade da ordem e da oportunidade da informação. Um sistema operacional multiusuário, por outro lado, aceita entradas que têm conteúdo variado e regulagem de tempo arbitrária, executa algoritmos que podem ser interrompidos por condições externas e produz saída que varia como uma função do ambiente e do tempo. Aplicações com essas características são indeterminadas.

2.3.3 Ciclo de Vida Clássico da Engenharia de Software

Para Pressman (1995), O ciclo de vida requer uma abordagem sistemática, sequencial ao desenvolvimento do software, que inicia no nível do sistema e avança ao longo da análise, projeto, codificação, teste e manutenção, o paradigma do ciclo de vida as seguintes atividades. Conforme o exemplo na figura seguinte.

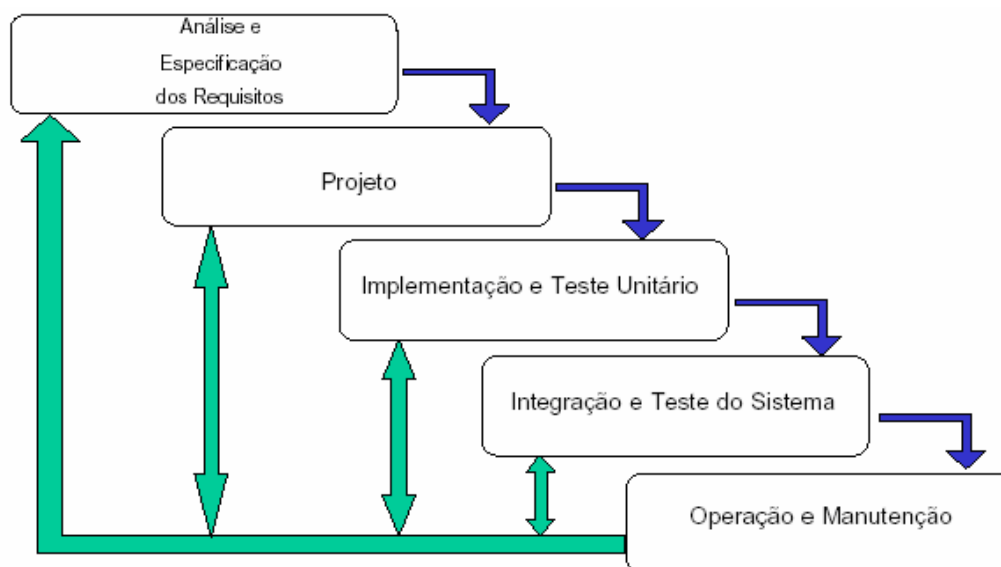


Figura 2 - Ciclo de vida clássico (Fonte: DCA, 2008)

2.3.3.1 Análise e Engenharia de Sistemas

Os requisitos estabelecidos são de extrema importância para todos elementos do sistema e prossegue com a atribuição de certo subconjunto desses requisitos ao software. Esta parte o projeto é importante também para o software fazer a interface com os elementos, tais como hardware, pessoas e banco de dados.

2.3.3.2 Análise de Requisitos de Software

As informações auxiliam o engenheiro na construção do programa, bem como na interface adequada para satisfazerem à requisição do cliente. Os requisitos, tanto para o sistema como para o software, são documentados e revistos com o cliente.

2.3.3.3 Projeto

O projeto de software consiste em processo de vários passos que se concentra em quatro atributos distintos do programa: estrutura de dados, arquitetura do software, detalhes procedimentais e caracterização de interface.

2.3.3.4 Codificação

A parte de codificação necessita de uma linguagem legível por máquina, esta etapa de codificação executa essa tarefa, se o projeto for executado detalhadamente, a codificação pode ser executada mecanicamente.

2.3.3.5 Testes

A partir do momento que o código for gerado, inicia-se a parte de testes do programa. Estes testes são feitos nas instruções internas do software e também nos aspectos funcionais externos, ou seja, as entradas devem corresponder resultados reais com as necessidades exigidas.

2.3.3.6 Manutenção

A manutenção é feita depois de ser entregue ao cliente, onde se deve: ao fato de ter encontrado erros, mudanças no seu ambiente externo, mudanças para adaptar ao sistema operacional ou dispositivo periférico ou acréscimo de cliente.

2.3.4 O Processo de Gerencia de Projetos

A gerência de projetos é a primeira camada do processo de engenharia de software. Nós a chamamos camada, em vez de etapa ou atividade, porque ela abrange todo o processo de desenvolvimento, do começo ao fim. (PRESSMAN, 1995).

Ao se conduzir um projeto de software bem-sucedido, devemos estar lúcidos com relação ao escopo do trabalho a ser feito, os riscos que poderão ocorrer, os recursos exigidos, as tarefas a serem desenvolvidas e executadas, os marcos de referência a serem acompanhados, o esforço (custo) despendido e a programação a ser escolhida e seguida. Contudo a gerência de projeto de software oferece essa compreensão.

2.3.5 Administração de Projetos

O processo de administração de projetos de software começa com um conjunto determinado de atividades que se interagem para formação do planejamento. A primeira dessas atividades é denominada estimativa.

Segundo Pressman, (1995), a estimativa dos recursos, custos e programação de atividades para um esforço de desenvolvimento de software exige experiência, acesso a boas informações históricas e a coragem para se comprometer com medidas quantitativas quando dados qualitativos forem tudo que existir. Pontos importantes para determinar o projeto:

- a) Complexidade o projeto:** Tem um forte efeito sobre a incerteza que é inerente ao planejamento.
- b) Tamanho do projeto:** É outro fator importante que pode afetar a precisão e a eficácia das estimativas.
- c) O grau de estrutura do projeto:** Também exerce um efeito sobre os riscos da estimativa.
- d) Disponibilidade de informações históricas:** Também determina os riscos da realização de estimativas.

2.3.6 Objetivos do Planejamento de Projetos

O objetivo do planejamento de projetos de software é fornecer estrutura que realize estimativas com relação aos recursos em curtos prazos. Essas estimativas são desenvolvidas em período curto de tempo de quando se inicia o projeto e devem ser atualizadas com o decorrer que se desenvolve o projeto.

2.3.7 Modelos de Processo

Segundo Pressman (1995), os modelos de processo são propostos para organizar o desenvolvimento de software, oferecendo um roteiro consideravelmente efetivo para as equipes de software.

Os modelos de processo podem ser prescritivos, em cascata, incrementais ou evolucionários.

2.3.8 Escopo do Software

O escopo do projeto é a primeira atividade determinada pelo planejamento, onde a função e o desempenho atribuídos ao software durante o trabalho de engenharia de sistema computadorizado devem ser avaliados para que o escopo seja claro e compreensível tanto em nível técnico como administrativo. Por tanto os dados quantitativos, ou seja, número de usuários simultâneo, quantidade de clientes, tempo máximo de resposta permitido, também são declarados explicitamente as restrições e os fatores mitigantes (por exemplo, algoritmos desejados que sejam bem compreendidos e estão disponíveis) são descritos.

2.3.9 Software Comercial

Uma das áreas mais usadas pelos desenvolvedores de software, onde os “sistemas” foram evoluídos e trocados por software de sistemas de informação administrativa (MIS), dando acesso a um ou mais banco de dados contendo informações comerciais. Essas aplicações reestruturam os dados organizando de uma maneira que possibilita operações comerciais e as tomadas de decisões administrativas, além disso, as aplicações de software comercial focalizam um importante item que é a computação interativa (por exemplo, processamento de transações em pontos de venda).

2.3.10 Importância do Software em Aplicação Comercial

O software com o passar do tempo passa a construir uma tecnologia chave, sendo usada nas mais diversas áreas, dentre elas será citada as aplicações comerciais, a indústria de computador passa a ser um dos ramos de negócio mais competitivo do planeta, onde o componente software torna-se a força prevalecente em termos de inovação tecnológica.

Os princípios de engenharia de software levam a um software otimizado fazendo com que as aplicações comerciais sejam confiáveis, fáceis de compreender e levando ao cliente conforto perante os produtos, deixando também o proprietário com uma melhor visão de seu comércio, tendo um controle satisfatório e obtendo uma economia de tempo e financeiro em relação a diminuição de empregados.

2.3.11 Análise Orientada a objeto

No fim da década 1980, a análise orientada a objeto na modelagem de dados começou a surgir com intensidade de uma forma poderosa e prática auxiliando no desenvolvimento do software, criando projetos para aplicações de todos os tipos.

Hoje, a análise orientada a objeto está fazendo um lento, firme, progresso como método de análise por seus próprios méritos e como complementos a outros métodos de análise.

2.3.11.1 Conceitos da Orientação a Objetos

Para entender o ponto de vista orientado a objeto, é necessário considerar um exemplo de objeto do mundo real, como exemplo a cadeira sendo um objeto da instância mobiliária onde essa classe possui os conjuntos de atributos dentre eles está, custo, dimensões, peso. Supõe agora que se deseja montar uma nova classe dentro da classe mobiliária, chamada cama, sendo uma mistura de cama e mesa herdando os atributos de mobiliário. Conforme a figura seguinte:

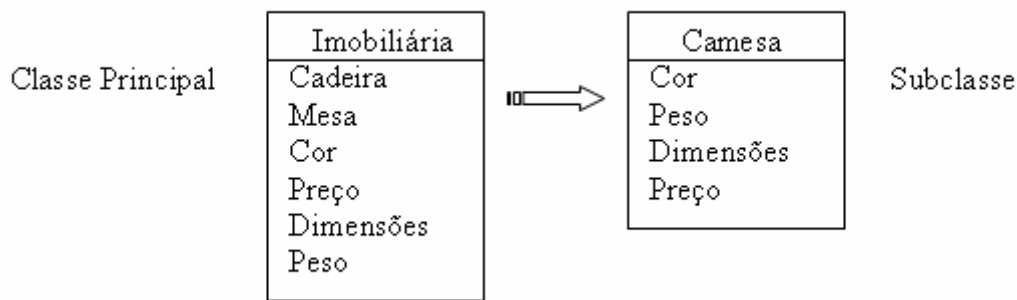


Figura 3 - Orientação à Objeto (Fonte: Autor)

2.3.11.2 Identificação de Objetos

Ao determinar um, objeto deve-se examinar a declaração do problema, ao executar uma “análise gramatical” da narrativa de processamento do sistema a ser construído. Os objetos fazem parte de uma classe onde herdaram atributos da mesma, ao pensar em uma classe bar, faz uma análise do sistema que fazem funcionar esta classe e logo se definem os objetos que fazem parte do sistema, mesa, cadeira, balcão, caixa e produtos.

Segundo Pressman, (1995), os objetos podem ser:

- a) **Entidades externas:** reproduzem ou consomem informações a serem usadas por um sistema baseado em computador, como por exemplo, outros sistemas, dispositivos, pessoas.
- b) **Coisas:** fazem parte do domínio de informação do problema, como relatórios, displays, cartas, cartazes.
- c) **Ocorrências ou eventos:** ocorrem dentro do contexto de operação do sistema, por exemplo, uma transferência de propriedade ou a conclusão de uma série de movimentos de robô.
- d) **Papéis (funções):** desempenhadas por pessoas que interagem com o sistema, como por exemplo, gerente, engenheiro, vendedor.
- e) **Unidades organizacionais:** que são pertinentes a uma organização, como divisão, grupo, equipe.
- f) **Lugares:** que estabelecem o contexto do problema e a função global do sistema, por exemplo, piso de fábrica ou área de descarga.

g) Estruturas: que definem uma classe de objetos ou, ao extremo, classes relacionadas de objetos, por exemplo, sensores, veículos de quatro rodas ou computadores.

2.3.11.3 Especificação de atributos

Os atributos especificam o objeto selecionado para inclusão no modelo de análise, eles dão as características que definem o objeto esclarecendo aquilo que o objeto significa no contexto do espaço do problema.

2.3.12 Modelagem de Dados

A técnica de modelagem de dados concentra-se unicamente nos dados, representando uma grande “rede de dados” de um determinado sistema. Modelagem de dados é muito importante devido ao relacionamento dos dados ser muito complexa, os dados dentro da modelagem são independentes, ou seja, os processos que transformam os dados não afetam sua modelagem.

2.3.12.1 Diagramas Entidade-Relacionamento

Para Pressman, (1995) a notação fundamental para a modelagem de dados é o diagrama entidade-relacionamemto (E-R).

Os objetos de dados são representados por um retângulo que ligados por relacionamento que possuem um formato de losango, esses ligamentos possuem linhas que identificam a quantidade, exemplo um funcionário pode vender muitos carros. Objetividade especifica na busca dos dados faz a diferença com relação aos outros modelos DBMS. A figura a seguir mostra um diagrama E-R:

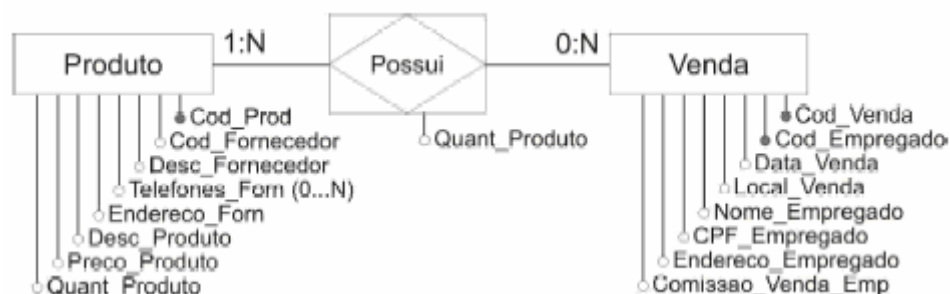


Figura 4 - Aplicação de um diagrama E-R (Fonte: PONTES, 2008)

2.4 UNIFIED MODELING LANGUAGE

A *Unified Modeling Language* ou (UML) é composta basicamente de métodos, classes, tecnologia de suporte e os fatores de sucesso na implementação. A classe é um modelo no qual são criados objetos que se relacionam, é uma descrição de um conjunto de instância que compartilham as mesmas operações, atributos, relacionamentos e semântica. Os objetos herdam da classe atributos que podem ser compartilhados por todos eles, também podem a partir da classe estabelecer relacionamentos com outras classes: duas classes podem compartilhar alguns atributos, mas manipulam outros atributos diferentes.

Na programação orientada a objetos, a herança permite expressar relacionamentos entre as classes onde as classes subjacentes derivam em subclasses mais complexas. Uma classe básica contém todos os atributos que são compartilhados entre as classes derivadas da classe base. Subclasses usam atributos compartilhados existentes, reimplementam atributos compartilhados ou criam novos atributos.

2.4.1 Diagramas da UML

Os diagramas da UML têm uma grande importância em vários aspectos de modelagem através da notação definida pelos seus vários tipos de diagramas. Um diagrama é uma apresentação gráfica representada por um conjunto de elementos de modelo, frequentemente mostrado como gráfico conectado a arcos que indicam os relacionamentos e vértices (outros elementos do modelo).

Para modelar um sistema complexo serão necessárias várias abordagens com aspectos diferentes como: o aspecto não funcional, que se baseia no tempo de processamento, confiabilidade e produção; o aspecto organizacional, que tem como base a organização do trabalho mapeamento e código; e o aspecto funcional que foca na estrutura estática e interação dinâmica.

2.4.1.1 Diagramas de caso de uso

Segundo Fowler (2005), os diagramas de caso de uso capturam os requisitos funcionais de um sistema, servem para descrever as interações típicas entre os usuários e o sistema, fornecendo uma narrativa sobre como o sistema é utilizado.

Os casos de uso possuem cenário onde mostram as seqüências de passos que descrevem uma interação entre o usuário e o sistema. Esse cenário representa várias alternativas ou ações que são feitas pelos atores dentro do sistema.

Cada caso de uso tem um ator principal, que pede ao sistema para que execute um serviço. O ator principal é aquele cujo objetivo o caso de uso está tentando satisfazer e, normalmente (mas nem sempre) é o iniciador do caso de uso. Podem existir outros atores com mais o sistema se comunica enquanto executa o caso de uso. Eles são conhecidos como atores secundários. A figura abaixo mostra um diagrama de caso de uso.

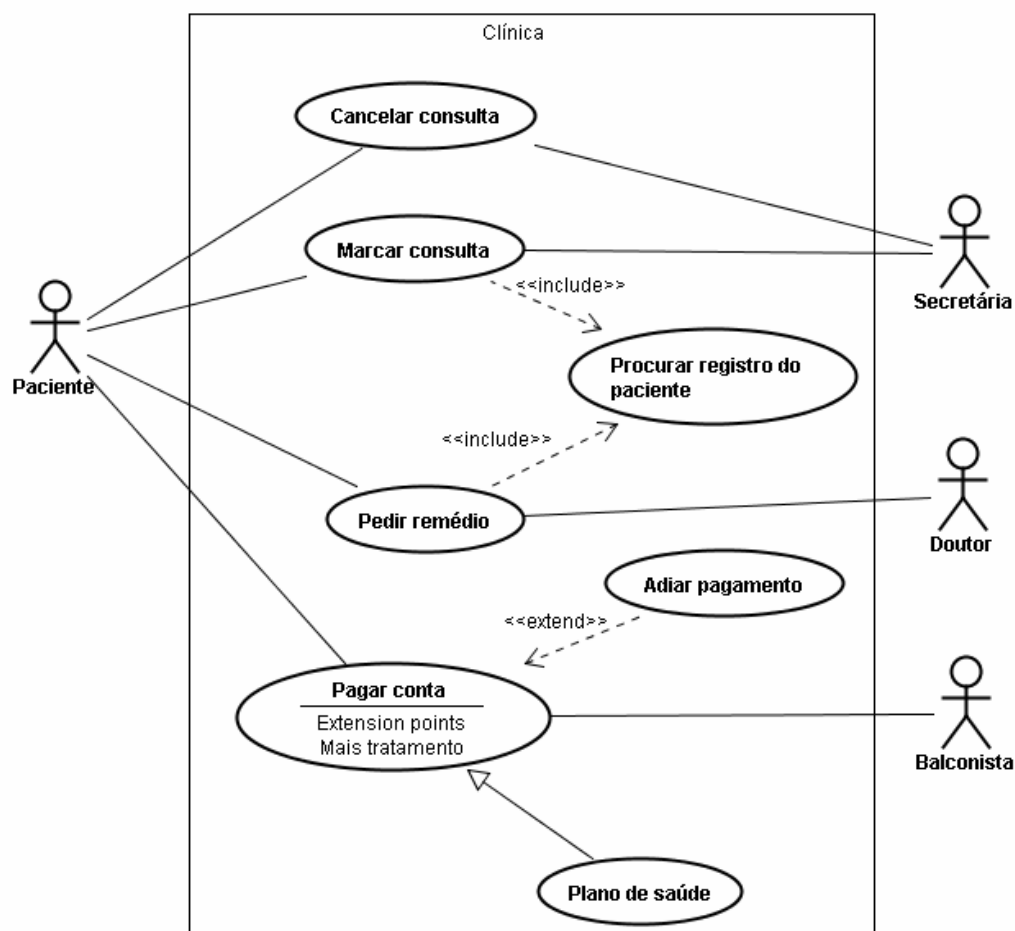


Figura 5 - Diagrama de Caso de Uso (Fonte: DSC, 2008).

2.4.1.2 Diagramas de Classe

Os diagramas de classes descrevem os objetos que estão contidos nos sistemas e os relacionamentos entre eles, também nos mostra as propriedades e as operações de uma classe juntamente com as restrições que se aplicam nas conexões dos objetos.

Diagrama de classe não aparece do nada ele é consequência do prévio levantamento de requisitos, definição de casos de usos onde serão mostradas as ações dos usuários. A figura 6 ilustra um Diagrama de Classe.

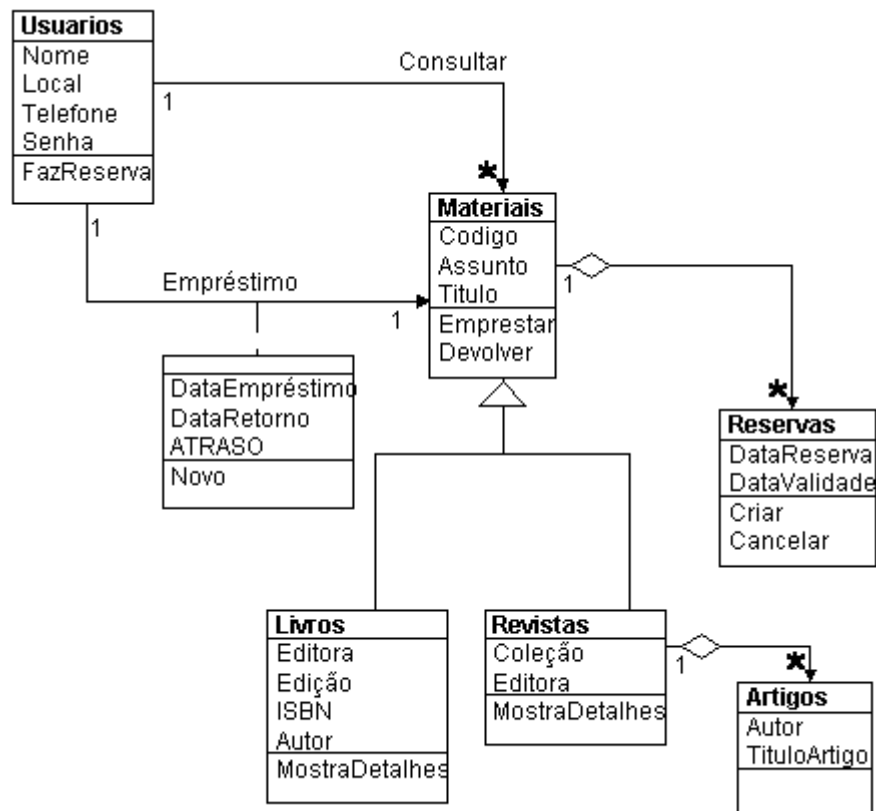


Figura 6 - Diagrama de Classe (Fonte: VOXXEL, 2008).

a) Propriedades

As propriedades correspondem aos campos de uma classes, esses campos possuem características dos objetos que se parecem com duas notações distintas: Atributos e associações. Embora elas se pareçam diferentes em um diagrama, na realidade elas são a mesma coisa.

b) Atributos

Atributos fazem a descrição de uma propriedade como uma linha de texto dentro da caixa de classe em si. A forma completa de um atributo é a seguinte:

Visibilidade nome: Tipo multiplicidade = valor-por-omissão {lista de propriedades}

Exemplo: nome: String [1] = “Sem Título” {readOnly}.

c) Associações

Segundo Fowler (2005) Associação é uma linha cheia entre duas classes, direcionada da classe de origem para a classe destino. O nome da propriedade fica no destino final da associação, junto com sua multiplicidade. O destino final da associação vincula á classe que é o tipo da propriedade. Conforme a figura abaixo:

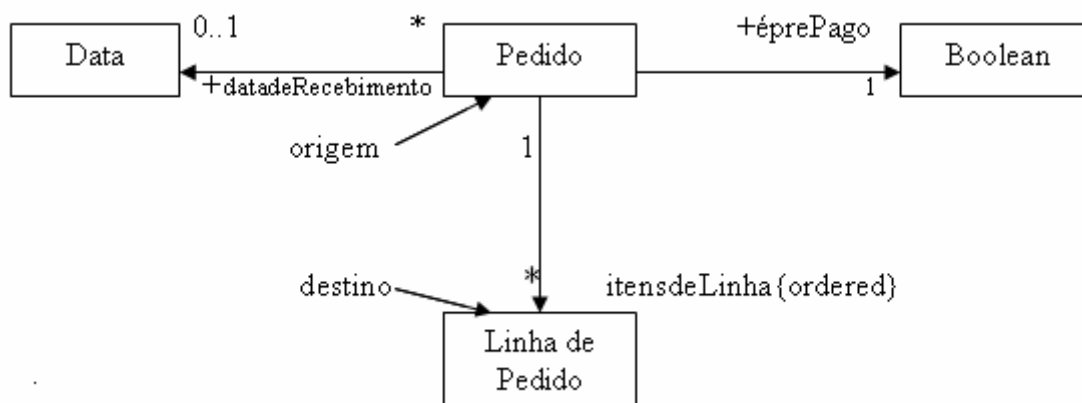


Figura 7 - Associações (Fonte: Autor)

d) Multiplicidade

Multiplicidade é a relação dos objetos identificando por exemplos quantos pedidos um cliente poderá fazer ou vários clientes podem fazer mais de um pedido ou até mesmo vários clientes podem fazer um pedido. As multiplicidades vistas mais comumente são:

- 1 (Um pedido deve ter exatamente um cliente)
- 0...1 (Um cliente corporativo pode ter ou não um único representante de vendas)

- * (Um cliente não precisa fazer um pedido e não existe nenhum limite superior para um número de pedidos que um cliente pode fazer- zero ou mais pedidos.)

e) Operações

Operações são exatamente as ações das classes, essas operações correspondem aos métodos presentes em uma classe.

f) Generalização

Generalização corresponde a uma classe principal onde as subclasses possuem alguns atributos dessa classe, um exemplo claro é o que envolve pessoas físicas e jurídicas de uma empresa. Elas possuem diferenças, porém muitas semelhanças, as semelhanças podem ser colocadas em uma classe geral: Cliente (o supertipo), com cliente pessoa física e cliente pessoa jurídica como subtipos.

2.4.1.3 Diagrama de atividade

Os diagramas de atividade consistem em uma técnica para mostrar a lógica de procedimento, realizando o processo de negócio e fluxo de trabalho. Este diagrama tem uma semelhança muito grande com os fluxogramas, tendo uma diferença em relação ao diagrama de atividade suportar comportamento paralelo. O nó inicial da o ponto de partida para a entrada e logo em seguida terá a primeira ação. Poderá, durante a sequência de ações, aparecerem a linha de separação que consiste em um fluxo de entrada e vários fluxos concorrentes de saída.

Segundo (Fowler, 2005) o diagrama de atividade permite que quem esteja seguindo o processo escolha a ordem na qual fazer as coisas. Em outras palavras, ele simplesmente determina as regras essenciais de sequência que se deve seguir.

A figura seguinte mostra um diagrama de atividade:

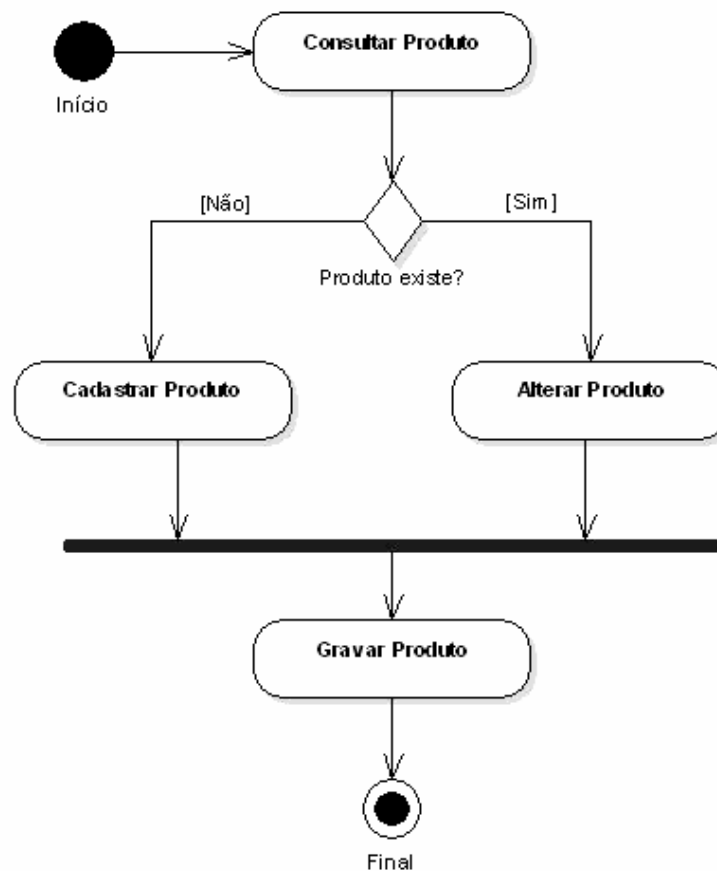


Figura 8 - Diagrama de Atividade (Fonte: LINHA, 2008)

Este diagrama de atividades mostra a primeira ação consultar produto em seguida tem o triângulo de decisão se o produto existe ou não, se existir você altera o produto se não cadastra produto logo após vem a separação em que a classe alterar e cadastrar pode estar na mesma classes, onde através desses divisores podemos separar as atividades de acordo com as classes, sendo a ultima ação gravar produto.

2.5 BANCO DE DADOS

Segundo Korth e Silberschatz (1995), um Banco de Dados (BD) consiste em um conjunto de dados inter-relacionados, projetados para gerenciar e armazenar informações.

Na década de 60, foram criados dois dos principais modelos para armazenamento de dados: o modelo de dados em rede e o modelo hierárquico.

De acordo com Souza (2007), o modelo de dados em rede, como mostra a figura 9, é representado apenas por coleções de registros e links, utilizando caixas e linhas, enquanto o modelo hierárquico organiza os dados em forma de árvores, conforme a figura 10. Ambos os modelos necessitavam de um conhecimento de estruturas física do banco para realizar consultas.

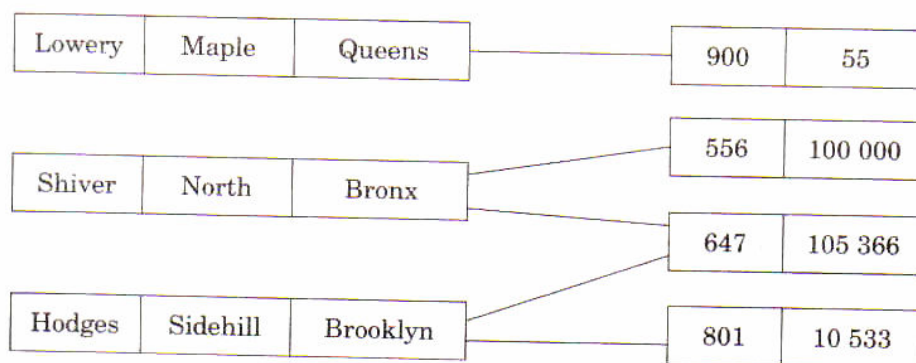


Figura 9 - Modelo em Rede (Fonte: KORTH e SILBERSCHATZ, 1995)

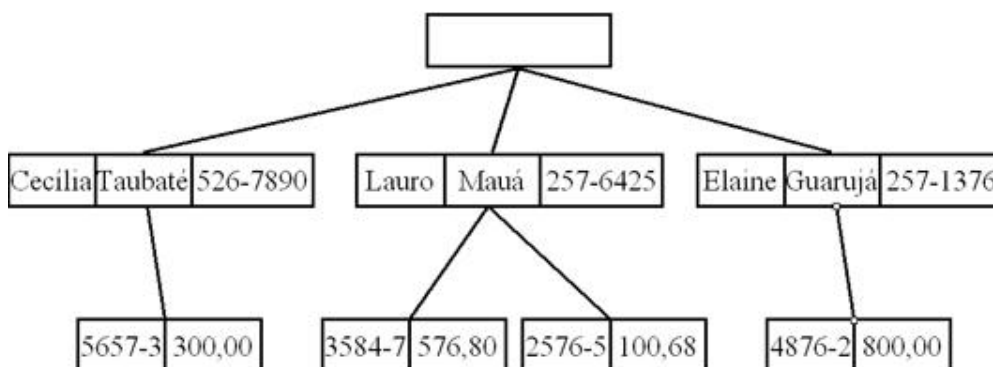


Figura 10 - Modelo Hierárquico (Fonte: DEVMEDIA, 2008)

No entanto, nenhum dos modelos citados anteriormente se tornou padrão, mas sim o modelo de dados relacional criado na década de 70 por Edgar Frank Codd, que separou a estrutura lógica do método de armazenamento físico do BD.

Em 1976, Peter Chen propôs o modelo Entidade – Relacionamento (E-R), figura que possibilitou ao projetista focar apenas na utilização dos dados, deixando de lado estrutura lógica da tabela.

2.5.1 Principais Características de um BD

Os bancos de dados têm como uma de suas principais características, a forma como são independentes dos programas, podendo ter seus dados compartilhados por múltiplas aplicações.

Um BD descreve a forma como os dados são armazenados, ou seja, a estrutura de cada arquivo, como tipo, formato, restrições, etc.

2.5.2 Abstração de dados

O SGBD fornece aos usuários uma visão abstrata dos dados omitindo alguns detalhes sobre o armazenamento das informações (KORTH e SILBERSCHATZ, 1995).

O nível Físico é o nível mais baixo, define a maneira que os dados serão armazenados no BD, sendo descritas estruturas complexas de baixo nível. O nível Conceitual define quais os dados que serão armazenados e o relacionamento entre eles, este nível é utilizado pelos administradores do BD. O nível de Visões do Usuário: o usuário tem acesso aos dados de acordo com sua necessidade, este nível tem o objetivo de tornar mais simples a interação com o sistema, fornecendo numeras visões para um banco de dados. A figura a seguir apresenta os níveis de abstração:

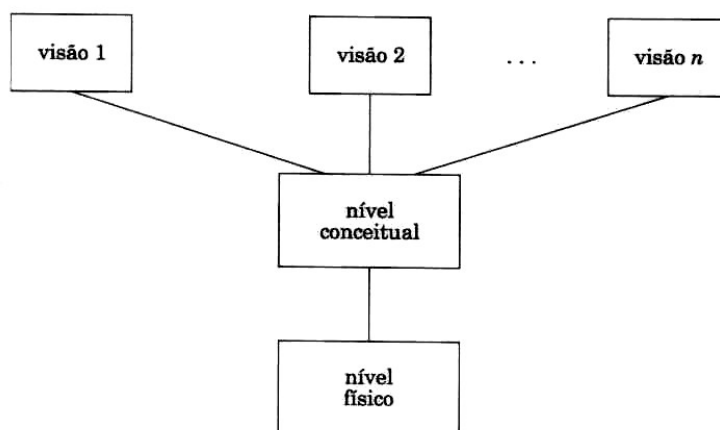


Figura 11 - Níveis de Abstração (Fonte: KORTH e SILBERSCHATZ, 1995)

2.5.3 Modelagem de Dados

A Modelagem Conceitual: é onde se descreve, de forma independente da implementação, a estrutura do banco de dados, sem informar como os dados estarão armazenados. Utiliza o diagrama Entidade-Relacionamento, projetado para estar mais próximo da visão que o usuário possui dos dados.

De acordo com Korth e Silberschatz (1995), o modelo Entidade-Relacionamento, foi projetado para facilitar o projeto de um BD, estando o mais próximo possível da visão de dados que o usuário possui, pois é baseado na percepção de objetos básicos do mundo real, as entidades e os relacionamentos.

Uma entidade consiste em um objeto concreto ou abstrato distinguível de outros objetos. São representados por um conjunto de atributos, ou seja, os valores dos dados. As entidades são classificadas em entidades fortes, quando existem de forma independente, e entidades fracas, quando dependem de uma entidade forte para existir.

Um relacionamento é uma associação entre diversas entidades, tornando possível o compartilhamento de informações. São classificados de acordo com o grau, ou seja, o número de entidades que se relacionam e de acordo com a cardinalidade, isto é, o número de registro que estão associados entre duas entidades. A figura 12 mostra a cardinalidade dos relacionamentos entre as entidades (a) e (b).

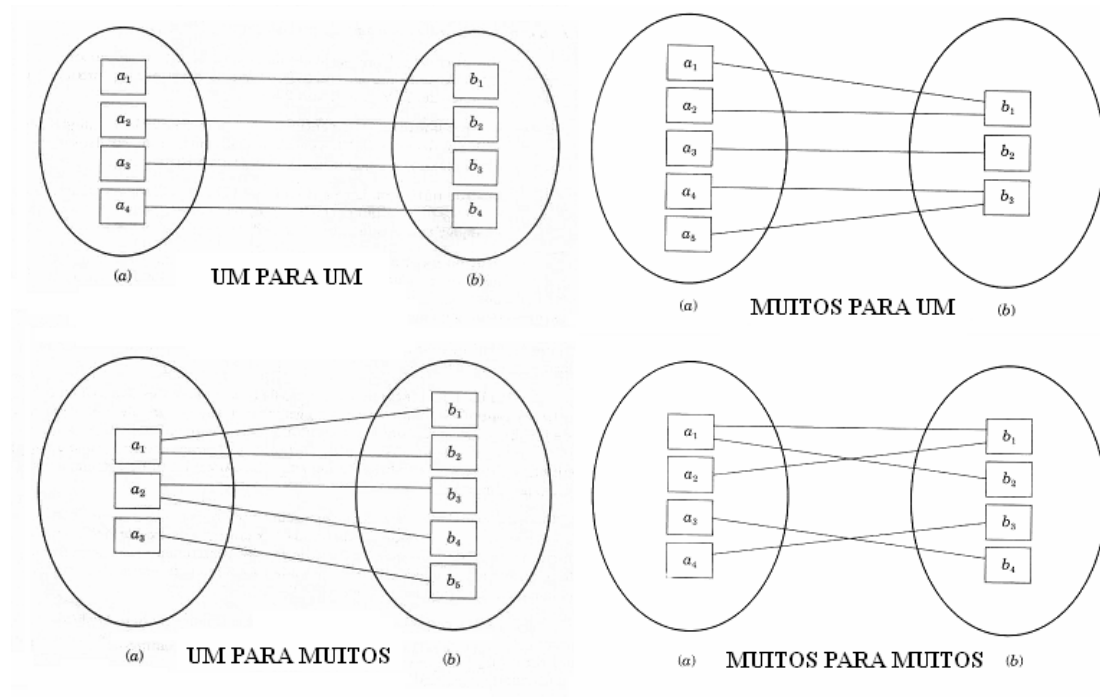


Figura 12 - Tipos de Cardinalidade (Fonte: Autor)

O modelo E-R é representado graficamente pelo diagrama Entidade-Relacionamento, que fornece um conceito generalizado sobre a estruturação dos dados no sistema. A figura seguinte mostra um diagrama E-R.

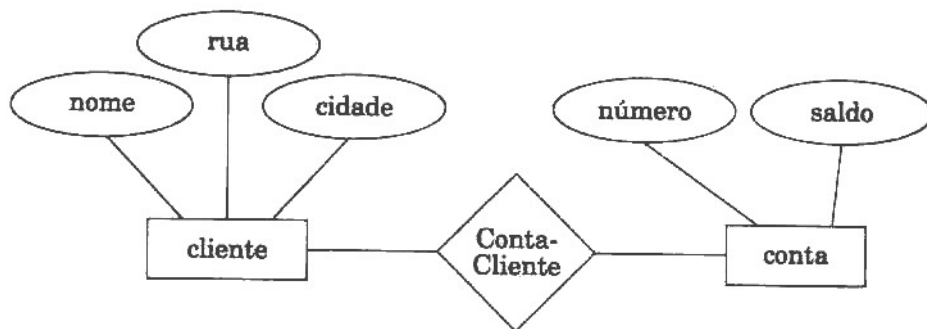


Figura 13 - Diagrama E-R (Fonte: KORTH e SILBERSCHATZ, 1995)

2.5.4 Modelagem Lógica

Na modelagem lógica, é onde se define como o banco de dados será implementado e se obtém uma visão mais detalhada sobre o armazenamento dos dados, organizando-os em um banco de dados relacional, conforme SOUZA (2007).

2.5.4.1 Banco de dados relacional

De acordo com Korth e Silberschatz (1995), um banco de dados relacional consiste em uma coleção de tabelas, onde suas linhas representam um relacionamento entre um conjunto de valores. A figura 14 representa um banco de dados relacional:

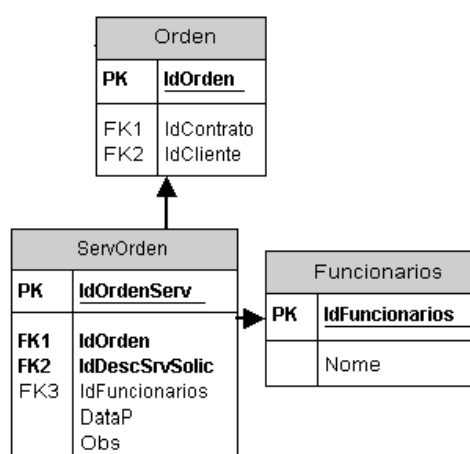


Figura 14 - Tabelas de um BD Relacional (Fonte: ENG, 2008)

2.5.4.2 Álgebra Relacional

Álgebra relacional é uma linguagem de consulta procedural, que consiste em uma coleção de operações canônicas para manipular relações, selecionando tuplas, ou seja, conjuntos de registros, para especificar uma determinada consulta em um BD. (SOUZA, 2007).

As operações *SELECT* e *PROJECT*, são algumas das operações fundamentais da álgebra relacional. O *Select* é utilizado para selecionar um conjunto de tuplas que satisfazem uma determinada condição de seleção, conforme o exemplo a seguir.

$\sigma_{\langle \text{condições_de_seleção} \rangle} (\langle \text{nome da seleção} \rangle)$.

O *Project*, por sua vez, seleciona um conjunto determinado de colunas de uma relação, como a seguinte exemplo:

$\pi_{\langle \text{lista_de_atributos} \rangle} (\langle \text{nome da relação} \rangle)$.

Project e *Select* também podem ser utilizados em conjunto, fazendo com que determinadas colunas sejam selecionadas. O exemplo a seguir mostra este caso:

$\pi_{\langle \text{lista_de_atributos} \rangle} (\sigma_{\langle \text{condições_de_seleção} \rangle} (\langle \text{nome da seleção} \rangle))$.

2.5.5 Structured Query Language (SQL)

Segundo Korth e Silberschatz (1995) a *Structured Query Language* (SQL), é uma linguagem de consulta estruturada desenvolvida no início dos anos 70, no laboratório de pesquisa da IBM, em San Jose.

A linguagem SQL tornou-se a linguagem padrão para acessar bancos de dados. Uma de suas principais característica é o fato de ela não ser uma linguagem procedural, por processar conjuntos de registros, ao invés de um registro por vez. Utilizando a linguagem SQL, o usuário é capaz de manipular tipos complexos de dados.

Divide-se basicamente em duas partes: a linguagem de definição de dados (DDL) e a linguagem de manipulação de dados (DML). (SOUZA, 2007).

A DDL oferece comandos para definir esquemas de seleção, modificação e remoção de relações e criação de índices. O código a seguir mostra um comando SQL para definição de dados, onde se cria uma tabela (relação).

```
CREATE TABLE <nome_tabela> (<nome_coluna1><tipo_coluna1>),  
(<nome_coluna_n><tipo_coluna_n>).
```

A DML inclui uma linguagem de consulta, que se baseia na álgebra relacional, onde se utilizam comandos para inserção, remoção e alteração de tuplas em um banco de dados.

Em SQL o comando *SELECT* é utilizado para selecionar tuplas e atributos em uma ou mais tabelas, como mostra a forma básica a seguir:

```
SELECT <lista de atributos>  
FROM <lista de tabelas>  
WHERE <condições>
```

Inserções e atualizações de registros em SQL são feitos a partir da utilização dos comandos *INSERT INTO* e *UPDATE*, respectivamente. As formas gerais a seguir mostram como são feitas estas operações nas tuplas de um banco de dados relacional:

Inserção:

```
INSERT INTO <nome da tabela> <(lista de colunas)>  
VALUES <(lista de valores)>;
```

Atualização:

```
UPDATE <tabelas>  
WHERE <condição>.
```

2.5.5.1 Views

Conforme Korth e silberschartz, as *Views* ou visões são definidas como um mecanismo utilizado para tornar mais simples as consultas do BD, estabelecendo a consulta que irá computar a visão. O exemplo a seguir mostra a criação de uma *View*:

```
CREATE VIEW <nome da visão> AS  
SELECT <lista de atributos>  
FROM <lista de tabelas>
```

2.5.5.2 Generators

Um *Generator* é uma ferramenta que retorna um valor incrementado toda vez que é chamado. São independentes de transações e podem ser usados de várias maneiras. Segue um exemplo de como um *generator* é criado:

```
CREATE GENERATOR GEN_CLIENTE_ID;  
SET GENERATOR GEN_CLIENTE_ID TO 1;
```

2.5.5.3 Triggers

De acordo com Korth e Silberschartz (1995), uma *trigger*, também conhecida como gatilho, consiste em um comando que é executado de forma automática pela aplicação como consequência de uma alteração no banco de dados.

Na construção de uma *trigger* é necessário especificar as condições em que a *trigger* será acionada e as ações que serão executadas após o seu acionamento.

Em SQL uma *trigger* é criada utilizando código conforme o exemplo a seguir.

```
CREATE TRIGGER <nome_da_trigger> FOR <nome_da_tabela>
ACTIVE <condição_de_ativação> POSITION 0
AS
BEGIN
    <corpo_da_trigger>
END
```

Através das *triggers*, operações podem ser executadas sempre que determinador eventos ocorrem em uma tabela ou *view*, como antes da inserção em uma tabela (*Before Insert*), após a inserção (*After Insert*), antes de efetuar uma alteração (*Before Update*), depois da alteração (*After Update*), antes de uma exclusão (*Before Delete*) ou depois da exclusão (*After Delete*).

Uma das principais utilizações das *triggers* é para implementar um mecanismo de auto-incremento, como mostra o seguinte exemplo:

```
CREATE TRIGGER P_CONTAS_BI FOR P_CONTAS
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
    IF (NEW.COD IS NULL) THEN
        NEW.COD = GEN_ID(GEN_P_CONTAS_ID,1);
    END
```

2.5.5.4 Stored Procedures

As *Stored Procedures* são conjuntos de instruções executadas dentro do banco de dados, utilizadas para encapsular tarefas repetitivas, recebendo parâmetros de entrada e retornando um valor na saída. (DEV MEDIA, 2008).

A utilização de *stored procedure* no banco de dados traz inúmeras vantagens:

Modularidade: Alternando somente as suas operações, as modificações estarão disponíveis para toda aplicação, pois o procedimento se encontra dividido do restante do software;

Redução do tráfego na rede: com a passagem de parâmetros para o servidor e utilizando o seu processamento, há uma redução do tráfego dos dados pela rede;

Melhor performance: os *stored procedure* são pré compilados e aguardam memória cachê até serem chamados para executar determinada operação e recebem o valor dos parâmetros tornando sua execução mais rápida e otimizada a performance da aplicação.

A criação de uma *stored procedure* é mostrada a seguir:

```
CREATE PROCEDURE <nome_da_procedure>(<parâmetros_de_entrada>
RETURN <lista_de_parâmetros_de_saída>
AS <lista_de_variáveis_locais>
BEGIN
    <corpo_da_procedure>
END
```

Comandos SQL também podem ser utilizados dentro de uma *stored procedure*, sendo possível a utilização de variáveis locais ou parâmetros de entrada. O exemplo seguinte mostra a utilização de um comando SQL na *stored procedure*:

```
CREATE PROCEDURE INSERIR_CLIENTE (
    ID_CLIENTE INTEGER, NOME_CLIENTE CHAR (30))
AS
BEGIN
    INSERT INTO CLIENTE (ID_CLIENTE, NOME_CLIENTE)
    VALUES (:ID_CLIENTE, :NOME_CLIENTE);
END
```

Para utilizar uma *stored procedure* em uma aplicação Delphi, por exemplo, é necessário que sejam passados os parâmetros de entrada e que o comando *ExecProc* seja executado, conforme o código a seguir:

```
procedure TForm1.btnGravarClick(Sender: TObject);
begin
    with spCliente do
        begin
            Params[0].AsInteger := cdsClienteID_CLIENTE.AsInteger;
            Params[1].AsString := cdsClienteNOME_CLIENTE.AsString;
            ExecProc;
        end;
end;
```

2.5.6 Arquitetura Cliente/ Servidor

De acordo com Oliveira (2000), a arquitetura Cliente/ Servidor, descreve o processamento cooperativo distribuído, onde o relacionamento entre os Clientes e o Servidor, realiza-se entre componentes de software e hardware.

2.5.7 Sistema Gerenciador de Banco de Dados (SGDB)

Segundo Korth e Silberschatz (1995), o principal objetivo dos SGDB's é prover um ambiente onde se pode recuperar e armazenar informações de maneira eficiente, definindo estruturas de armazenamento e mecanismo de manipulação de dados, evitando redundância e inconsistência desses dados e fornecendo segurança às informações.

São os principais exemplos de um SGDB: Firebird, InterBase, PostgreSQL, SyBase, SQL-Server, MySQL, e outros.

2.5.8 Firebird

Segundo (Cantu, 2005), com a abertura de seu código fonte do InterBase em julho de 2000, tornando-o *Free* e *Open Source*, a comunidade de usuários criou um novo banco de dados, o Firebird, compatível com o padrão SQL-ANSI-92.

Entretanto, a Borland continuou desenvolvendo uma versão comercial do InterBase, esta não teria o código aberto e nem seria *free*, o que causou uma migração

de usuários para o Firebird, pois após a criação de novas versões e atualizações no seu código ao contrário do InterBase o Firebird se mostrou uma melhor opção por estar em pleno desenvolvimento e pela garantia de continuar sendo *Open Source*.

2.5.8.1 Principais Características

O Firebird mantém total compatibilidade com o InterBase 6.0, possuindo inúmeras melhorias relacionadas à adição de novos recursos e correções de *bug's*, sendo a facilidade de instalação e manutenção do Firebird um diferencial em relação aos seus concorrentes, pois não exige conhecimento específico.

Ao contrário de seus concorrentes o Firebird utiliza um modelo de concorrência diferente do modelo baseado em travas, o *Versioning*, baseado em um sistema otimista de concorrência, que presume que a chance de mais de um usuário alterar um registro de dados ao mesmo tempo seja muito baixa.

Não necessita de manter um log de transações, para desfazer as últimas operações não concluídas, o Firebird altera automaticamente o *flag* de transações pendentes para *Rollback* assim que o servidor é reiniciado, retornando quase instantaneamente o banco a um estado consistente.

Sua alocação de dados é feita de forma dinâmica, ou seja, a medida que os dados são inseridos, os arquivos do banco de dados crescem no disco. Porém, quando um volume de informações é removido do banco, o seu tamanho em disco não diminui, pois o Firebird deixa o banco com o mesmo tamanho, reaproveitando o espaço já alocado para novas informações, visando melhor performance.

O Firebird é multiplataforma, com versões para as plataformas Win32, MacOS, Linux, Solaris, e outras e suporta os protocolos de rede TCP/IP e NetBEUI.

2.5.8.2 Usuários de peso

O Firebird é utilizado em uma grande variedade de empresas internacionais e nacionais, destacando-se a Embrapa (Empresa Brasileira de Pesquisa Agropecuária), ESALQ-USP, e centenas de empresas de desenvolvimento de software.

2.6 PROGRAMAÇÃO

No desenvolvimento de softwares é necessário que sejam definidas seqüências de instruções que executam tarefas até atingir um objetivo ou solução de um determinado problema. (MORAES, 2000)

2.6.1 Algoritmo

Segundo Moraes (2000), algoritmos são seqüências finitas de comandos e instruções que, quando obedecidos, levam a execução de uma tarefa e alcançam um objetivo proposto.

Os algoritmos são independentes das linguagens de programação, pois para serem escritos é utilizado o pseudocódigo, uma linguagem simples e de fácil entendimento, que não é executado em um sistema real. O exemplo a seguir mostra um algoritmo escrito utilizando pseudocódigo:

```
INICIO
    LEIA (A);
    LEIA (B);
    LEIA (C);
    MEDIA  $\leftarrow$  (A + B + C) / 3;
    ESCREVA ("A média é:", MEDIA);
FIM.
```

2.6.2 Linguagens de Programação

De acordo com Farrer et al. (1989), as Linguagens de Programação são utilizadas para transcrever os algoritmos para uma linguagem que o computador compreenda direta ou indiretamente e o traduza para linguagem de máquina, para que possa ser executado.

2.6.2.1 Linguagem de Máquina

A Linguagem de Máquina diz respeito ao conjunto de instruções que tem a capacidade de ativar diretamente os dispositivos eletrônicos do computador, como registradores e acumuladores, sendo diferente para cada arquitetura. (FARRER et al., 1989)

É extremamente rudimentar, sendo totalmente expressa em forma binária ou hexadecimal, o que torna seu entendimento mais difícil de ser assimilado. Também necessita de um cuidado especial para se estabelecer os posicionamentos dos dados e instruções na memória do computador.

2.6.2.2 Tradução

Existem dois métodos de tradução da linguagem de programação utilizada para a linguagem de máquina, que são a compilação e a interpretação. (WIKIPEDIA, 2008)

Na compilação, o código é traduzido antes que o programa seja executado, gerando um arquivo executável, como nas linguagens Pascal e C, por exemplo; enquanto na interpretação a tradução é feita durante a execução do programa, um exemplo disso é a linguagem Java.

2.6.2.3 Níveis de Abstração

As linguagens de programação se dividem em dois tipos distintos, quanto ao grau de abstração: as linguagens de baixo nível e as linguagens de alto nível. (WIKIPEDIA, 2008)

Quando possuem características mais próximas da arquitetura do computador, são classificadas como linguagens de baixo nível, pois trabalha diretamente com registradores do processador. A linguagem Assembly é um exemplo de linguagem de baixo nível.

As linguagens de alto nível são mais próximas da linguagem humana e dispensam o conhecimento de instruções em nível de hardware. Como exemplos de linguagens de alto nível podem ser citados Pascal, Object Pascal (Delphi), C, C++, C #, Java, PHP e Visual Basic, dentre outras.

2.6.2.4 Paradigmas de Programação

Diversas formas são identificadas quanto à abordagem da sintaxe de uma linguagem de programação. Alguns exemplos são: a Programação Estruturada e a Programação Orientada a Objeto. (WIKIPEDIA, 2008).

A Programação Estruturada tem como princípios básicos três mecanismos fundamentais para o desenvolvimento de algoritmos, que são a sequência, a seleção e a iteração, o que torna a compreensão do programa mais fácil.

No mecanismo de sequência, o programa é executado passo a passo, de forma que tarefas são executadas sequencialmente; na seleção é onde se encontram as estruturas de decisão; na iteração são utilizadas as estruturas de repetição (DCA, 2008).

Conforme Leite e Rahal (2002), a Programação Orientada a Objeto (POO), é uma classificação utilizada para linguagens que implementam os conceitos básicos de abstração, encapsulamento, herança e polimorfismo.

Na POO, o programador deve criar um conjunto de classes que irão definir os objetos que serão utilizados no desenvolvimento do software, sendo que cada classe deverá determinar os métodos, que definem as ações que os objetos irão executar, e os atributos, ou seja, as características de um determinado objeto.

2.6.3 Linguagem Delphi

Segundo Anselmo (1997), Delphi é baseado em Object Pascal, que é uma linguagem Orientada a Objeto, pois suporta completamente todos os conceitos de POO, como encapsulamento, herança e polimorfismo.

2.6.3.1 Object Pascal

A linguagem Object Pascal possibilita o desenvolvimento de aplicações utilizando banco de dados Cliente/ Servidor, ponteiros para alocação de memória, criação e reutilização de objetos e bibliotecas dinâmicas. (ANSELMO, 1997)

Na linguagem Object Pascal são utilizadas constantes para armazenar valores fixos, que são declaradas utilizando a palavra reservada *const* seguida do nome da constante e do valor atribuído a ela; as variáveis, isto é, os identificadores que armazenam valores não fixos, têm sua declaração utilizando a palavra reservada *var* antes do nome da variável, e logo depois indica qual o tipo da variável.

São utilizados como estruturas de condição os comandos *if... then... else...*, que escolhe o resultado de uma condição booleana e comando *case...of*, que consiste em uma lista de declarações que satisfaz a condição de um seletor. Como estruturas de repetição são usadas *repeat...until*, que repete um bloco de comandos até que satisfaça certa condição; o *for... to ... do...*, que incrementa uma variável e repete um bloco até que um valor final do intervalo seja atingido pela variável; e o *while...do...*, que repete um bloco até que satisfaça determinada condição.

Blocos de Procedimentos e Funções são utilizados para implementar os métodos vinculados a cada objeto, sendo que os procedimentos não retornam um valor específico, enquanto as Funções são obrigadas a retornar algum valor.

Como Object Pascal é considerada uma Linguagem Orientada a Objeto, ela tem a capacidade de trabalhar com objetos, pertencentes a classes já existentes ou criadas pelo próprio programador. O exemplo a seguir mostra a declaração de um objeto TForm1, que é uma instância da classe TForm:

type

```
TForm1 = class (TForm);
```

2.6.3.2 Borland Delphi 7

O Delphi 7 possui diversas características que se tornam diferenciais do produto em relação aos concorrentes, sendo um compilador rápido e otimizado, que gera executáveis sem a necessidade de utilizar bibliotecas *run time*, resultando em um desempenho muito superior ao de outras ferramentas e tendo a possibilidade de se criar novos componentes, o que o torna bastante flexível e adaptável, facilitando o trabalho do programador.

Logo quando o Delphi é iniciado, alguns objetos são visíveis formando o seu ambiente de trabalho, que são os seguintes: *Form*, *Code Editor*, *Component Palette*, *Object Inspector* e *Object Tree View*.

Os *Forms* são os formulários onde são adicionados e organizados os outros objetos que irão interagir com o usuário; em uma aplicação ele se torna uma janela, ou seja, a interface que o usuário utilizará no software. A figura a seguir mostra um *form*:



Figura 15 - *Form* criado no Delphi (Fonte: Autor)

No Code Editor é onde se gera todo o código-fonte do projeto. Ao ser aberto um novo projeto, na página do editor de códigos do Delphi é gerado automaticamente uma *Unit* com o arquivo de código (.PAS). A figura abaixo mostra um *Code Editor*:

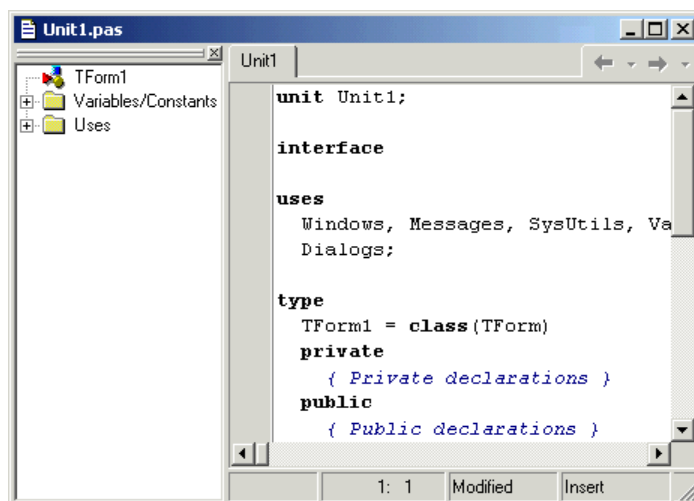


Figura 16 - Code Editor (Fonte: Autor)

Os componentes são elementos que serão utilizados no desenvolvimento da interface da aplicação, eles estão localizados na palheta de componentes (*Component Palette*). Os objetos da palheta são divididos em grupos de acordo com sua funcionalidade, conforme mostra a figura 17. A linguagem Delphi torna possível a criação de novos componentes, assim como a remoção, adição de componentes e criação de novas páginas na palheta de componentes.



Figura 17 - Component Palette (Fonte: Autor)

O *Object Inspector* é composto por duas páginas: *Properties* e *Events*, onde são exibidos as propriedades e os eventos, respectivamente, que podem ser utilizados para personalizar os componentes. O *Object TreeView*, tem como sua principal função exibir os componentes do formulário de uma forma hierárquica, com ele é possível ter uma visão mais ampla dos componentes tornando mais simples alterar a hierarquia entre os componentes e encontrá-los mais rapidamente. Ambos são mostrados na figura seguinte:

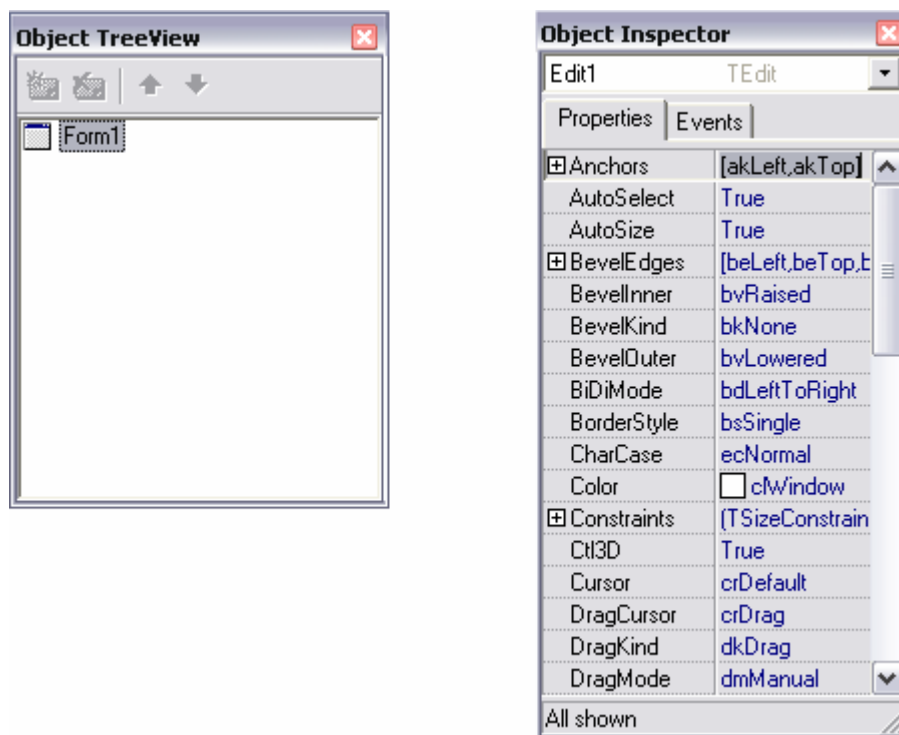


Figura 18 - Object TreeView e Object Inspector (Fonte: Autor)

2.6.3.3 Conexão com o Banco de Dados

De acordo com Oliveira (2000), quando uma aplicação que utiliza a arquitetura Cliente/ Servidor é desenvolvida, a primeira tarefa a ser cumprida é a conexão com o Banco de Dados.

Existem inúmeras tecnologias de acesso ao banco de dados que o Delphi é capaz de suportar, entretanto a Borland desenvolveu uma tecnologia própria, compatível com as plataformas Windows e Linux, o dbExpress (IMASTERS, 2008).

O dbExpress é compatível com plataformas Windows e Linux e tem como uma de suas principais vantagens o baixo overhead adicionado às operações no banco, o que torna o desempenho extremamente alto, trabalhando com a tecnologia ClientDataSet do Delphi.

No Delphi, os componentes de conexão com o banco de dados são adicionados em um DataModule. O primeiro componente que deverá ser inserido no DataModule é o SQLConnection, nativo da palheta dbExpress. Com um clique duplo sobre ele é possível criar uma nova conexão, a partir da tela mostrada na figura a seguir:

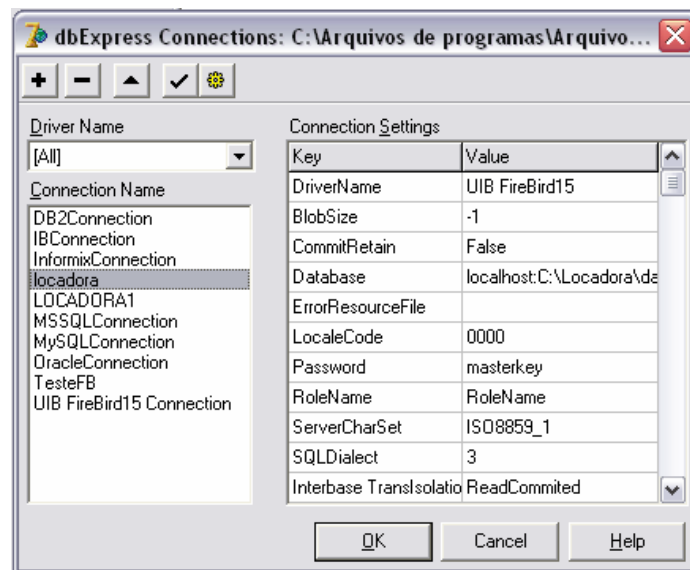


Figura 19 - Conexão no SqlConnection (Fonte: Autor)

Após configurar a conexão, deverá ser inserido no DataModule o componente SQLDataSet, também da palheta dbExpress, este componente é ligado na conexão criada através da propriedade SqlConnection. Na propriedade Command Text do SQLDataSet, pode ser criado um código SQL como um select, que irá trazer os campos da tabela que serão utilizados na aplicação.

Na palheta DataAccess se encontram os outros componentes que serão utilizados, como o DataSetProvider, o ClientDataSet e o DataSource, este será adicionado diretamente no *form* da aplicação. A figura seguinte mostra um DataModule com os componentes necessários para uma conexão simples:

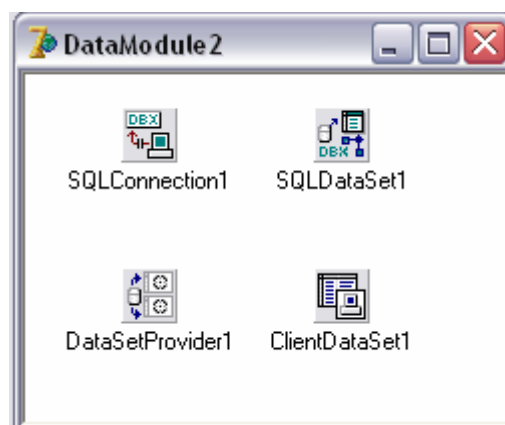


Figura 20 - DataModule e os Componentes de Conexão (Fonte: Autor)

O DataSetProvider é ligado no SQLDataSet através da propriedade DataSet, depois liga-se o ClientDataSet no DataSetProvider indicando-o na propriedade Provider Name .Depois de ligar os componentes, com um duplo clique no ClientDataSet, abre-se uma janela onde serão incluídos os campos trazidos pelo select, clicando com o botão direito e escolhendo a opção Add all Fields, conforme a figura seguinte:

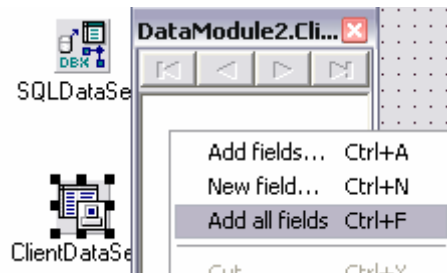


Figura 21 - Adicionando campos no ClientDataSet (Fonte: Autor)

Com a adição dos campos no ClientDataSet, o componente DataSource necessita ser ligado a ele, pois é o último nível antes que seja ligado à aplicação.

Os componentes da palheta DataControls, como DBEdit, DBText, DBGrid são os componentes do Delphi capazes de serem ligados ao banco de dados. Através da propriedade DataSource, é indicado qual o DataSource que será utilizado pra conectar ao banco e na propriedade DataField é informado a qual campo o componente DataControl estará ligado, como mostra a figura.

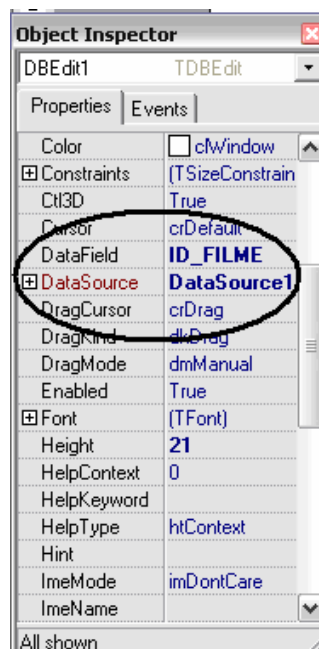


Figura 22 - Ligação no DataSource (Fonte: Autor)

3 DESENVOLVIMENTO

3.1 Modelos de Processo de Engenharia de Software

A engenharia de software teve um papel importante no desenvolvimento do sistema, onde se obteve uma organização através de cronogramas para dividir as tarefas com determinadas metas e informações coletadas.

Foi utilizado um modelo de processo evolucionário para organizar o processo de desenvolvimento do software: o modelo de prototipagem. Este modelo foi escolhido por oferecer uma melhor abordagem sobre formas de IHM que deverão ser assumidas, testes de eficiência de algoritmos, etc. A figura a seguir ilustra o funcionamento do paradigma de prototipagem:

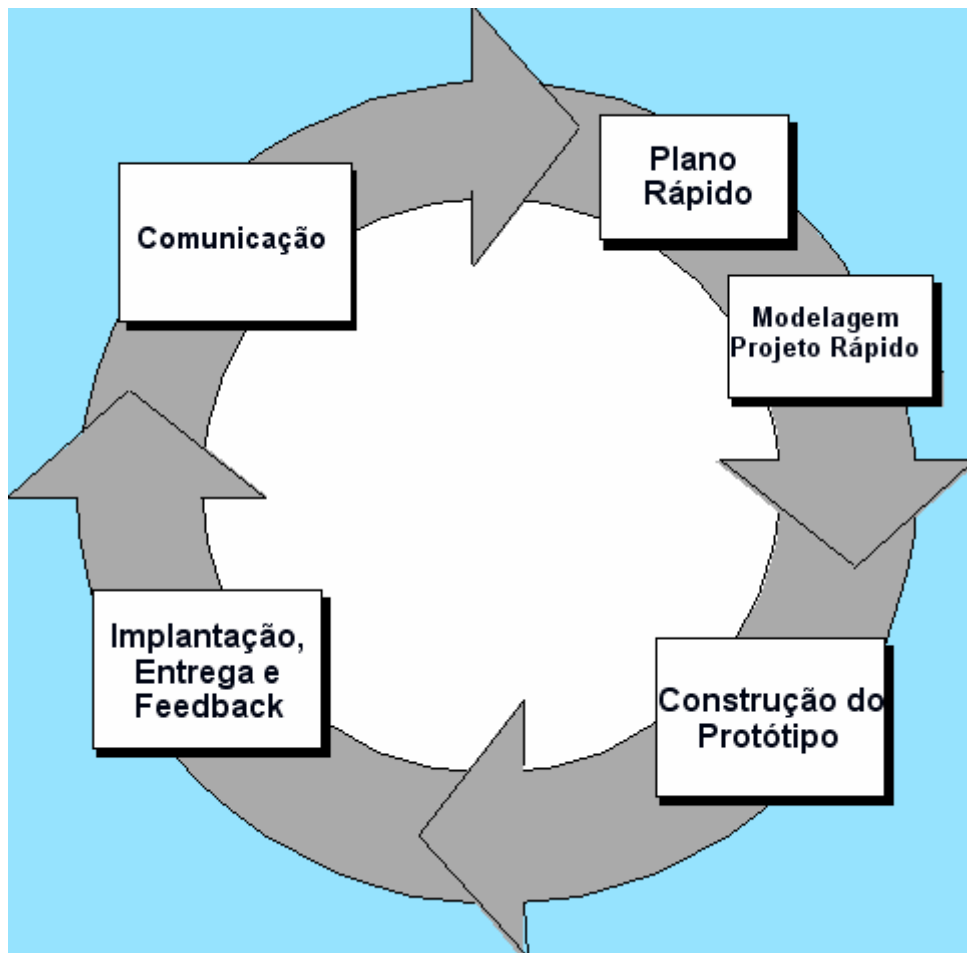


Figura 23 - Modelo de Prototipagem (Fonte: PRESSMAN, 1995)

3.1.1 Comunicação

Na fase de comunicação, foi feita uma reunião com o cliente, no caso o dono de uma vídeo-locadora que não ainda não é informatizada, para a definição do objetivo geral do software, coleta de requisitos e identificação das necessidades conhecidas. Nessa reunião definiu-se que o software seria voltado para utilização dos clientes da locadora e que necessitaria da aplicação de conceitos de usabilidade para a construção de sua interface.

3.1.2 Criação e modelagem do projeto rápido

Uma iteração, ou seja, repetição da prototipagem foi planejado de forma rápida, ocorrendo a modelagem. Na modelagem do projeto rápido, foi representado os aspectos que serão visíveis para o usuário, como o *layout* da interface da aplicação de locação, mostrando a navegabilidade do software e utilizando os recursos de IHM.

3.1.3 Construção do protótipo

O resultado do projeto rápido é um protótipo do software para vídeo-locadoras, que inicialmente apenas faz as operações de busca, locação e reserva de filmes, que pode ser implantado no estabelecimento para que seja testado e avaliado pelo usuário.

3.1.4 Implantação, Entrega e Feedback

Na implantação e entrega do protótipo para o cliente, o protótipo foi avaliado e foi retornado um *feedback* por parte do cliente contendo informações sobre esta avaliação prévia do software, para que fosse possível refinar os requisitos e ajustar o software para que satisfaça as necessidades do usuário, ocorrendo a iteração, isto é, iniciando novamente com a comunicação com o cliente e repetindo este ciclo, com o objetivo de que seja desenvolvido um software que atenda totalmente o que o usuário precisa.

O Anexo 1 mostra o *feedback* retornado pelo dono da locadora, que testou e avaliou o protótipo do software. Foi feita uma coleta de novos requisitos, entrando mais uma vez no ciclo a partir da comunicação com o cliente.

3.2 Modelagem UML

Foram utilizados os conceitos de UML no desenvolvimento da modelagem do protótipo do sistema de locadora, onde foram aplicados os diagramas UML.

3.2.1 Ferramenta utilizada

Para desenvolver os diagramas UML com precisão, foi utilizado o software Visual Paradigm, uma ferramenta CASE, voltada para UML, que é capaz de criar diversos diagramas, como diagramas de caso de uso, de atividades, de classes, de sequência, dentre outros. A imagem seguinte mostra a interface do Visual Paradigm:

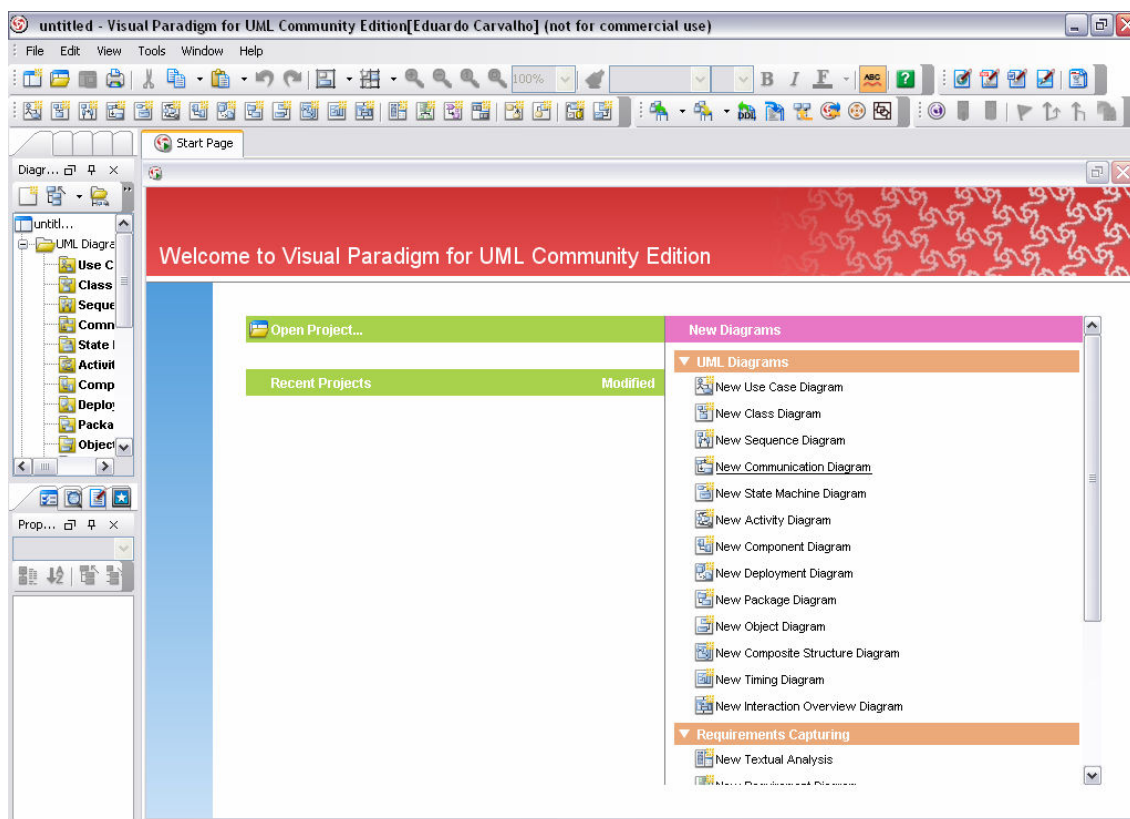


Figura 24 - Visual Paradigm Community Edition.

O software possui uma versão para testes, que expira em 30 dias, a Professional Edition for Windows 6.3 e uma versão gratuita com recursos limitados, a Community Edition. A versão escolhida foi a Community Edition, pois os recursos que ela apresenta atendem totalmente as necessidades do projeto em desenvolvimento.

3.2.2 Construção dos diagramas

Cada diagrama possui uma visão parcial do sistema, que estando em conjunto, formam um todo integrado e coerente. Na fase de modelagem do sistema, foram construídos três diagramas: o diagrama de caso de uso, o diagrama de atividade e o diagrama de classe.

3.2.2.1 Diagrama de caso de uso

O diagrama de caso de uso desse sistema mostra as ações do cliente em um cenário de locadora, onde ao interagir com o sistema ele faz a reserva do filme obtendo através da opção de reserva as extensões agregadas, opções essas que são: obter tipo de preço, a censura do filme, reserva por categoria, tipo filme e também através do caso de uso locação tem se a extensão locação por itens, ou seja, terá uma opção dentro de locação. Cada ação representa um caso de uso dentro do cenário da Locadora.

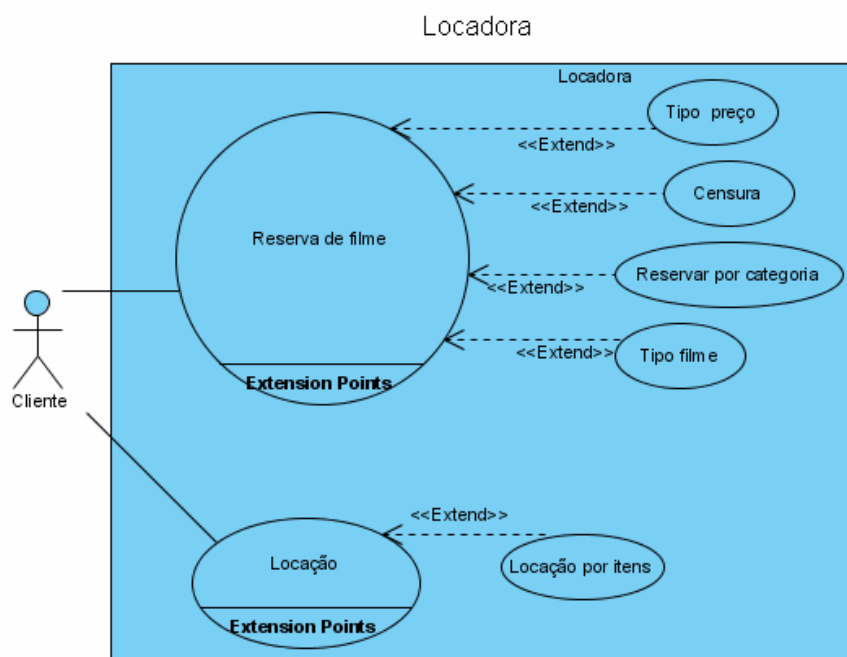


Figura 25 - Diagrama de Caso de Uso

3.2.2.2 Diagrama de classe

O diagrama de classe é a parte do trabalho que mostra a interação dos objetos que compõem o sistema, onde teremos o objeto cliente fazendo reserva de filmes, sendo que cada um desses objetos possuem seus devidos atributos, esses atributos vão

classificar cada um deles, eles também possuem métodos para desenvolver sua função específica no sistema. Na figura abaixo podemos ver também a multiplicidade dos relacionamentos, os asteriscos significam muitos para um ou muitos para muitos, onde um cliente poderá fazer várias reservas de vários filmes, sendo que vários filmes podem estar em uma mesma categoria ou censura e vários filmes podem ter o mesmo preço. Os objetos categoria, censura, tipo preço e tipo filme são uma composição da classe filmes, sendo que se a classe filme for excluída, automaticamente as outras classes que estão unidas a ela também serão.

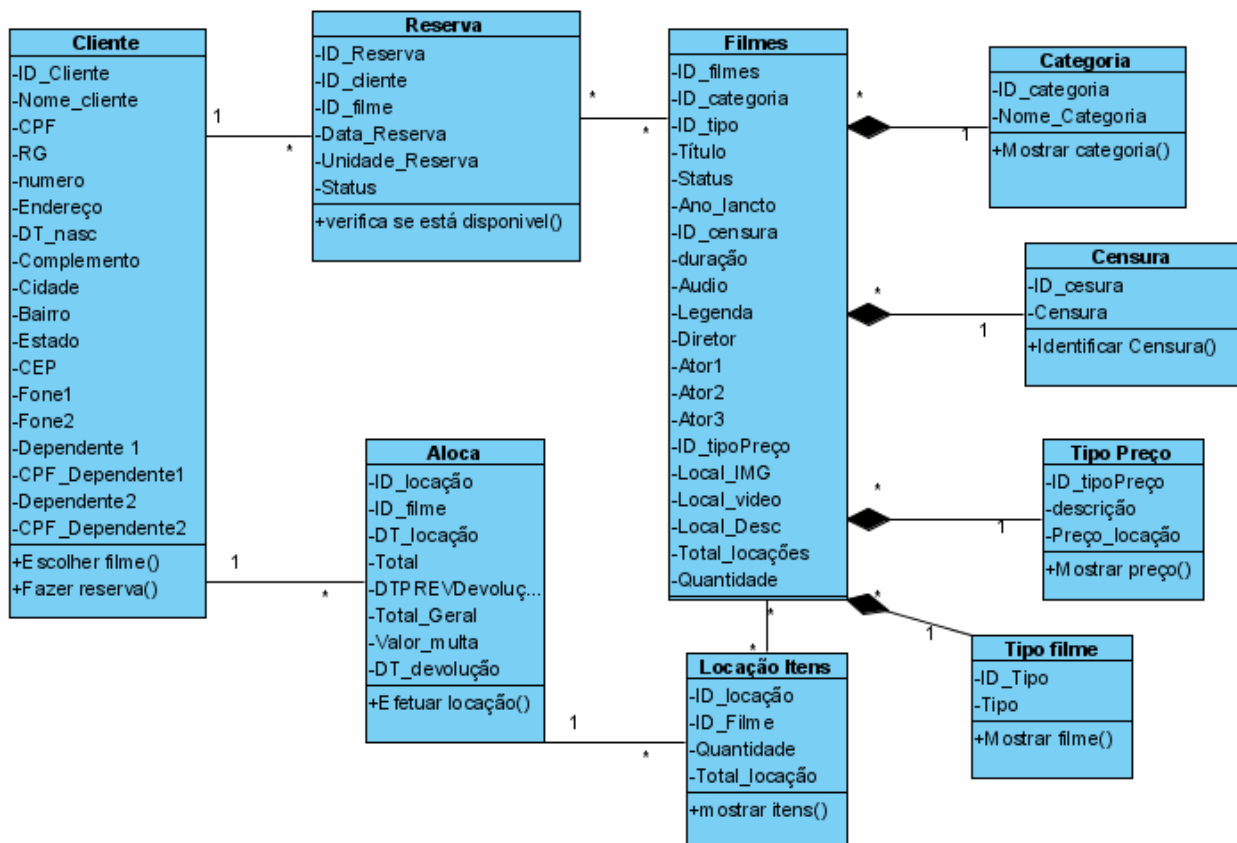


Figura 26 - Diagrama de Classe

3.2.2.3 Diagrama de atividades

Diagrama de atividade ilustra as etapas das tarefas feitas paralelamente pelo usuário no sistema, o círculo preto significa o início, logo o cliente efetua a primeira ação entrando no sistema com sua senha em seguida ele efetua a busca de filmes, escolhe o filme desejado. Logo depois entra no triângulo de decisão tendo as opções de fazer a locação deste filme, reservá-lo ou simplesmente cancelar a operação e sair do

sistema. Se a opção escolhida for fazer a locação, ele poderá alugar outro filme ou concluir a locação. Caso escolha alugar outro filme, deverá passar por todos os passos anteriores, e caso escolha concluir a locação, o cliente efetua o pagamento e termina a atividade de locação.

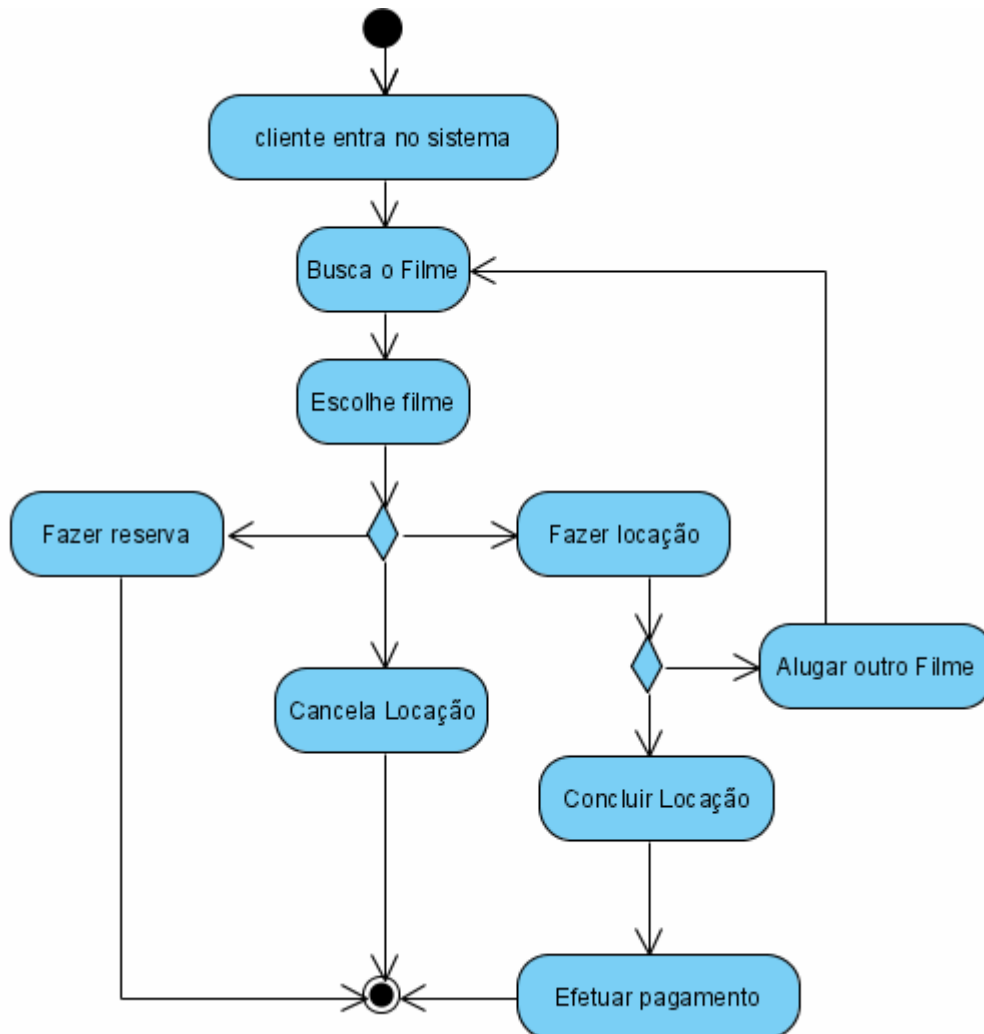


Figura 27 - Diagrama de Atividade

3.3 Construção do banco de dados

O banco de dados que irá armazenar e gerenciar os dados da aplicação foi construído utilizando o modelo relacional. No desenvolvimento da aplicação foram aplicados os conceitos de banco de dados com a ajuda de algumas ferramentas.

3.3.1 Ferramentas utilizadas

O SGBD utilizado na aplicação é o Firebird 2.0, o principal motivo pelo qual este SGDB foi escolhido, foi as inúmeras vantagens que ele apresenta, como por exemplo, o fato de ser uma ferramenta OpenSource e free; a simplicidade de instalação, configuração e administração; a compatibilidade total com aplicativos implementados em Delphi.

Para o gerenciamento das tabelas e dos dados em geral do BD foi utilizado o IBExpert, que não é uma ferramenta gratuita, mas possui uma versão *trial*, o IBExpert Trial Version, que é compatível com Firebird 2.0. A figura a seguir mostra a interface do IBExpert.

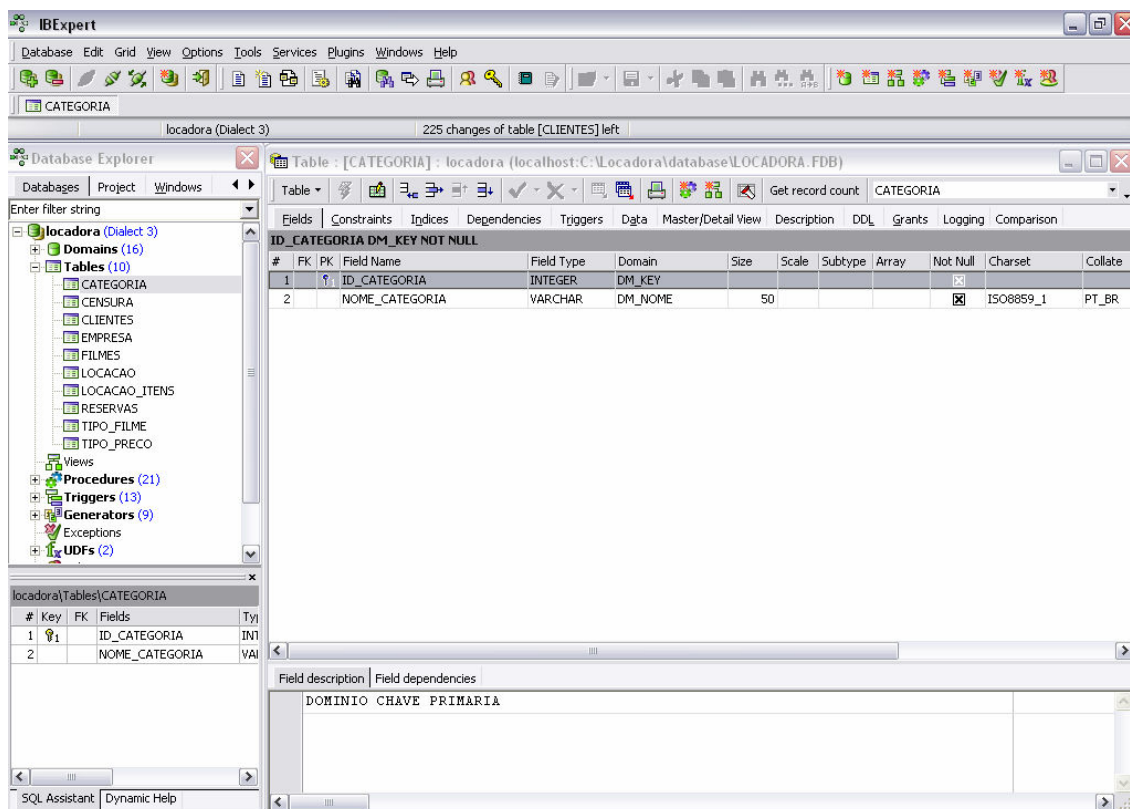


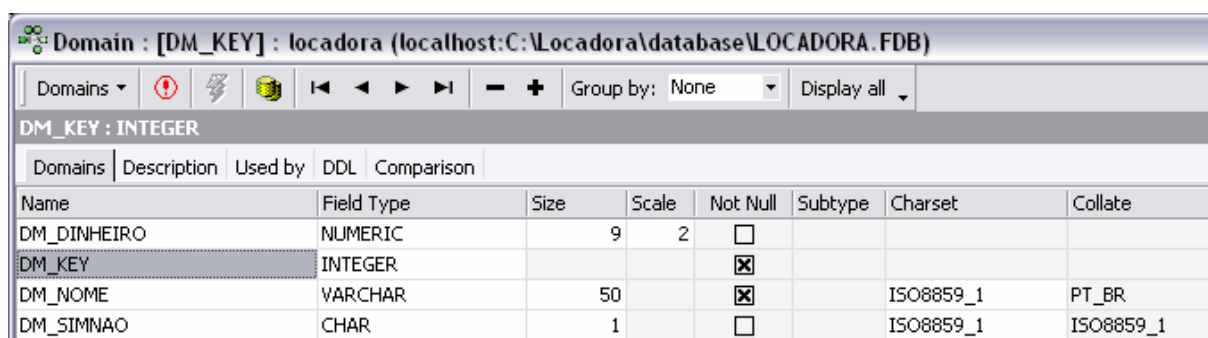
Figura 28 - Interface do IBExpert

A utilização do IBExpert para administrar o banco de dados facilita e agiliza as operações nas tabelas, como consultas simples e complexas, criação de stored procedures, triggers, generators, dentre outros.

A biblioteca utilizada para acesso aos dados no Firebird é a fbclient.dll, que é uma biblioteca específica para trabalhar com Firebird.

3.3.2 Criação de domínios e tabelas

Utilizando a ferramenta IBExpert, foram enviados primeiramente os domínios, que tem a finalidade de simplificar a identificação dos tipos de cada campo das tabelas, como por exemplo, domínio para chaves primárias, domínio para nomes, domínio para valores do tipo moeda: fazendo uma padronização dos tipos e agilizando a construção do BD. A figura 28 mostra alguns dos domínios criados no banco de dados da aplicação:



Name	Field Type	Size	Scale	Not Null	Subtype	Charset	Collate
DM_DINHEIRO	NUMERIC	9	2	<input type="checkbox"/>			
DM_KEY	INTEGER			<input checked="" type="checkbox"/>			
DM_NOME	VARCHAR	50		<input checked="" type="checkbox"/>		ISO8859_1	PT_BR
DM_SIMNAO	CHAR	1		<input type="checkbox"/>		ISO8859_1	ISO8859_1

Figura 29 - Domínios criados no banco

Depois da definição dos domínios, foram criadas nove tabelas:

- a) **CLIENTES**: armazena os clientes do estabelecimento;
- b) **FILMES**: armazena todos os filmes que a locadora possui;
- c) **CATEGORIA**: categorias ou gêneros dos filmes como, por exemplo: comédia, ação, terror, etc.
- d) **CENSURA**: mostra a censura do filme, ou seja, a partir de qual classificação etária ele é voltado, como 12 anos, 18 anos, livre, etc.
- e) **TIPO_PRECO**: o tipo de faixa de preço que o filme se encaixa, como por exemplo: promoção, lançamento, preço padrão da locadora;
- f) **TIPO_FILME**: o tipo de unidade do filme se é um DVD (unitário), se é uma coleção com três DVD's;
- g) **RESERVAS**: é a tabela que armazena as reservas de um filme, quando não se encontra disponível, o cliente pode reservá-lo para alugá-lo quando estiver disponível.

h) **LOCACAO**: indica qual cliente efetuou a locação, sendo que os filmes alugados são armazenados na tabela **LOCACAO_ITENS**.

i) **LOCACAO_ITENS**: indica os filmes alugados.

A figura abaixo mostra todas as tabelas do banco, indicando seus campos, chaves primárias e estrangeiras e os relacionamentos existentes entre elas.

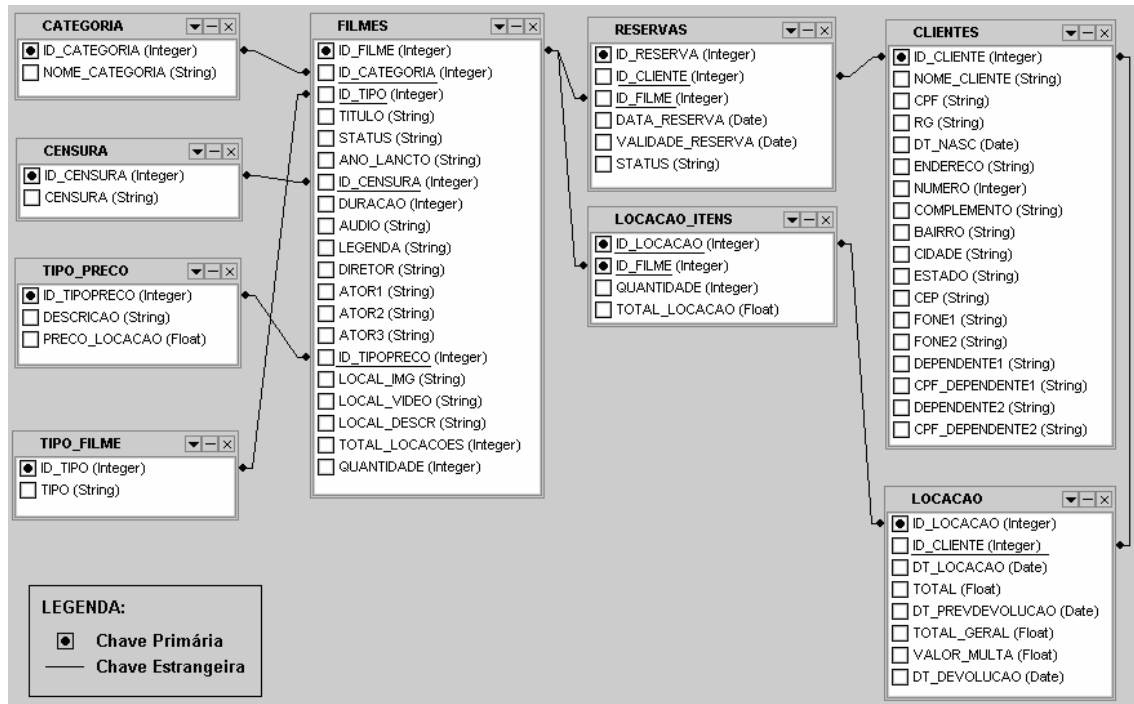


Figura 30 - Tabelas e relacionamentos do BD

3.3.3 Geração de autonumeração

Com as tabela já criadas, o próximo passo foi criar as *SEQUENCES*, que no Firebird 2.0, são semelhantes aos *GENERATORS* do Firebird 1.5, gerando uma seqüência numérica para os campos chave primária, que necessitam do uso de autonumeração. O código SQL a seguir mostra a criação de uma *sequence* utilizada no BD, que tem a finalidade de armazenar o valor atual da seqüência numérica da tabela Filmes:

```
CREATE SEQUENCE SEQ_FILMES_ID
```

Após a criação das sequences, foi preciso cirar os gatilhos, ou triggers, para que antes da inserção dos elementos nas tabelas, sejam gerados um novo id, utilizando as sequences criadas anteriormente e incrementando o valor 1. A seguir é mostrado o

código SQL utilizado para criar uma trigger para auto incremento no campo ID_FILME da tabela FILMES.

```
CREATE TRIGGER FILMES_BI FOR FILMES
ACTIVE BEFORE INSERT POSITION 0
AS
BEGIN
    IF (NEW.ID_FILME IS NULL) THEN
        NEW.ID_FILME = GEN_ID (SEQ_FILMES_ID, 1);
    END
```

3.3.4 Utilização de Stored Procedures

Foram criadas também stored procedures para as operações de inserção/atualização e para inclusão. O motivo pelo qual foi decidido fazer estas operações dentro do banco de dados, foi pelo fato de que desta maneira aumenta-se a performance da aplicação, executando-a mais rapidamente do que se todas estas operações fossem feitas na própria aplicação e eliminando os riscos de inconsistência no banco de dados.

A stored procedure mostrada no código a seguir executa as operações de inserção e atualização na tabela CATEGORIAS, que é onde são cadastrados as categorias, ou gêneros, dos filmes. Seu funcionamento acontece da seguinte forma: se o parâmetro passado já existir a operação executada é a de atualização, e se não existir efetua a inclusão.

```
CREATE PROCEDURE CATEGORA_IU (
    ID_CATEGORIA INTEGER, NOME_CATEGORIA VARCHAR (50))
AS BEGIN
    IF (EXISTS (SELECT ID_CATEGORIA FROM CATEGORIA WHERE
        (ID_CATEGORIA = :ID_CATEGORIA))) THEN
        UPDATE CATEGORIA
        SET NOME_CATEGORIA = :NOME_CATEGORIA
        WHERE (ID_CATEGORIA = :ID_CATEGORIA);
    ELSE
        INSERT INTO CATEGORIA (
            ID_CATEGORIA,
            NOME_CATEGORIA)
```

```
VALUES (
    NEXT VALUE FOR SEQ_CATEGORIA_ID,
    :NOME_CATEGORIA);
END
```

As exclusões dos registros de uma tabela são feitas no banco de dados conforme o código seguinte, que mostra a exclusão de uma reserva do banco de dados.

```
CREATE PROCEDURE RESERVAS_D (
    ID_RESERVA INTEGER)
AS
BEGIN
    DELETE FROM RESERVAS
    WHERE (ID_RESERVA = :ID_RESERVA);
END
```

3.3.5 Consultas à partir de Views

Para facilitar as consultas que serão utilizadas na aplicação, foram criadas views, uma delas é a view RESERVAS_CLIENTE, que a partir do campo ID_CLIENTE, traz todas as reservas de filmes que um determinado cliente efetuou, e que ainda estão em aberto. O código a seguir mostra o funcionamento da view.

```
CREATE VIEW RESERVAS_CLIENTE ( ID_RESERVA, ID_CLIENTE, ID_FILME,
    TITULO, DATA_RESERVA, STATUS)
AS
SELECT    RESERVAS.ID_RESERVA,
    RESERVAS.ID_CLIENTE,
    RESERVAS.ID_FILME,
    FILMES.TITULO,
    RESERVAS.DATA_RESERVA,
    RESERVAS.STATUS
FROM      RESERVAS INNER JOIN FILMES ON (RESERVAS.ID_FILME =
FILMES.ID_FILME)
WHERE     (RESERVAS.STATUS = 'A');
END
```

3.4 Implementação da Aplicação

Após a criação do banco de dados e a definição dos mecanismos de exclusão, alteração e inclusão dos dados no banco, foi iniciada a implementação do software.

O foco principal do projeto é desenvolver um software em que o próprio cliente irá efetuar a busca, locação e reserva dos filmes. Portanto a construção do software foi focada nestas operações relacionadas à locação.

Como uma vídeo-locadora necessita de um gerenciamento mais complexo, uma área administrativa, onde serão feitos os cadastros dos clientes, filmes, gêneros, e as devoluções dos filmes, poderá ser construída, utilizando a mesma conexão com o banco; esta será citada como uma perspectiva de trabalhos futuros.

3.4.1 Ferramenta Utilizada

A linguagem escolhida para ser utilizada na implementação do software foi a linguagem Delphi. Um dos principais motivos para a utilização desta linguagem foi pelo fato de ser baseada em Object Pascal, uma linguagem de alto nível, que tem uma sintaxe de fácil entendimento, muito próxima da linguagem humana; e também por ter uma grande quantidade de material de aprendizado disponível na internet. O compilador utilizado no projeto foi o Borland Delphi 7.

3.4.2 Conexão com o BD

Primeiramente foram criado um DataModule, e nele inseridos todos os componentes necessários para efetuar a conexão com o banco de dados. Além dos componentes citados anteriormente, foi utilizado o componente SQLStoredProc que será ligado diretamente nas Stored Procedures (SP) do banco.

A conexão com o banco de dados foi criada utilizando o driver UIBFirebird15, que é um driver específico para trabalhar com os componentes de conexão da palheta dbExpress e com banco de dados Firebird.

Depois de configurar a conexão com o banco no SQLConnection, os demais componentes foram conectados devidamente entre si, sendo que todos os ClientDataSet do DataModule são ligados em um DataSetProvider genérico, assim concluindo a conexão com o banco de dados. A figura a seguir apresenta os componentes de conexão no DataModule:

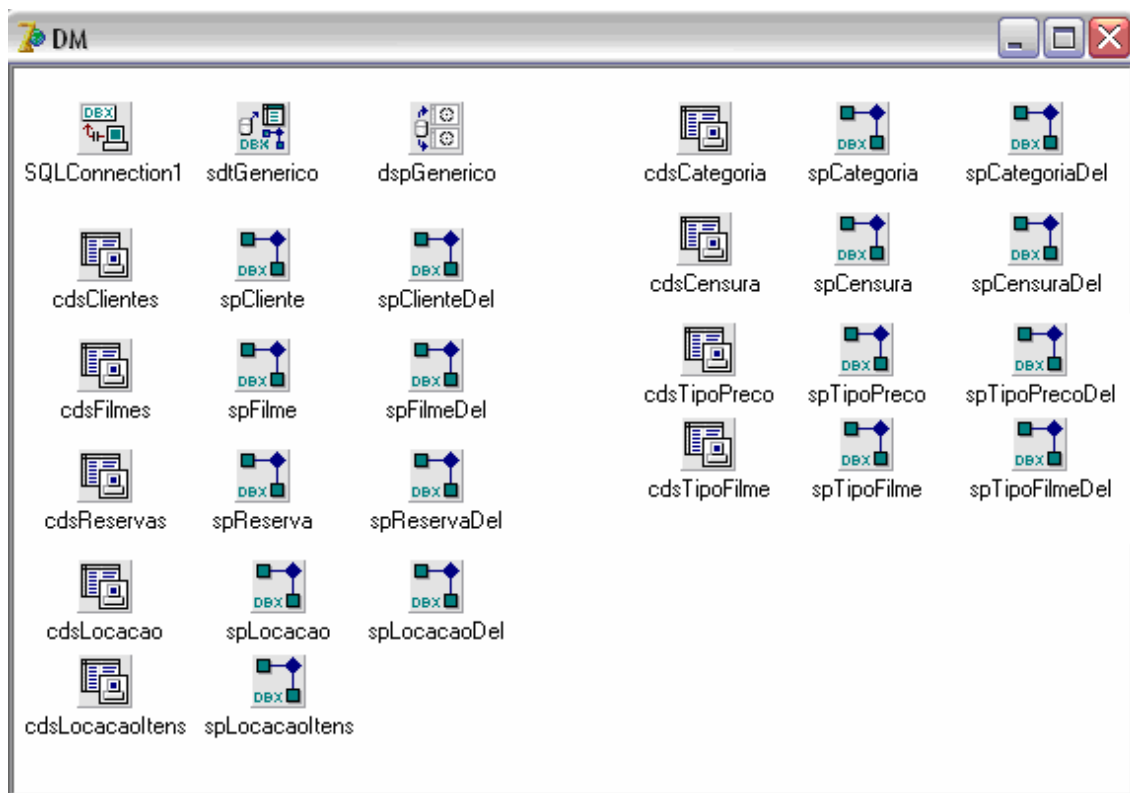


Figura 31 – DataModule e demais componentes de conexão

Como os procedimentos de exclusão, alteração e inclusão dos dados foram criados dentro do próprio banco, na aplicação apenas serão passados os parâmetros necessários para que as stored procedures do BD façam estas operações.

Ao concluir a conexão, foram criados os métodos para as operações de inclusão, alteração e exclusão, onde são passados os parâmetros. O código a seguir mostra um fragmento do método para inclusão e atualização na tabela de reservas:

with spReserva do

begin

Params[0].AsInteger := cdsReservasID_RESERVA.AsInteger;

params[1].AsInteger := cdsReservasID_CLIENTE.AsInteger;

params[2].AsInteger := cdsReservasID_FILME.AsInteger;

params[3].AsDate := cdsReservasDATA_RESERVA.AsDateTime;

Params[4].AsDate := cdsReservasVALIDADE_RESERVA.AsDateTime;

params[5].AsString := cdsReservasSTATUS.AsString;

ExecProc;

end;

O métodos que fazem as passagens de parâmetros para inclusão e alteração, são implementados nos eventos BeforePost de cada ClientDataSet, ou seja, antes que as modificações sejam gravadas no banco.

3.4.3 Interface da Aplicação

Durante o desenvolvimento da interface, se obteve uma preocupação especial para que a interface satisfaça as condições de usabilidade, tornando a interação com o usuário fácil e satisfatória, diminuindo a taxa de erros. Como a interface foi projetada para ser simples e clara o máximo possível, conseqüentemente, sua utilização terá fácil aprendizado e durante as várias vezes que o usuário a utilizará, suas funcionalidades serão memorizadas.

Como a aplicação é voltada para utilização dos clientes da locadora, ela foi desenvolvida de um modo que a interação com o usuário possa ser feita de várias formas, como a forma tradicional, com teclado e mouse; e também com a utilização da tecnologia touch screen (monitores ou lentes sensíveis ao toque). Pensando nesta possibilidade, no desenvolvimento da interface, foram utilizados recursos que tornam possíveis a utilização dos recursos touch screen para interação com o usuário, como a criação de botões maiores e um teclado virtual.

A interface da aplicação foi desenvolvida no Delphi, e foi baseada basicamente em um form principal, onde foram adicionados os componentes necessários para tornar o software funcional. Foram utilizados os componentes ActionList, que tem a finalidade de listar as ações que poderão ser feitas durante um evento de um objeto; uma ImageList, onde armazena-se as imagens utilizadas no sistema; e os DataSources que farão uma ponte de ligação entre a aplicação e o banco de dados.

Como a aplicação possui várias telas diferentes, para continuar utilizando apenas um form, foi necessário utilizar o componente PageControl, nativo do Delphi 7, através do PageControl podem ser criadas inúmeras páginas na aplicação conforme as necessidades existentes.

O PageControl principal, foi renomeado para PagePrincipal e nele foram criados dois TabSheets, isto é, duas páginas que foram apresentadas como pgPesquisa, onde são feitas as buscas de filmes e pgLocacao, onde é mostrado os detalhes do filme, sinopse, trailer e opções de locação ou reserva.

3.4.4 Tela de Busca

Primeiramente, na TabSheet de busca, a pgPesquisa, foram adicionados os botões relacionados a cada tipo de busca, como pesquisa pelo título do filme, pelo gênero, pelos atores que estrelam o filme, pelo diretor do filme, pelo ano de lançamento; seguindo o mesmo padrão foram adicionados botões para busca avançada, como busca pelos lançamentos, pelos filmes mais alugados e pelos filmes em promoção. Estes botões foram criados a partir dos componentes BitBtn.

Foram utilizados os componentes Panel (painéis), que auxiliam na organização da interface separando certos objetos. Dentro de um dos Panels da tela de busca foi inserido outro PageControl, onde foram criados TabSheets para cada tipo de busca. Este PageControl foi denominado pageBusca e em cada TabSheet foram adicionados os componentes para a exibição dos dados pesquisados, que são os DBGrids e os objetos que onde serão digitados os parâmetros de pesquisa, neste caso os Edits.

A figura seguinte mostra a tela de busca da aplicação:



Figura 32 - Tela de busca

Para que a interação com o usuário seja feita independente dos meios que serão usados para utilizar a aplicação, foi criado um teclado virtual, que é composto por um

teclado numérico, um teclado de letras maiúsculas, pontuação, e outros símbolos e um teclado de funções, como as teclas espaço e apagar. Todas as funcionalidades baseadas nos conceitos de Usabilidade, principalmente na facilidade de aprendizado e na eficiência e satisfação do usuário.

A tela de busca por título é a tela padrão de busca, sendo que o cliente pode escolher o tipo de busca desejada. Quando o cliente escolhe fazer um tipo de busca, o componente ClientDataSet ligado no DataSource utilizado é fechado e depois aberto, para se caso algum filme for incluído na maquina do servidor, este poderá aparecer atualizado na aplicação do cliente, os teclados ficam visíveis e os índices dos PageControls mudam de acordo com a escolha.

Clicando no botão de busca desejado, o DBGrid é preenchido com os dados relacionados à pesquisa. O preenchimento do grid é implementado no evento OnChange do Edit, onde enquanto algum texto é digitado o DBGrid é preenchido em tempo real e o resultado do texto buscado é indicado no próprio DBGrid. Quando se escolhe um gênero como parâmetro de busca, os filmes relacionados a este gênero são filtrados, aparecendo no grid de filmes.

A propriedade Filter do ClientDataSet possibilita a implementação desta operação, pelo fato de quando o usuário der um duplo clique no grid de gêneros, uma variável receberá o valor do campo ID_GENERO do gênero desejado, logo depois o Filter do ClientDataSet referente aos filmes receberá o valor armazenado na variável, assim filtrando todos os filmes do gênero escolhido. O código a seguir mostra como a propriedade Filter do Client é utilizada:

```
dmLocacao.CdsBuscaFilme.Filter := 'ID_CATEGORIA = '+ IntToStr(codGenero);  
ShowMessage(dmLocacao.CdsBuscaFilme.Filter);  
dmLocacao.CdsBuscaFilme.Filtered := True;
```

Os outros tipos de busca também utilizam a propriedade Filter, como a busca por ator/ atriz, por diretor, ano de lançamento e as buscas avançadas.

Após efetuar a busca, o cliente pode dar um clique duplo no filme escolhido, esta ação seta o TabSheet da página de detalhes do filme, ou seja, a página de locações. Este método é implementado no evento duplo clique do DBGrid de filmes, onde é verificado se existem trailer e sinopse cadastrada no banco.

Para visualizar o código-fonte referente às funcionalidades do teclado virtual consultar o Anexo 2.

3.4.5 Tela de Detalhes do Filme

Após a indicação do filme escolhido, a página índice do PageControl torna-se a tela onde são mostrados os detalhes do filme como os dados gerais, a sinopse e o trailer. A interface desta tela também foi projetada levando em conta os critérios de usabilidade, como a preocupação de que o cliente não utilize as tarefas de um modo não esperado, por isso o teclado não está disponível nesta tela, pois o usuário não terá a necessidade de utilizá-lo; outro aspecto da usabilidade é a facilidade de utilização e memorização do sistema, pelo fato da utilização de recursos audiovisuais, como a exibição do trailer do filme, o que torna a interação entre o usuário e a aplicação mais natural, agradável e com o objetivo de ser o mais satisfatória possível.

Na construção da tela de detalhes do filme, foram utilizados botões para visualizar a sinopse e o trailer, e também para efetuar as operações de locação e reserva; e mais um PageControl, para organizar todos os detalhes do filme por páginas, economizando espaço na tela e tornando a interface mais simples e clara. A seguir é mostrada a imagem da tela de detalhes do filme e suas funcionalidades.

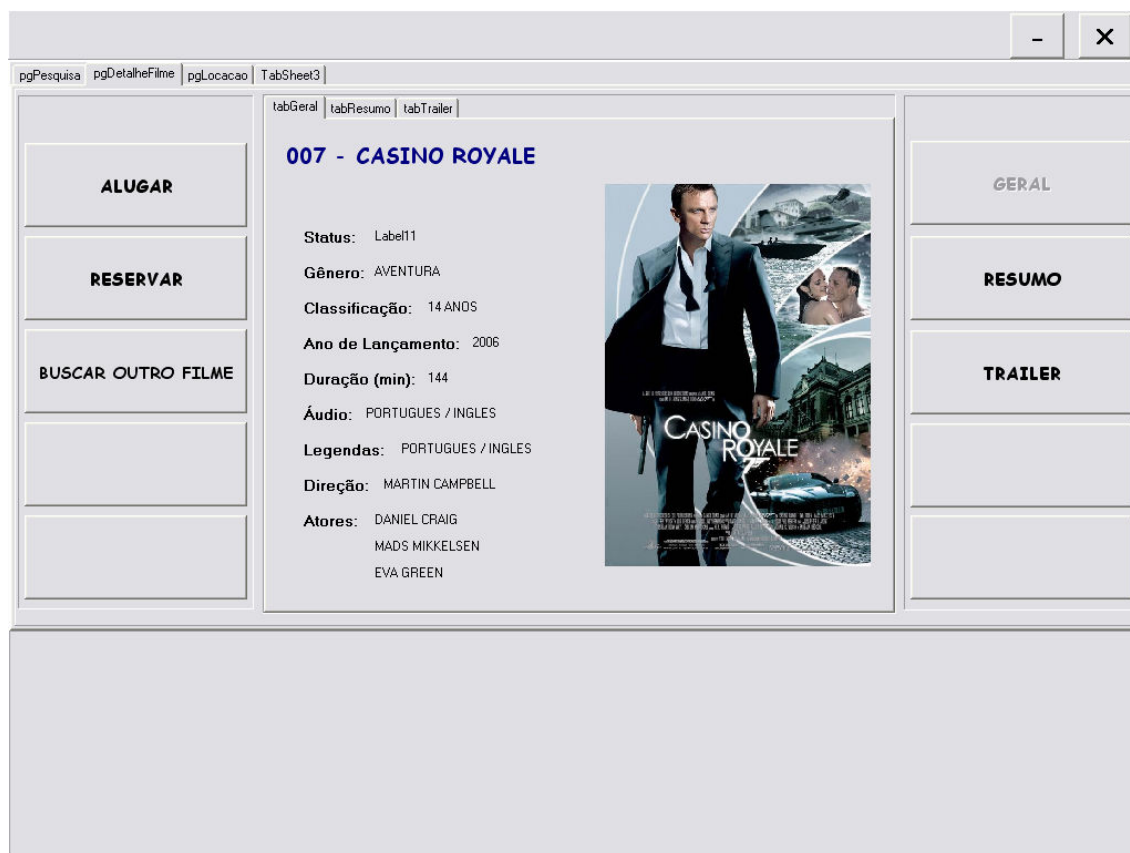


Figura 33 - Tela de detalhes do filme

A capa do filme é uma imagem armazenada no diretório da aplicação, sendo que no banco é gravada o caminho do diretório. Quando o filme é selecionado, é feita uma verificação se existe ou não no banco o caminho da imagem da capa do filme. Se existir, carrega a imagem da capa, senão carrega uma imagem padrão. O código a seguir mostra como é feita esta verificação:

```
if (dmLocacao.cdsBuscaFilme.fieldByName('LOCAL_IMG').AsString) <> " then
    imgCapaFilme.Picture.LoadFromFile(dmLocacao.cdsBuscaFilme.fieldByName
('LOCAL_IMG').AsString)
else imgCapaFilme.Picture.LoadFromFile('C:\Locadora\capa\default.jpg');
```

A visualização da sinopse é feita através do componente RichEdit onde são exibidos textos no formato Rich Text (.rtf), estes arquivos também são armazenados no diretório da aplicação e no banco de dados está gravado o caminho do diretório e não o próprio arquivo. A verificação é feita de um modo semelhante ao da imagem, apenas mudando o campo da tabela FILMES que deve ser verificado. A figura a seguir mostra a tela de detalhes do filme na parte de sinopse:

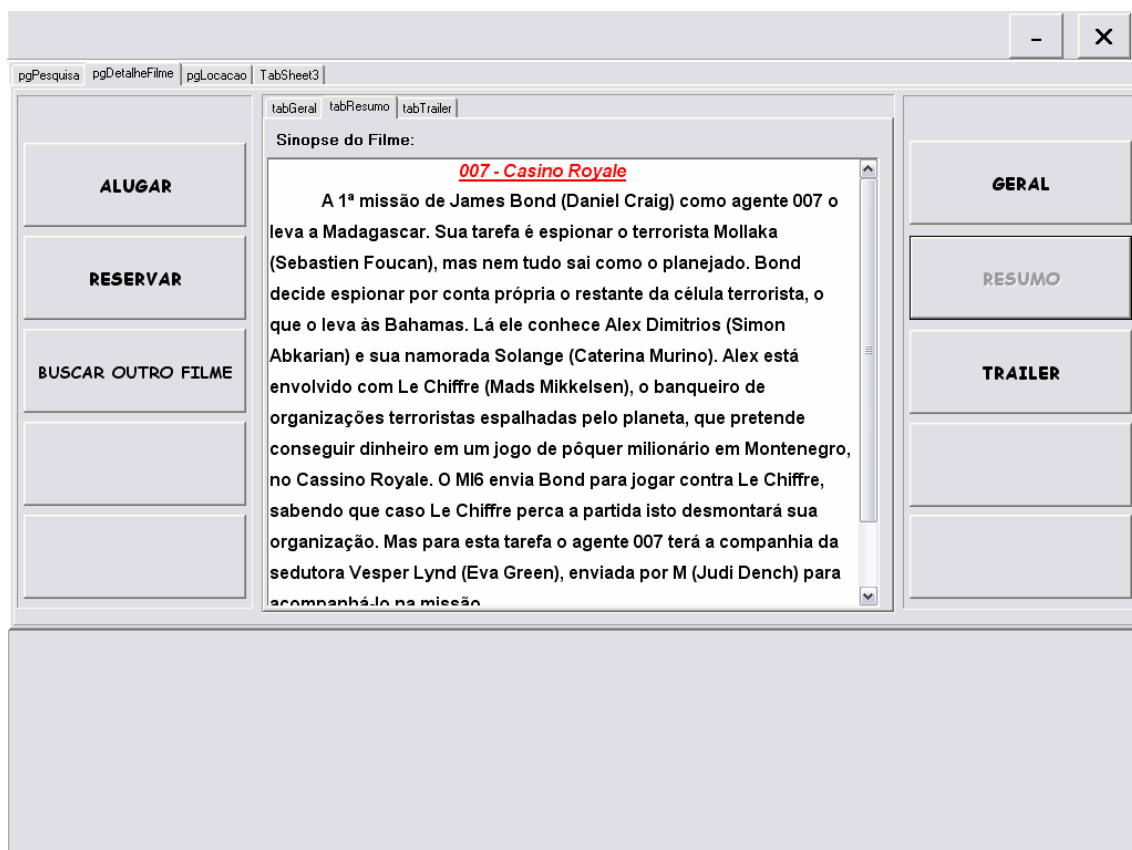


Figura 34 - Exibição da sinopse do filme

Com os trailers o mecanismo de armazenamento é o mesmo que o das sinopses, gravando apenas o caminho do diretório no banco. Para tornar possível que o usuário

assista a vídeos diretamente dentro do software, foi preciso construir um player próprio, utilizando os componentes DSPack, que são o FilterGraph e o VideoWindow, que é uma espécie de tela onde o vídeo é exibido. O Anexo 3 mostra o código-fonte utilizado para construir um player de vídeo, utilizando os componentes DSPack.

O *player* que executa os vídeos foi construído utilizando além dos componentes DSPack, outros componentes nativos do Delphi: um SoundLevel, pra controlar o volume do som; uma TrackBar, para navegar pelo vídeo; uma ToolBar onde foram inseridos os botões de navegação (play, pause, stop, tela cheia) A figura seguinte ilustra o *player* de vídeo construído para rodar os trailers.

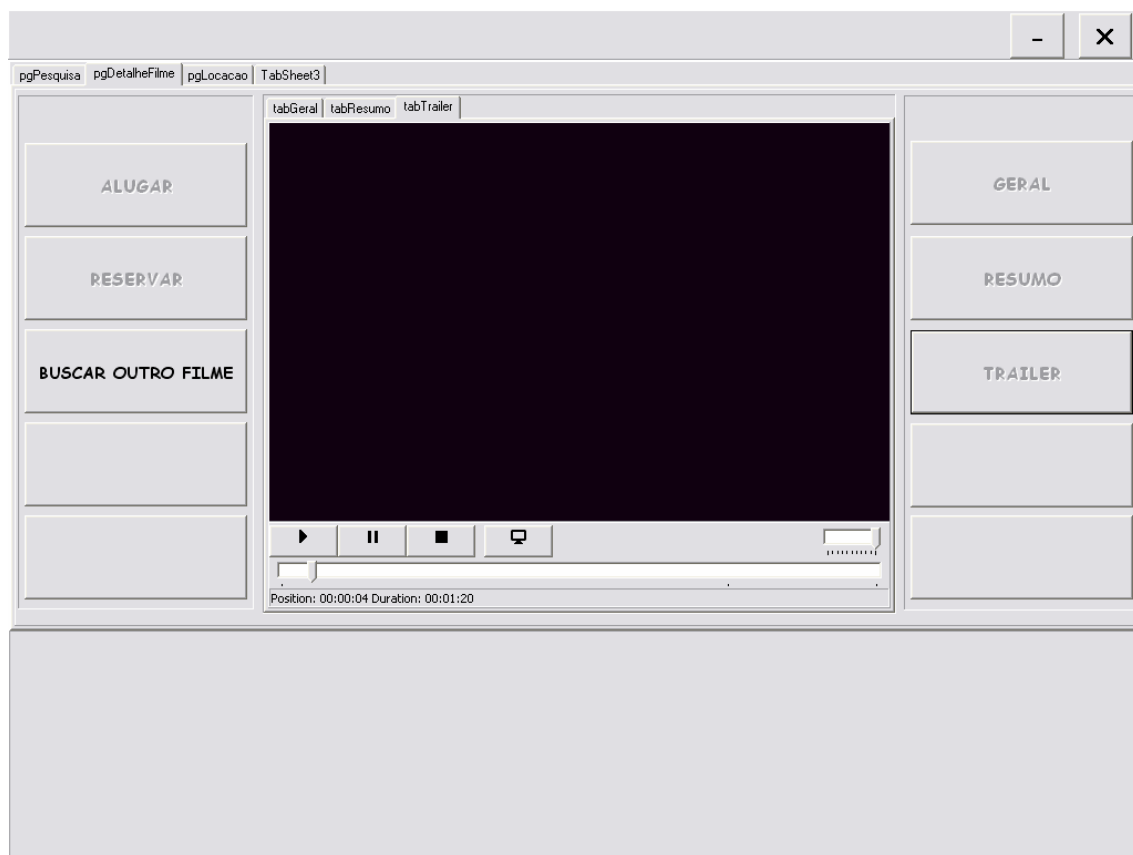


Figura 35 - Player que executa os trailers

Os botões do lado esquerdo são os componentes que efetuam locação e reserva. Quando o usuário seleciona um filme para ver seus detalhes, ele tem a opção de alugá-lo, reservá-lo, ou simplesmente voltar e buscar outro filme.

Caso o filme não esteja disponível no momento, o cliente poderá efetuar uma reserva, válida por um determinado período. O sistema verifica se o filme está disponível pelo campo STATUS da tabela FILMES, sendo que quando estiver gravado o caractere 'D' neste campo é porque o filme está disponível.

Esta reserva poderá conter apenas um filme por vez, sendo que durante a visualização dos detalhes do filme é que se pode reserva-lo, clicando no botão reservar e confirmando numa caixa de diálogo. A figura seguinte mostra a operação de reserva de filmes.

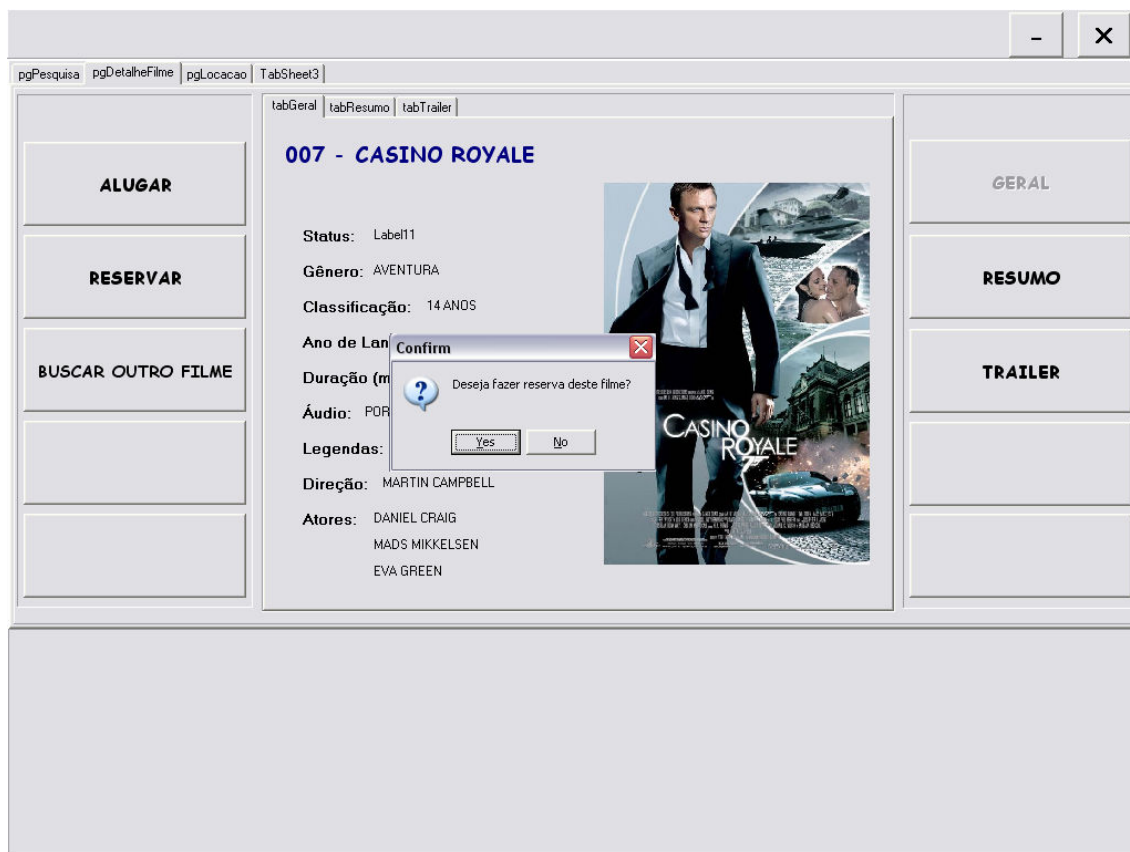


Figura 36 - Reserva do filme

Se a opção for a locação, o filme será adicionado na tabela que indica quais os filmes estão sendo alugados, a tabela `LOCACAO_ITENS`, com isso utilizou-se a relação Mestre / Detalhe, onde a locação é o Mestre e os itens são os detalhes.

Quando o filme é adicionado na locação, o usuário tem a opção de alugar mais algum filme ou concluir a locação. Se a opção for alugar mais filmes, a aplicação irá voltar para a tela de busca onde será escolhido o próximo filme. Caso o cliente deseje concluir a locação, será calculado o total e um recibo será gerado na tela e impresso, contendo o nome do cliente, a data da locação, a data prevista para devolução, o código da locação, que é gerado sequencialmente, os detalhes da locação, isto é, os filmes que foram alugados e o valor a ser pago. Após a emissão deste recibo, o cliente se dirige ao caixa e o entrega ao funcionário da locadora, que irá conferi-lo e entregar os filmes escolhidos ao cliente.

A figura seguinte mostra a operação de locação de filmes, na tela em que é mostrado os detalhes da locação, como a data e os filmes que serão alugados.

The screenshot shows a web application window for movie rental. The window has a title bar with standard minimize, maximize, and close buttons. Below the title bar is a tabbed interface with tabs labeled 'pgPesquisa', 'pgDetalleFilme', 'pgLocacao', and 'TabSheet3'. The 'pgLocacao' tab is active.

On the left side of the 'pgLocacao' tab, there is a vertical sidebar containing four buttons: 'ALUGAR OUTRO FILME', 'EXCLUIR DA LISTA', 'CONCLUIR LOCAÇÃO', and 'CANCELAR LOCAÇÃO'.

The main content area of the 'pgLocacao' tab displays the following information:

- COD LOCAÇÃO:** 10
- DATA DA LOCAÇÃO:** 20/11/2008
- DATA DE DEVOLUCAO:** 22/11/2008
- VALOR TOTAL:**

Below this information is a table with two columns: 'TÍTULO' and 'PREÇO'.

TÍTULO	PREÇO
* 007 - CASINO ROYALE	R\$ 3,50

Figura 37 - Tela de Locação

4 CONCLUSÃO

A área de usabilidade de sistemas ainda é um campo relativamente pouco explorado atualmente, pelo fato de existirem uma grande quantidade de softwares que não se encaixam nos conceitos de usabilidade.

Através do desenvolvimento deste projeto, pode-se afirmar que ele trouxe muitos benefícios, pois mostra como uma interface pode ser construída utilizando os conceitos de usabilidade, sem aumentar o custo da implementação do software, melhorando a interatividade com o usuário e otimizando o atendimento aos clientes, com diminuição de filas.

Permitiu-se também o entendimento da importância da modelagem do sistema utilizando UML, servindo de base para a construção do banco de dados e da aplicação e também de seguir os conceitos de Engenharia de Software, para aumentar a qualidade do software.

Por fim, o desenvolvimento do projeto fez com que adquiríssemos um grande conhecimento sobre as disciplinas estudadas e aprendêssemos a trabalhar em equipe, o que contribuiu para um crescimento profissional e pessoal.

4.1 Dificuldades Encontradas

No decorrer do desenvolvimento de projeto, foram encontradas algumas dificuldades, nas quais podemos citar:

- A dificuldade para encontrar conteúdos relacionados a algumas áreas.
- A busca da melhor interface para satisfazer o usuário, que é um dos principais desafios da área de IHM.

4.2 Trabalhos Futuros

Como perspectiva para trabalhos futuros pode ser citada a idéia de construir uma aplicação completa a partir do protótipo criado, que possa também cuidar de toda a parte administrativa do estabelecimento, tornando esse protótipo um software pronto para ser comercializado. Poderá também ser feito um estudo muito mais aprofundado sobre a interface, melhorando a interação com o usuário.

5 REFERÊNCIAS BIBLIOGRÁFICAS

ANSELMO, Fernando A. F., **Borland Delphi: Desvendando o Caminho das Pedras**. 1995-1997.

AUTOMAÇÃO. Disponível em: <<http://www.automacao comercial.org>> Acesso em 03 Nov. 2008.

CANTU, Carlos Henrique. **Firebird Essencial**. Rio de Janeiro, Editora Ciência Moderna Ltda. 2005.

DCA. Disponível em:
<<http://www.dca.fee.unicamp.br/cursos/EA876/apostila/HTML/node7.html>>. Acesso em 14 nov. 2008.

DEV MEDIA. Disponível em: <<http://www.devmedia.com.br>> Acesso em 16 Nov. 2008.

DSC. Disponível em:
<<http://www.dsc.ufcg.edu.br/~sampaio/cursos/2007.1/Graduacao/SI-II/Uml/diagramas/usecases/usecases.htm>> Acesso em 15 Nov. 2008.

FARRER, Harry. et al. **Algoritmos Estruturados**. Rio de Janeiro. LTC Livros Técnicos e Científicos Editora S.A. 1989.

FOWLER, Martin. **UML Essencial**. Porto Alegre, Bookman, 2005.

IMASTERS. Disponível em:
<http://imasters.uol.com.br/artigo/2169/interbase/tecnologia_dbexpress_e_conexao_ao_firebird/> Acesso em 17 nov. 2008.

KORTH, Henry F.; SILBERSCHATZ, Abraham. **Sistemas de Banco de Dados**, 2ª edição, São Paulo, Makron Books. 1995.

LEITE, Mário; RAHAL Jr., Nelson Abu Sanra. (Agosto de 2002) **Programação Orientada a Objeto: uma abordagem didática.**

Disponível em: <http://www.ccuec.unicamp.br/revista/infotec/artigos/leite_rahall.html>. Acesso em 14 nov. 2008.

LINHA. Disponível em:

<<http://www.linhadecodigo.com.br/ArtigoImpressao.aspx?id=853>>. Acesso em 13 nov. 2008.

MORAES, Paulo Sérgio de. **Lógica de Programação.** Unicamp. 2000. (Apostila).

OLIVEIRA, Wilson José de. **Banco de Dados InterBase com Delphi.** Florianópolis. Visual Books.. 2000.

PONTES. Disponível em: <<http://lf.pontes.sites.uol.com.br/sghd/revisao.pdf>>. Acesso em 22 set. 2008.

PRESSMAN, Roger S.. **Engenharia de software.** 3ª edição. São Paulo. Makron Books, 1995.

RETONDARO. Disponível em:

<<http://www.retondaro.pro.br/luis/ihc/ihcPauloLima1.pdf>> Acesso em 24 set. 2008.

ROCHA, Heloísa V. da; BARANAUSKAS, Maria Cecília C. **Design e Avaliação de Interfaces Humano-Computador.** Campinas. Nield/Unicamp. 2003.

SOUZA, Paulo Roberto Rodrigues de. **Banco de Dados.** 2007. (Apostila).

VOXXEL. Disponível em: <<http://www.voxxel.com.br/pages/images/introu3.gif>> Acesso em: 17 nov. 2008.

WIKIPEDIA. Disponível em: <http://pt.wikipedia.org/wiki/P%C3%A1gina_principal> Acesso em 18 Out. 2008.

6 ANEXOS

6.1 Anexo 1 – Feedback do Cliente

Na fase de implantação, entrega e feedback do Modelo de processo de Prototipagem, ocorreu uma nova entrevista com o cliente, o senhor Leandro, proprietário da Locadora Central, situada em Três Corações, onde ele ressaltou a importância do sistema para seu estabelecimento, onde seus clientes muitas vezes solicitam ajuda para procurar e escolher um filme, pedindo a opinião dos funcionários sobre o filme e até para que ligassem o aparelho de DVD, para mostrar trechos dos filmes que lhes interessavam, tornando o atendimento mais lento, acumulando muitas pessoas na fila de atendimento.

Ao analisar o sistema, ele comentou que a disponibilidade da sinopse e principalmente dos trailers, faria com que o cliente ficasse mais a vontade para fazer a escolha do filme desejado. E a opção do próprio cliente poder efetuar a locação e a reserva do filme, diminuiria consideravelmente o acúmulo de pessoas na fila de atendimento em sua locadora, agilizando e otimizando o atendimento ao público, poupando os seus clientes de qualquer transtorno desnecessário.

Os clientes da vídeo-locadora se identificaram muito com a interface, comentando a sua clareza e simplicidade, que não gera dúvidas sobre a utilização de suas ferramentas e não os induzem a cometer erros, e principalmente a facilidade de aprendizagem, pois, segundo eles, por ser uma interface simples de se utilizar, as suas funcionalidades são memorizadas mais rapidamente fazendo com que eles consigam utilizá-la sem ajuda de um atendente, de forma independente, o que gera mais conforto, satisfação e confiabilidade do usuário para com o sistema.

Porém os clientes opinaram na questão em que poderia ainda aumentar um pouco o tamanho dos botões do player de vídeo que toca o trailer do filme, melhorando a interação no caso de utilização de telas touch-screen.

Foram identificados novos requisitos e novas necessidades, como a implantação de uma parte administrativa para o gerente e demais funcionários da locadora terem um controle das locações e reservas efetuadas através de relatórios e também fazerem cadastros dos filmes, clientes e outras informações necessárias. Também foi identificada a necessidade de melhorar a segurança do software, pedindo a confirmação do login do

usuário sempre que ele for fazer mais de uma operação, evitando que os usuários façam locações e reservas no nome de outros usuários.

O software pode ser modificado de acordo com as informações coletadas, entrando novamente no ciclo do paradigma da prototipagem, visando a evolução do protótipo até que se torne inteiramente pronto para ser comercializado.

6.2 Anexo 2 – Implementação do Teclado Virtual

O código-fonte a seguir mostra a rotina para a implementação do teclado virtual utilizado no protótipo. Os métodos de inserção de caracteres são relacionados ao evento onClick de cada tecla e a exclusão inserção de um espaço entre os caracteres é relacionado ao evento onClick dos botões btnBackspace e btnEspaco, respectivamente. Em todos os métodos verifica-se, primeiramente, qual página é o índice do componente PageControl. Será exibido apenas o método para inserção na página de índice 0, pois nas demais páginas a lógica é praticamente a mesma.

O método a seguir é para inserção e caracteres:

```
//-----TECLADO VIRTUAL-----  
//evento para inserir um caracter no edit  
procedure TfrmLocacao.btnQClick(Sender: TObject);  
begin  
  case pageBusca.TabIndex of  
    0:begin  
      if edtTituloFilme.Focused = true then  
        edtTituloFilme.Text := edtTituloFilme.Text + tbutton(sender).caption  
      else  
        begin  
          edtTituloFilme.SetFocus;  
          edtTituloFilme.Text := edtTituloFilme.Text + TButton(Sender).Caption;  
        end;  
      edtTituloFilme.SelStart := length(edtTituloFilme.Text);  
    end;  
  end;  
end;
```

O método seguinte implementa a inserção de um espaço entre os caracteres, que na verdade é uma string vazia.

```
// tecla espaco  
procedure TfrmLocacao.btnEspacoClick(Sender: TObject);  
begin
```

```

case pageBusca.TabIndex of //verifica em qual tabsheet se encontra
0:begin
    if edtTituloFilme.Focused = true then //verifica se o foco esta no edit
        edtTituloFilme.Text := edtTituloFilme.Text + ' ' //insere um espaco vaziao na frente
                                //do conteudo do edit
    else
        begin
            edtTituloFilme.SetFocus; //senao coloca o foco do cursor no edit
            edtTituloFilme.Text := edtTituloFilme.Text + ' '; //insere o espaco
        end;
        edtTituloFilme.SelStart := length(edtTituloFilme.Text); //manda o cursor pra frente
                                //do último caractere
    end;
end;

```

O método a seguir é relacionado ao evento que apaga os caracteres do Edit:

```

// pressionar a tecla APAGAR (backspace)
procedure TfrmLocacao.btnBackspaceClick(Sender: TObject);
begin
    case pageBusca.TabIndex of
    0:begin
        if edtTituloFilme.Focused = true then
            edtTituloFilme.Text := Copy (edtTituloFilme.Text,1,length (edtTituloFilme.Text)
-1)
        else
            begin
                edtTituloFilme.SetFocus;
                edtTituloFilme.Text := Copy(edtTituloFilme.Text,1,length(edtTituloFilme.Text)
-1);
            end;
            edtTituloFilme.SelStart := length(edtTituloFilme.Text);
        end;
    end;
end;

```

6.3 Anexo 3 – Construção de um Player de Vídeo

```
//-----  
// duplo clique na tela de vídeo, tela cheia  
procedure TfrmLocacao.VideoWindowDbClick(Sender: TObject);  
begin  
    VideoWindow.FullScreen := not VideoWindow.FullScreen;  
    btFullScreen.Down := VideoWindow.FullScreen;  
end;  
  
// clique no botão play, ativa a propriedade play do filterGraph  
procedure TfrmLocacao.btPlayClick(Sender: TObject);  
begin  
    FilterGraph.play;  
    TravaBotoesLocacao;  
end;  
  
//clique no botão pause, ativa a propriedade pause do filterGraph  
procedure TfrmLocacao.btPauseClick(Sender: TObject);  
begin  
    FilterGraph.Pause;  
end;  
  
// clique botão stop, para o trailer e volta na tela geral de descrição do filme  
// habilitando novamente os botões de navegação  
procedure TfrmLocacao.btStopClick(Sender: TObject);  
begin  
    FilterGraph.Stop;  
    DestravaBotoesLocacao;  
    pageFilme.TabIndex := 0;  
    btnGeral.Enabled:= False;  
    if (dmLocacao.cdsBuscaFilme.fieldByName('LOCAL_VIDEO').AsString) <> " then  
        btnTrailer.Enabled:=True  
    else btnTrailer.Enabled:=False;
```

```

if (dmLocacao.cdsBuscaFilme.fieldByName('LOCAL_DESCR').AsString) <> " then
  btnResumo.Enabled:=True
else btnResumo.Enabled:=False;
end;

// botão tela cheia, faz a mesma coisa que duplo clique na tela de video
procedure TfrmLocacao.btFullScreenClick(Sender: TObject);
begin
  VideoWindow.FullScreen := not VideoWindow.FullScreen;
  btFullScreen.Down := VideoWindow.FullScreen;
end;

//quando muda a posição do soundLevel, altera também a prop. Volume do FilterGraph
procedure TfrmLocacao.SoundLevelChange(Sender: TObject);
begin
  FilterGraph.Volume := SoundLevel.Position;
end;

//mostra o tempo do vídeo na barra de status, conforme posição do TrackBar
procedure TfrmLocacao.TrackBarTimer(sender: TObject; CurrentPos,
  StopPos: Cardinal);
begin
  StatusBar.SimpleText := format('Position: %s Duration: %s',
    [TimeToStr(CurrentPos / MiliSecPerDay), TimeToStr(StopPos / MiliSecPerDay)])
end;

//-----

```