

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



***Framework* para interface e gerenciamento de bancos de dados**

LIDIANA MENDES DA SILVA

Dezembro

2009

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

***Framework* para interface e gerenciamento de bancos de dados**

Lidiana Mendes da Silva

Dissertação apresentada ao Programa de Pós-graduação em Engenharia Elétrica da Universidade Federal de Uberlândia, como parte do cumprimento dos requisitos necessários à obtenção do grau de Mestre em Ciências.

Área de Concentração: Engenharia Biomédica

Prof. Dr. Alcimar Barbosa Soares
Orientador (UFU)

Prof. Dr. Alexandre Cardoso
Coordenador do Curso de Pós-Graduação

Dezembro

2009

Dados Internacionais de Catalogação na Publicação (CIP)

S586f Silva, Lidiania Mendes da, 1976-
Framework para interface e gerenciamento de bancos de dados /
Lidiania Mendes da Silva. - 2010.
118 f. : il.

Orientador: Alcimar Barbosa Soares.
Dissertação (mestrado) – Universidade Federal de Uberlândia,
Programa de Pós-Graduação em Engenharia Elétrica.
Inclui bibliografia.

1. Banco de dados - Teses. I. Engenharia biomédica - Banco
de dados - Teses. I. Soares, Alcimar Barbosa, 1965- II.
Universidade Federal de Uberlândia. Programa de Pós-Graduação
em Engenharia Elétrica. III. Título.

CDU: 681.3.07

**FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA**

LIDIANA MENDES DA SILVA

Dissertação apresentada ao PPGEEL em Engenharia Elétrica
da Universidade Federal de Uberlândia para a obtenção do
grau de Mestre em Ciências.

Banca examinadora:

Dr. Alcimar Barbosa Soares (UFU) - Orientador

Dr. Alexandre Cardoso (FEELT/UFU)

Dra. Selma Terezinha Milagre (UFG)

UBERLÂNDIA, MG

2009

AGRADECIMENTOS

A **DEUS** por iluminar meu caminho, pelas pessoas que colocaram à minha volta, e pelas oportunidades que tive em minha vida.

Aos meus pais, **Dirceu** e **Nemer**, pelas angústias e preocupações que passavam por minha causa, por terem dedicado suas vidas a mim, pelo amor, carinho e estímulo que me ofereceram.

As minhas irmãs **Luciana** e **Fabiana**, por estarem sempre ao meu lado prontas para ajudar.

Ao professor **Alcimar**, orientador deste trabalho, meu reconhecimento, respeito e agradecimento pelos ensinamentos recebidos durante todo o tempo da realização do mestrado.

Aos meus queridos amigos do BioLab (Laboratório de Engenharia Biomédica e Automática), que de uma forma ou de outra contribuíram para o meu crescimento.

À todos **professores e funcionários** da Universidade Federal de Uberlândia sempre dispostos a ajudar.

A **Marly**, por me ajudar em tudo que foi necessário durante o curso de mestrado.

Enfim, à **todos** aqueles que de alguma forma contribuíram para este trabalho.

Não conheço ninguém que conseguiu realizar seu sonho, sem sacrificar feriados e domingos pelo menos uma centena de vezes. O sucesso é construído à noite! Durante o dia você faz o que todos fazem. Mas, para obter um resultado diferente da maioria, você tem que ser especial. Se fizer igual a todo mundo, obterá os mesmos resultados. Não se compare à maioria, pois infelizmente ela não é modelo de sucesso. Se você quiser atingir uma meta especial, terá que estudar no horário em que os outros estão tomando chope com batatas fritas. Terá de planejar, enquanto os outros permanecem à frente da televisão. Terá de trabalhar enquanto os outros tomam sol à beira da piscina. A realização de um sonho depende de dedicação. Há muita gente que espera que o sonho se realize por magia. Mas toda magia é ilusão. A ilusão não tira ninguém de onde está. Ilusão é combustível de perdedores.

“Quem quer fazer alguma coisa, encontra um meio. Quem não quer fazer nada, encontra uma desculpa”.

Roberto Shinyashiki

LISTA DE ACRÔNIMOS

AV - Ambiente Virtual

BD - Banco de Dados

BLL - Business Logic Layer (Camada da Lógica de Negócios)

CC - Crânio-caudal

DAL - Data Access Layer (Camada de Acesso a Dados)

DLL - Dinamic Link Library (Biblioteca de Ligação Dinâmica)

ECG - Eletrocardiografia

EEG - Eletroencefalografia

EMG – Eletromiografia

MLO - Médio-lateral oblíqua

PI - Processamento de Imagens

RV - Realidade Virtual

SGBD - Sistema de Gerenciamento de Banco de Dados

SQL - Structured Query Language (Linguagem de Consulta Estruturada)

RESUMO

SILVA, L. M. ***Framework para interface e gerenciamento de bancos de dados.***

Dissertação de Mestrado. Faculdade de Engenharia Elétrica da Universidade Federal de Uberlândia. Uberlândia, 2009.

A utilização de bancos de dados específicos para determinado aplicativo não permite ao usuário a substituição do banco de dados ou o compartilhamento com outras bases de informações sem a reconstrução de todo o aplicativo, tornando não trivial a aplicação das soluções propostas na literatura. Isso reduz a interoperabilidade entre os aplicativos de *software* de diferentes fornecedores de equipamentos biomédicos existentes. Este trabalho descreve o projeto e o desenvolvimento de um *framework* para interface e gerenciamento de bancos de dados utilizados em aplicações biomédicas. O sistema se caracteriza por permitir que diferentes bancos de dados possam ser utilizados para diferentes aplicativos a fim de armazenar os dados para posteriores análises. O sistema foi projetado utilizando a técnica de orientação a objetos, *plug-ins* e reflexão, permitindo criar aplicativos capazes de conectarem-se com os diferentes bancos de dados. Os resultados demonstram que o *framework* permite a coleta e o armazenamento de informações biomédicas por meio de sistemas adaptativos e de *plug-ins*, minimizando os problemas de falta de compatibilidade entre bancos, as dificuldades de manutenção e melhorando a integração dos mesmos.

Palavras-chaves: Banco de dados; Software adaptativo; Reflexão; *Plug-ins*

ABSTRACT

SILVA, L. M. **Interface and management framework for systems data bank.** Master Dissertation. Faculty of electrical Engineering. University of Uberlândia. Uberlândia, 2009.

The usage of specific databases for certain application does not allow the replacement of database or sharing information with other databases by user, without rebuilding the entire application, turning non-trivial the application of the solutions proposed in the literature. This reduces the interoperability among different suppliers of present biomedical data. This paper describes the design and development of a system for interface and management of biomedical signals databases. The system's features includes the use of different databases and the storage of the data for further analysis and was designed using the object-oriented technique, plug-ins and reflection technique, leading to create an application capable of connecting with the different databases. The developed system has capability of storage biomedical information, through the use of adaptive systems and plug-ins, reducing the problems of lack of compatibility between data banks, difficulties in maintenance and integration among them. The experiment results showed that the framework is able to collect patients biomedical data, that may be registered with their clinical features, and interact with differentdatabases.

Keywords: Data base; Adaptative software; Reflection; *Plug-ins*

SUMÁRIO

INTRODUÇÃO	1
1.1 APLICAÇÕES DE SISTEMAS DE INFORMAÇÃO NA ÁREA BIOMÉDICA	3
1.2 SISTEMAS DE INFORMAÇÃO EM PESQUISA CIENTÍFICA	4
1.3 MOTIVAÇÃO	4
1.4 OBJETIVOS DO TRABALHO	5
1.5 ESTRUTURA DA DISSERTAÇÃO	5
BANCOS DE DADOS	7
2.1 VANTAGENS E DIFICULDADES	10
2.2 ESTRUTURAS DE DADOS	12
2.3 MODELAGEM	18
2.4 FERRAMENTAS DE MODELAGEM	18
2.5 SERVIDORES DE BANCOS DE DADOS	22
APLICAÇÕES DE BANCOS DE DADOS NA ÁREA BIOMÉDICA	26
3.1 APLICAÇÕES DE BANCOS DE DADOS PARA IMAGENS MÉDICAS	26
3.2 APLICAÇÕES DE BANCO DE DADOS PARA SINAIS BIOMÉDICOS	40
TÉCNICAS E FERRAMENTAS PARA DESENVOLVIMENTO DE APLICATIVOS PARA OPERAÇÃO E MANUSEIO DE BANCOS DE DADOS	51
4.1 FRAMEWORK	51
4.1.1 Framework de componentes	52
4.2 MODELO MULTICAMADAS	54
4.2.1 Camada de Apresentação	56
4.2.2 Camada da Lógica de Negócios	56
4.2.3 Camada de Acesso a Dados	57
4.2.4 Vantagens e desvantagens da utilização do Modelo Multicamadas	59
4.3 DB PROVIDER FACTORY	60
4.4 REFLEXÃO	64
4.5 PLUG-IN	69
PROPOSTA DE UM FRAMEWORK PARA INTERFACE E GERENCIAMENTO DE BANCOS DE DADOS	72
5.1 REQUISITOS	72
5.2 DESENVOLVIMENTO DO FRAMEWORK	75
5.3 O GERENCIADOR DO PLUG-IN	82
5.4 A IMPLEMENTAÇÃO DO FRAMEWORK	83
5.5 DESENVOLVENDO APLICATIVOS COM O FRAMEWORK	86
5.6 CRIANDO OS PLUG-INS	88
5.7 CONCLUSÃO	89
PROVA DE CONCEITO	90
6.1 ESTRUTURA DO FRAMEWORK	90
6.2 MODELAGEM DE DADOS	91
6.3 CRIAÇÃO DOS REGISTROS NOS BANCOS DE DADOS	96
6.4 ACESSO AO PLUG-IN	96
6.5 APLICATIVO	97

6.5.1 Interface Principal	97
6.5.2 Menu do programa	98
6.5.2.1 Menu Auxiliar	98
6.5.2.2 Menu Cadastros	99
6.6 EXEMPLOS DE UTILIZAÇÃO	102
6.7 ATUALIZAÇÕES NO SISTEMA PARA OUTROS BANCOS DE DADOS.....	106
6.8 CONCLUSÃO	107
CONCLUSÕES E TRABALHOS FUTUROS	108
7.1 TRABALHOS FUTUROS	111
REFERÊNCIAS.....	112

LISTA DE FIGURAS

Figura 3.1: Interface Inicial do aplicativo sistema de controle de pacientes para cadastro de dados e imagens. Fonte: Geraldeli et al., 2002, p. 28.	31
Figura 3.2: Cadastro das principais informações de pacientes do aplicativo sistema de controle de pacientes. Fonte: Geraldeli et al., 2002, p. 28.	31
Figura 3.3: Cadastro de consulta de pacientes do aplicativo sistema de controle de pacientes. Fonte: Geraldeli et al., 2002, p. 29.	32
Figura 3.4: Tela que mostra as quatro imagens mamográficas de um exame em diversas posições do software para treinamento na interpretação de imagens mamográficas em sistemas digitais. Fonte: Pires, Medeiros; Schiabel, 2004, p. 244.	33
Figura 3.5: Tela que mostra o grupo crânio caudal, no mesmo tamanho físico, com o objetivo de avaliar de forma mais eficiente as estruturas internas, desenvolvido pelo software para treinamento na interpretação de imagens mamográficas em sistemas digitais. Fonte: Pires, Medeiros; Schiabel, 2004, p. 246.	34
Figura 3.6: Interface para treinamento, na qual são exibidos os dados do paciente e algumas opções de configuração na parte superior. Fonte: DELFINO, S. R e NUNES, F. L. S, 2009, p. 6.	37
Figura 3.7: Interface para treinamento, na qual mostra a colisão detectada quando o usuário encosta a agulha no nódulo. Fonte: DELFINO, S. R e NUNES, F. L. S, 2009, p. 6.	38
Figura 3.8: Interface utilizada para exemplificação de estudo de caso. Fonte: DELFINO, S. R e NUNES, F. L. S, 2009, p. 7.	38
Figura 3.9: Interface Inicial do sistema de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 3.	42
Figura 3.10: Resultado do sinal ECG de um paciente armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 3.	43
Figura 3.11: Resultado do EMG de um paciente armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 3.	43
Figura 3.12: Resultado das últimas 10 amostras da Pressão Arterial de um paciente armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.	44
Figura 3.13: Resultado da Frequência Cardíaca de um paciente, armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.	44

Figura 3.14: Resultado da Oximetria de pulso de um paciente, armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.	45
Figura 3.15: Resultado da Temperatura de um paciente, armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.	45
Figura 3.16: Interface para monitorização remota dos sinais do aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 5.....	46
Figura 3.17: Interface para relaxamento através da técnica de neurofeedback usando o ambiente circo. Fonte: SÁ, A. A. R, 2004, p. 3.	48
Figura 3.18: Interface para relaxamento através da técnica de neurofeedback usando o ambiente neblina. Fonte: SÁ, A. A. R, 2004, p. 3.	49
Figura 3.19: Interface para relaxamento através da técnica de neurofeedback usando o ambiente quebra-cabeça. Fonte: SÁ, A. A. R, 2004, p. 3.....	49
Figura 4.1: Apresentação do modelo multicamadas. Fonte: CARVALHO, R. M, 2008.	55
Figura 4.2: Herança da classe <i>DbProviderFactory</i> . Fonte: AECE, I, 2005.	62
Figura 4.3: Esquema simplificado de classes para acesso a dados no ADO.NET 2.0. Fonte: CARVALHO, R. M, 2008.	63
Figura 4.4: Pacotes da plataforma .net utilizados na reflexão com as classes disponíveis nesse pacote.	67
Figura 4.5: Arquitetura da plataforma <i>Eclipse</i>	71
Figura 5.1: Modelo de desenvolvimento tradicional na maioria dos aplicativos.	73
Figura 5.2 (a): Variante da arquitetura proposta, baseada em um aplicativo, um <i>plug-in</i> e diversos bancos de dados.	74
Figura 5.2 (b): Outra variante da arquitetura proposta, baseada em diversos aplicativos, <i>plug-in</i> e diversos bancos de dados.	74
Figura 5.3: Modelo multicamadas da arquitetura proposta.	76
Figura 5.4: Diagrama de blocos da arquitetura proposta.	77
Figura 5.5: Diagrama da classe <i>DbProviderFactory</i> com os diferentes bancos de dados.	80
Figura 5.6: Diagrama de classes para habilitar e desabilitar os diversos bancos de dados.	82
Figura 6.1: Estrutura do <i>framework</i>	91
Figura 6.2: Visão geral da modelagem de dados do aplicativo exemplo.....	95
Figura 6.3: Classe <i>plug-in</i> do <i>framework</i>	97
Figura 6.4: Interface principal do <i>framework</i> para interface e gerenciamento de bancos de dados.	98
Figura 6.5: Interface para escolha do banco de dados a ser utilizado do <i>framework</i> para interface e gerenciamento de bancos de dados.	99
Figura 6.6: Interface de cadastramento do sinal coletado, imagem e informações relativas aos pacientes do <i>framework</i> para interface e gerenciamento de bancos de dados.	100
Figura 6.7: Interface fornecida pelo sistema ao usuário para edição de dados.	101

LISTA DE TABELAS

Tabela 2.1: Registro de departamento exemplificando a chave primária.	16
Tabela 2.2: Registro de funcionários exemplificando a chave primária e a chave estrangeira em relação à Tabela departamentos.	17
Tabela 2.3: Registro de funcionários exemplificando chave alternativa.	17
Tabela 2.4: Comparativo das características de ferramentas de modelagem.	21
Tabela 2.5: Comparativo das características dos Servidores de Bancos de Dados.	24

CAPÍTULO 1

INTRODUÇÃO

O volume de informações disponíveis nos dias atuais aumenta, consideravelmente, a cada momento. A gestão desse volume de dados torna-se, por consequência, um desafio cada vez maior. Para suprir tais limitações, frequentemente, lança-se mão de recursos externos como agendas, listas de tarefas, receitas, calculadoras, computadores ou outros. Esses recursos são utilizados não só no cotidiano pessoal, mas, principalmente, em empresas ou organizações que manipulam elevados volumes de informações, necessitando manter o conhecimento sobre diversos aspectos de suas atividades (WATSON, 2004).

A qualidade do desempenho de diversas atividades depende do acesso e manuseio eficiente de informações. A utilização de registros convencionais (fichas, livros, pastas etc) ocupa muito tempo dos profissionais, gerando, ainda, desperdício de espaço e tempo, além da possibilidade de deterioração dos registros, dificuldade de controle e de organização dos dados, entre outras (SABBATINI e ORTIZ, 1994).

Por outro lado, as organizações que trabalham com grande número de dados podem obter benefícios importantes com a informatização, dentre os quais a organização sistemática e melhor legibilidade dos registros, a maior rapidez de acesso aos dados, a agilidade e racionalização da emissão de documentos e relatórios, a otimização de tempo e espaço, a

diminuição de erros e desperdícios, e a melhoria da imagem do serviço profissional (SABBATINI e ORTIZ, 1994).

Com os grandes avanços tecnológicos, a partir da década de 40, praticamente todas as áreas do conhecimento permitiram importantes saltos técnico-científicos. Em especial, podemos destacar os avanços observados na área de saúde, que têm permitido a realização de inúmeros exames e diagnósticos, de forma não invasiva e extremamente precisa, baseados, por exemplo, em análises de registros de sinais biológicos variados (SABBATINI, 1995). O resultado é um aumento muito grande na quantidade de pacientes atendidos, na quantidade de exames e análises clínicas realizadas, num menor espaço de tempo. Por conseguinte, o volume de dados gerados por esses sistemas tem crescido enormemente (MACHADO, 2004).

Os dados são organizados obedecendo a um conjunto de codificações e regras pré-estabelecidas (tipos de dados, ordem interna etc). Por exemplo, num prontuário informatizado, podem ser incluídas informações como nome, idade, sexo, documentação, data de nascimento, profissão, anamnese, exames realizados (eletroencefalograma - EEC, eletrocardiograma - ECG, eletromiograma - EMG etc), diagnósticos, parâmetros de normalidade, observações, tratamentos indicados e solicitação de outros exames (ILHA, 1993 e MACHADO, 2004).

O gerenciamento inadequado desse amplo e diversificado conjunto de informações pode resultar em dados não padronizados, duplicidades, inconsistências, falta de interoperabilidade, além de perda de informações úteis. Para solucionar tais problemas, os pesquisadores da área de informática desenvolveram técnicas especiais para armazenamento, manipulação e tratamento de dados, em unidades especiais, conhecidas como “bancos de dados”. Tais bancos se caracterizam por uma organização sistemática das informações, dentro de rigorosos padrões, de forma a garantir a segurança e a confiabilidade dos dados.

Entretanto, a criação de um banco de dados deve envolver um planejamento cuidadoso e o conhecimento de todos os fatores envolvidos no processo.

1.1 Aplicações de Sistemas de Informação na área Biomédica

Os primeiros sistemas hospitalares computadorizados foram desenvolvidos nos grandes hospitais-escola da Europa e Estados Unidos, na década de 60. Naquela época, apenas as grandes instituições podiam arcar com os altos custos envolvidos na aquisição dos equipamentos necessários para a implantação de sistemas de gerenciamento de dados hospitalares. Os aplicativos de *software* necessários eram desenvolvidos apenas por fabricantes de hardware e outras empresas que possuíam o conhecimento do funcionamento dos elementos de *hardware* (JOHANSTON, 1993).

Na década de 90, o prontuário médico manuscrito ainda era a principal fonte primária de dados sobre a saúde de pacientes. Vale ressaltar que os sistemas de informação desenvolvidos para a área Biomédica exigem a aquisição de equipamentos compatíveis com o volume de dados cadastrados, uma vez que são em grande número as informações geradas nos aspectos relacionados à saúde (JOHANSTON, 1993).

No Brasil, já pode ser observada a utilização de bancos de dados em inúmeras instituições de saúde (exames e tratamentos), em clínicas particulares de diversas especialidades e em pesquisas. Dentre os objetivos que se espera alcançar com o desenvolvimento dessa tecnologia, está a possibilidade de aperfeiçoar a conexão e comunicabilidade entre os diversos setores médicos e biomédicos como: eletroencefalografia, eletrocardiografia, radiologia, ultrassonografia, diagnósticos laboratoriais etc (ILHA, 1993).

1.2 Sistemas de Informação em pesquisa científica

Os Sistemas de Informação têm sido utilizados com sucesso, em diversas áreas de pesquisa, facilitando a organização, padronização, disponibilização e comparação entre dados científicos. Esse tipo de aplicação proporciona maior agilidade e diminui os erros de análise entre as informações obtidas nas pesquisas, possibilitando, ainda, o armazenamento e acesso a números expressivos de publicações científicas produzidas no país e no exterior. Ao utilizar os sistemas de registros clínicos computadorizados, podem ser observadas melhorias na qualidade dos serviços, favorecendo usuários do sistema (médicos, empresários e demais colaboradores da instituição) e pacientes (ILHA, 1994).

1.3 Motivação

As informações, de uma forma geral, necessitam de aplicativos eficientes para sua manipulação. Desse modo, a utilização de bancos de dados vem aumentando e se difundindo, a cada dia, nas diversas áreas, possibilitando a integração de diferentes bases de dados para se obter informações mais completas. Contudo, a conexão entre os diferentes softwares e bancos de dados tem representado grandes desafios, uma vez que os bancos de dados são elaborados com finalidades, tecnologias e características específicas, em que nem sempre são compatíveis entre si, ou interoperáveis. Tal estratégia de integração apresenta diversas dificuldades, como, por exemplo, se o usuário necessitar mudar de banco de dados, porque ultrapassou o tamanho máximo do banco de dados atual ou por reduções de custos, terá de refazer grande parte da aplicação por acessar todos os seus recursos diretamente. Assim, uma necessidade que se impõe é a construção de soluções para conectar aplicações diversas com bancos de dados diversos.

1.4 Objetivos do trabalho

Este trabalho tem como objetivo construir um *framework* para interfaceamento de bancos de dados, contendo informações sobre dados biomédicos, visando diminuir o impacto de alterações do aplicativo base e permitir a conexão de bancos de dados diferentes a um mesmo software, de maneira simples e transparente.

O *framework* será implementado na forma de uma biblioteca de software.

Para desenvolver essa arquitetura, as seguintes metas foram propostas:

- Selecionar plataformas necessárias de bancos de dados existentes e suas aplicações na área biomédica.
- Desenvolver um *framework* para armazenamento de dados.
- Propor uma estrutura para os elementos da arquitetura proposta.
- Desenvolver aplicativo para validação da arquitetura proposta.

1.5 Estrutura da dissertação

Este trabalho está organizado da seguinte maneira:

Capítulo 1 – Aborda a contextualização do assunto, apresentação da motivação, objetivos e estrutura da dissertação.

Capítulo 2 – Exibe conceitos de bancos de dados com suas vantagens e dificuldades, ferramentas de modelagem, tabelas comparativas e ferramentas de modelagem e servidores de bancos de dados.

Capítulo 3 – Apresenta um levantamento bibliográfico de aplicações de bancos de dados nas plataformas *Microsoft Access*, *Oracle* e *Microsoft Sql Server* na área biomédica.

Capítulo 4 – Mostra os conceitos e características da arquitetura em camadas, bem como as vantagens e desvantagens de sua utilização, além de apresentar os componentes para construção da arquitetura proposta.

Capítulo 5 – Apresenta a proposta de um *framework* para interface e gerenciamento de bancos de dados, para aplicações biomédicas, com suas etapas de desenvolvimento e a documentação necessária para a utilização do mesmo.

Capítulo 6 – Evidencia a avaliação do *framework*, com a construção de um aplicativo, em que o *framework* é utilizado para armazenar dados biomédicos, conectando-se com bancos de dados diferentes.

Capítulo 7 – Exibe as discussões e conclusões obtidas a partir da análise dos resultados e sugere trabalhos futuros que poderão ser realizados para estender as contribuições fornecidas por esta pesquisa.

CAPÍTULO 2

BANCOS DE DADOS

Um Banco de Dados – (BD) pode ser considerado como uma coleção de dados inter-relacionados e estruturados, projetado para suprir as necessidades de um grupo específico de aplicações e usuários. Um BD organiza e estrutura as informações de modo a facilitar consultas, atualizações e deleções de dados; um dos seus objetivos é isolar os usuários dos detalhes mais internos dos bancos de dados, provendo independência dos dados às aplicações (estrutura física de armazenamento e a estratégia de acesso). Geralmente os BD's são utilizados para definir dados digitais, ou em outro formato, conter informações necessárias ao usuário (KORTH; SILBERCHATZ e SUDARSHAN, 1999).

As informações, de forma geral, são elementos essenciais para empresas, governos, pesquisadores e diferentes segmentos sociais que utilizam variados dados para assegurar a eficácia do próprio funcionamento. Essas informações digitais organizadas facilitam o acesso aos registros, gerando relatórios e estatísticas diversas. Nesse sentido, os bancos de dados apresentam coleções ordenadas, coerentes, com significados, possuem propósito específico e representam aspectos do mundo real (MACHADO, 2004).

Os registros que fazem parte de um banco de dados consistente devem ser bem organizados e conferidos nas instituições produtoras, requerendo um minucioso trabalho de análise, tratamento e organização das informações, devendo ser dotados de toda a documentação necessária ao manuseio (FERREIRA, 2003).

Para o manejo adequado de um grande volume de informações, é necessária sua organização em um sistema computadorizado, em banco de dados (WATSON, 2004).

À medida que ocorreu o aumento de uso de computadores e de números de usuários das áreas comerciais e particulares, os sistemas de banco de dados foram sendo aperfeiçoados, levando em consideração as necessidades dos clientes e as falhas detectadas, expandindo seu uso e capacidade de armazenamento, devido à infinidade de aplicações possíveis (listas de e-mails, produtos, serviços, vídeos etc) (SANCHES, 2005).

Essa ampla aplicabilidade, facilitada pelos avanços tecnológicos e desenvolvimento de sistemas, possibilita variadas utilizações (GERALDELI et al., 2002 e WATSON, 2004).

O desenvolvimento tecnológico atual tem acompanhado a demanda dos segmentos da sociedade (políticos, econômicos, científicos etc), por meio de ferramentas que possibilitem produção, armazenamento, acesso e transmissão de informações com rapidez, flexibilidade, eficácia, baixo custo e segurança. (FERREIRA, 2003 e WATSON, 2004).

Dessa forma, as instituições governamentais e as empresas perceberam a necessidade de investimentos em capacitação de pessoal, *software*, *hardware* e métodos específicos para a organização e acesso aos dados gerados pelas atividades desenvolvidas nas diversas áreas de conhecimento e atividades humanas (FERREIRA, 2003 e WATSON, 2004).

Entretanto, segundo Watson (2004), para a devida organização e acesso àqueles dados, é importante que os dados armazenados possuam as seguintes características:

- compartilhamento: prontamente acessíveis a mais de uma pessoa ao mesmo tempo;
- disponibilidade: facilmente disponíveis para um tomador de decisão;
- segurança: protegidos contra destruição e uso não-autorizado;
- precisão: confiáveis, armazenados de maneira precisa;
- oportunidade: correntes e atuais;
- pertinência: apropriados para a decisão.

Ainda, segundo Watson (2004), a forma de gerenciamento de dados é uma tarefa que enfrenta inúmeros desafios para seu completo domínio; dentre estes, podem-se destacar:

- falta de controle de dados: são estes controlados de maneira precária;
- interface pouco amigável: dificuldade para acessar os dados;
- atrasos: freqüentes atrasos durante a solicitação dos dados;
- deficiência na reprodução da realidade: sistemas gerenciadores de dados não refletem a complexidade do mundo real;
- falta de integração entre os dados: estão estes dispersos em diferentes sistemas.

Em ambientes organizacionais, o acesso aos dados exige rapidez e eficiência, além de uma série de outros requisitos necessários à qualidade de tomada de decisões, exigindo a manutenção de arquivos seguros (WATSON, 2004).

Seja qual for a área de atuação de uma organização, a grande quantidade de informações acumuladas em seus bancos de dados informatizados pode esconder conhecimentos valiosos e úteis para a tomada de decisões. O aumento acentuado no volume dos dados, associado à crescente demanda por conhecimento novo para decisões estratégicas,

tem provocado o interesse crescente em descobrir conhecimento em banco de dados (WATSON, 2004).

2.1 Vantagens e Dificuldades

Um sistema de bancos de dados, elaborado de acordo com as necessidades do cliente e operado de maneira adequada, pode proporcionar inúmeras vantagens ao usuário; como, por exemplo:

- elimina ou reduz as redundâncias: como o cadastro de dados é organizado em arquivos, evita-se a duplicidade de cadastros e a inconsistência de dados, provocadas pelos valores diferentes registrados para um mesmo campo (quando cadastrados em sistemas diferentes). O armazenamento de dados reunidos em um único local assegura maior confiabilidade da informação, além de permitir o compartilhamento com outros setores da organização (ILHA, 1994 e SANCHES, 2005);
- permite o compartilhamento dos dados: em que se podem definir níveis de permissão para utilização e atualização simultânea de um dado por mais de um usuário (SANCHES, 2005);
- restrições de segurança: possibilidade de definir níveis de acesso de acordo com a hierarquia ou necessidade da organização. Dessa forma, alguns usuários terão acesso restrito a alguns campos ou funções, outros não terão nenhum tipo de acesso; enquanto outros poderão ter acesso irrestrito (SANCHES, 2005);
- independência dos dados: a forma física de armazenamento dos dados no banco de dados e a recuperação das informações pelos programas de aplicação são independentes da estrutura interna dos arquivos, permitindo a manutenção dos

programas e inclusão de novos campos sem alteração de todos os programas ou dados (SANCHES, 2005);

- manutenção da integridade: impede que um código ou chave em uma tabela não tenha correspondência em outra tabela (SANCHES, 2005);
- lembretes automáticos: execução automática de procedimentos, de acordo com informações armazenadas. O *triggering* (gatilho) provoca um tipo de resposta quando um determinado procedimento é executado, de acordo com a programação prevista. Diversas funções podem ser previstas nesse recurso, como lembretes de compromissos ou regras do serviço (ILHA, 1994);
- valoração das informações e capacidade de síntese: indicação de valores ou impressões diagnósticas de acordo com determinados parâmetros (SANCHES, 2005);
- facilitação de estudos e pesquisas: o sistema pode prever a possibilidade de comparação de dados, possibilitando que o profissional se utilize dessas informações para estudo de caso ou para pesquisa em geral (SANCHES, 2005).

Dentre as dificuldades observadas para implementação e uso em um sistema de banco de dados, podem ser citadas:

- a forma irregular de arquivamento dos dados originais (manuscritos em papel), sem padronização de termos, abreviaturas e campos, em formato ilegível e desorganizado. O preparo dessas informações para cadastro no banco de dados demanda tempo, mão de obra e gastos adicionais (ILHA, 1994);
- necessidade de implementar elementos como: uma linguagem técnica global, eficaz e normalizada para a comunicação entre os diversos profissionais de projeto; simplificar os processos de produção e o arquivo da documentação técnica

publicada; garantir a consistência e uniformidade na linguagem (dados catalogados, utilização de metadados que facilitam a pesquisa de informação, dentre outros); unificar as características dos diversos objetos, permitindo a sua fácil substituição; minimizar o tempo e os recursos despendidos na comunicação e interpretação dos desenhos; diminuir as dificuldades na organização dos projetos e nos processos de comunicação (por meio da redução das perdas e da redundância da informação); facilitar a partilha da informação; suprimir barreiras de índole técnica; alcançar maior rentabilidade, eficácia e qualidade (MACHADO, 2004);

- vários aspectos do processo de modelagem podem ser dificultados pelo modelador, provocando conflitos posteriores, devido às experiências e conhecimentos específicos do modelador, por exemplo. Um dos mais frequentes conflitos potenciais está relacionado às chaves, suas características e sua forma de utilização (chaves simples e chaves compostas, representação de supertipos e representação de hierarquias) (MACHADO, 2004 e WATSON, 2004);

2.2 Estruturas de Dados

Um conjunto de arquivos (ou tabelas) relacionados configura um banco de dados; os arquivos contêm registros (ou linhas), com campos (ou colunas) evidenciando detalhes de cada dado. Todas as linhas de uma tabela armazenam dados sobre um mesmo tipo de item, da mesma forma que cada coluna da tabela contém o mesmo tipo de dado (WATSON, 2004).

Para que os dados de um banco possam relacionar-se entre si, eles são organizados em tabelas (ou relações) e armazenados de modo a possibilitar sua manipulação em linhas e colunas (MACHADO, 2004).

As tabelas são formadas por conjuntos de colunas (campos ou atributos) e linhas (registros ou *tuplas*). Cada coluna tem um nome único e contém dados diferentes associados às informações de cada linha. As linhas correspondem a registros individuais, correspondentes às colunas e são identificados por uma chave primária (MACHADO, 2004).

A chave primária pode ser uma coluna ou conjunto de colunas onde os dados não podem ser repetidos, como o número de RG, CPF, ou um número atribuído artificialmente, por exemplo. Para evitar redundâncias na chave primária, utiliza-se um número inteiro único. Em bancos de dados do tipo relacional, o identificador é denominado chave estrangeira, representando um relacionamento entre dados de duas tabelas (MACHADO, 2004).

Os relacionamentos podem ser de um para um, de um para muitos ou de muitos para muitos, de acordo com o número de dados em cada tabela relacionada. Quando uma linha é relacionada com outra, tem-se o relacionamento de um para um; quando uma linha é vinculada a muitas, tem-se o relacionamento de um para muitos; quando muitas linhas de uma tabela estão relacionadas com muitas linhas de outra tabela, tem-se o relacionamento de muitos para muitos (MACHADO, 2004).

Os bancos de dados relacionais armazenam dados em tabelas (relações matemáticas) interligadas por campos-chave. De acordo com Chen (apud 1976, CHAGAS et al, 2004), o modelo de dados é mais bem entendido em um nível de abstração mais elevado do que o das tabelas, por meio de um esquema conceitual representando entidades, que podem ser entendidas como algo da realidade modelada em que se deseja manter informações no banco de dados, podendo representar tanto objetos concretos quanto objetos abstratos; relacionamentos, que são conjuntos de associações entre entidades; e atributos, que são definidos como características relevantes associadas a cada ocorrência de entidade ou relacionamento.

Já Watson (2004) define o banco de dados relacional como um conjunto de relações, em que cada relação é um termo matemático para uma tabela. Uma relação é conceituada como uma tabela bidimensional, organizada em linhas e colunas.

Conforme explica Watson (2004), nesse tipo de banco de dados, cada linha deve ser identificada com uma chave primária; as tabelas são ligadas por meio dos significados dos dados. O modelo relacional permite que vários registros sejam processados ao mesmo tempo.

Definição de chaves

Vários aspectos do processo de modelagem podem ser dificultados pelo modelador, provocando conflitos posteriores, devido às experiências e conhecimentos específicos do modelador, por exemplo. Um dos mais freqüentes conflitos potenciais está relacionado às chaves, suas características e sua forma de utilização (chaves simples e chaves compostas, representação de supertipos e representação de hierarquias).

Uma chave corresponde a uma ou mais colunas identificadas na descrição de uma tabela, um índice ou uma restrição referencial. A mesma coluna pode fazer parte de mais de uma chave (por exemplo, uma primária e uma estrangeira). Atributos cujos valores no mundo real podem ser duplicados (nomes, por exemplo) não devem ser definidos como chaves de uma tabela (MACHADO, 2004; WATSON, 2004).

Chave Primária

A chave primária de uma relação é o seu identificador único. Pode ser composta por uma coluna ou conjunto de colunas cujos valores, considerando a combinação de valores de todos os campos da coluna, nunca se repetem e que podem ser usadas como um índice para os

demais campos da tabela do banco de dados. A chave primária garante que cada linha possa ser endereçada de maneira única. Em chaves primárias, não pode haver valores nulos nem repetição de colunas, conforme exemplifica a Tabela 2.1.

Se a chave primária é simples (formada por um único campo da tabela), esse campo não pode ter dois ou mais registros de mesmo valor, e também não pode conter nenhum registro nulo. Se a chave primária é composta (formada por mais de um campo), os valores de cada campo podem se repetir, mas não a combinação desses valores.

Uma chave composta é um conjunto ordenado de colunas de uma mesma tabela (na comparação de chaves compostas, os componentes equivalentes das chaves são comparados).

Chave Candidata

Em uma tabela pode haver mais de uma chave com capacidade de identificação única das linhas da tabela, sendo cada uma destas chaves chamada de chave candidata. Uma das chaves candidatas será definida como chave primária e as outras serão chaves alternativas. A chave primária deve ser escolhida entre aquelas que apresentem menor número de atributos ou que seja mais significativa conceitualmente (WATSON, 2004).

Chave Estrangeira

O campo é chave primária de uma tabela e faz parte dos campos de outra tabela; é uma referência de um elemento de uma tabela a um elemento de outra tabela; uma ligação lógica entre as tabelas, conforme exemplifica a Tabela 2.2 (MACHADO, 2004).

É um atributo, possivelmente composto, que é também uma chave primária de uma relação; é a maneira como os relacionamentos são representados. Pode ser entendida como sendo uma ligação que mantém junto um conjunto de tabelas, a fim de formar um banco de dados relacional. A chave primária e a chave estrangeira podem ou não estar em relações diferentes, mas ambas precisam ser definidas em um mesmo domínio.

“O valor da chave estrangeira precisa ser igual a um valor de chave primária, assegurando que cada componente da chave estrangeira precisa ser igual ao componente equivalente de uma chave primária composta” (WATSON, 2004).

Chave Alternativa

Quando há muitas chaves candidatas, uma é escolhida para ser a chave primária e as outras restantes são chamadas de chaves alternativas, conforme exemplifica a Tabela 2.3 (WATSON, 2004).

Tabela 2.1: Registro de departamento exemplificando a chave primária.

Departamentos

Linhas	Colunas	
	Cod_depto	Nome_depto
	D1	Engenharia
	D2	Financeiro
	D3	Marketing

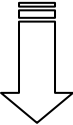
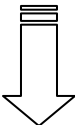
**Chave primária**

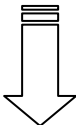
Tabela 2.2: Registro de funcionários exemplificando a chave primária e a chave estrangeira em relação à Tabela departamentos.

Funcionários

Linhas	Colunas		
	Cod_empregado	Nome	Cod_depto
	E1	Souza	D1
	E2	Santos	D2
	E5	Silva	D3



Chave primária

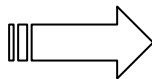


**Chave estrangeira
em relação à tabela
departamentos**

Tabela 2.3: Registro de funcionários exemplificando chave alternativa.

Tabela de Funcionários

Cod_depto	Nome	Cod_empregado	CPF
D1	Souza	E1	122.300.322
D2	Santos	E2	422.550.362
D3	Silva	E5	322.456.122



Chave alternativa

Para se definir dados e relacionamentos que devem ser armazenados no banco de dados, utiliza-se um método de modelagem, cujo objetivo é criar uma representação das necessidades e relacionamentos. Esse documento é resultante de um processo que exige interação do contexto com as necessidades do usuário, sendo que seu resultado pode estar sujeito a revisões e adaptações posteriores. Nessa fase do processo, os diagramas podem ser

desenhados a lápis ou com o auxílio de softwares específicos para a atividade (WATSON, 2004).

2.3 Modelagem

A modelagem de um banco de dados é a construção gráfica de modelos durante o planejamento de um *software* que represente suas características, funcionalidades ou comportamentos de inter-relacionamentos, de acordo com a análise de requisitos. A análise de requisitos é o estudo das características de que o sistema necessita para atender às necessidades e expectativas do cliente (WATSON, 2004).

O objetivo da modelagem de dados é transmitir e apresentar uma representação única, não redundante e resumida dos dados de uma aplicação. Em projetos conceituais de aplicações em bancos de dados, o modelo Entidade-Relacionamento é o mais utilizado para a representação e entendimento dos dados que compõem a essência de um sistema (MACHADO, 2004).

Assim é necessário que o banco de dados seja modelado de acordo com o conjunto de componentes e as relações determinadas, sendo essa atividade realizada, normalmente, por meio da construção de modelos gráficos, representando a estrutura do sistema. O principal objetivo da modelagem é especificar a estrutura geral do banco de dados, facilitando a visualização do projeto (WATSON, 2004).

2.4 Ferramentas de Modelagem

A modelagem de software utiliza uma notação gráfica apoiada por ferramentas, adequando-se como um conjunto de componentes inter-relacionados entre si. Na criação de

sistemas com qualidade, é necessária a utilização de uma boa metodologia para a modelagem tanto do software, como do banco de dados. Para tanto, são construídos modelos gráficos que simbolizam os artefatos de software utilizados e seus inter-relacionamentos (HEUSER, 1999 e BARBIERE, 1994).

Os bancos de dados são modelados por meio do modelo de Entidade Relacionamento (ER), facilitando o projeto e possibilitando especificar a estrutura geral do mesmo. Essa metodologia funciona para modelos conceituais relacionais, objeto-relacionais ou orientado a objetos, conforme descrito por COUGO (1997) e HEUSER (1999):

- **Modelo conceitual** – É o ponto de partida do projeto do banco de dados em que há a descrição de mais alto nível de sua estrutura, não contendo detalhes de implementação. É independente do tipo de Sistema de Gerenciamento de Banco de Dados (SGBD) usado, sendo construídos modelos semânticos, gerando o modelo conceitual de dados;
- **Modelo lógico** – Nesse modelo, é descrita a estrutura do banco de dados, podendo a mesma ser processada pelo SGBD. O modelo conceitual é mapeado para um modelo lógico de dados, dependente da classe de modelos utilizada pelo SGBD, mas não do SGBD escolhido para a implementação. Futuras alterações no modelo lógico devem ser, primeiramente, efetuadas no modelo lógico de dados;
- **Modelo físico** – Realiza o mapeamento do modelo lógico em um esquema físico, de acordo com o SGBD específico, em que são descritos a implantação da base de dados, as estruturas de armazenamento e os métodos de acesso.

A seguir apresentam-se comparações de algumas ferramentas de modelagem, como o *Power Designer*, o *DB Designer* e o *AllFusion ERwin Data Modeler*, utilizadas para

especificar a estrutura para o planejamento dos bancos de dados, facilitando na visualização do projeto e nas suas características.

As ferramentas de modelagem surgiram para facilitar a criação dos modelos, permitindo, inclusive, a geração de uma parte do código-fonte do software a partir do modelo e até a geração do banco de dados, a partir do modelo de entidade relacionamento (KORTH; SILBERCHATZ e SUDARSHAN, 1999).

Para facilitar a compreensão das características das ferramentas de modelagem de dados, segue uma breve explicação sobre os recursos mais importantes que as mesmas podem disponibilizar:

- **suporte a banco de dados:** a ferramenta oferece suporte a bancos de dados como o *Oracle*, *MySQL*, *PostgreSQL*, *MSSQL*, *Interbase/Firebird*;
- **criação de modelos:** permite a criação de modelos de dados do tipo conceitual, lógico e físico;
- **independência entre modelos:** possibilita a independência entre os modelos conceitual, lógico e físico;
- **modelo lógico x modelo físico:** permite que, a partir do modelo lógico, seja gerado o modelo físico;
- **engenharia reversa:** permite engenharia reversa. Assim, objetos existentes em um banco de dados ou *script* podem ser importados para um modelo de dados;
- **importação de modelos:** a ferramenta importa modelos gerados de outras ferramentas de modelagem de banco de dados;
- **geração de scripts:** capacidade de gerar *scripts* para criação de banco de dados a partir do modelo de dados;
- **suporte a elementos da modelagem de dados:** a ferramenta suporta a maior parte dos elementos da modelagem de dados, como tipos de dados, chaves

primária e estrangeira, índices, funções, *stored procedures*, *views* e entidades fracas;

- **representação automática de relacionamentos:** permite a implementação automática de relacionamento entre tabelas (entre chaves estrangeiras e chaves primárias).

Tabela 2.4: Comparativo das características de ferramentas de modelagem.

Características das ferramentas de modelagem			
Características importantes	Ferramentas		
	<i>Power Designer</i>	<i>Erwin</i>	<i>DB Designer</i>
Freeware	Não	Não	Sim
Multi plataforma	Windows	Windows	Windows e linux
Suporte a bancos	ADABAS, DB2, Informix, Interbase/Firebird, Access, SQL Server, MySQL, Oracle, PostgreSQL)	Oracle, DB2/UDB, SQL Server, Teradata, ODBC, Sybase, Informix, Ingres, Progress, Access	MySQL, Oracle, MSSQL, SQLite
Permite engenharia reversa	Sim	Sim	Sim
Criação de modelos	Cria modelo lógico e físico	Cria modelo lógico e físico	Cria somente o modelo físico
Independência entre modelos	Sim	Sim	Não
Importação de modelos	Sim	Sim	Sim
Geração de scripts	Sim	Sim	Sim
Suporte a elementos da modelagem de dados	Sim	Sim	Sim
Representação automática de relacionamentos	Sim	Sim	Sim

Na Tabela 2.4, são comparadas as características de três das principais ferramentas de modelagem disponíveis atualmente (ZAIDAN, 2003).

Conforme se observa, todas as ferramentas analisadas obtêm padrões de modelagem de dados muito parecidos. Porém, o *Erwin* é uma ferramenta *case* para modelagem de dados, relacional e dimensional, que permite a construção de modelos de dados lógicos (DER) e modelos de dados físicos. O *Erwin* caracteriza-se, também, pelo fácil manuseio e rápido entendimento das funcionalidades, sem deixar de disponibilizar recursos avançados para a construção, documentação, padronização, disponibilização e consulta dos modelos de dados. Tais características apontam essa ferramenta para desenvolvimento dos bancos de dados utilizados neste trabalho.

2.5 Servidores de Bancos de Dados

O processo de armazenamento de dados define as estruturas utilizadas para a organização e a representação dos dados, tarefa que exige o uso de muita memória computacional. Uma opção, nesse caso é armazenar os bancos de dados em um servidor e, quando necessário, transferi-los para a memória principal. Atualmente, os equipamentos disponíveis apresentam capacidade suficiente para armazenar grandes bancos de dados, minimizando as perdas de velocidade que ocorrem quando são acessados em recursos de memórias secundárias. De qualquer forma, os gerenciadores de bancos de dados possuem estruturas para otimizar o acesso, que são os índices para os dados (SILVA, 2002).

O SGBD tem por objetivo estabelecer o ambiente adequado e eficiente para controle do banco de dados, propiciando o armazenamento e a recuperação das informações (SILVA, 2002).

Características fundamentais

Destacam-se entre os recursos mais importantes de um servidor de bancos de dados os seguintes:

referential Integrity: conhecido como "integridade referencial"; esse recurso consiste em restrições ou regras existentes para uma correta inserção de dados, por exemplo, para impedir que uma tabela seja preenchida sem que isso ocorra em outra;

transactions: também conhecidas como transações; são instruções executadas em um bloco designado por parâmetros que indicam seu início e seu fim;

views: consistem em um tipo de tabela virtual, formada por campos extraídos de uma tabela "verdadeira", facilitando o controle dos dados acessados;

trigger: também chamados de gatilhos; são recursos que permitem o acionamento de uma sequência de comandos logo em seguida, ou logo após, a um evento;

SQL: o *Structured Query Language* é uma linguagem padrão de comunicação com base de dados. Uma linguagem normalizada, que permite trabalhar com qualquer tipo de linguagem (ASP ou PHP), em combinação com qualquer tipo de base de dados (*Microsoft Access, SQL Server, MySQL* etc);

Stored Procedures: esse recurso consiste em comandos *SQL* "guardados" no servidor, por exemplo, executar tarefas repetitivas, evitando que um cliente tenha que as executar constantemente;

tabela temporária: é uma tabela com vida útil, de uma sessão ou transação. Ela está vazia quando a sessão ou transação começa e descarta os dados ao fim da sessão ou transação;

XML: é uma meta-linguagem de marcação semelhante à HTML, com formato específico para troca de dados;

replicação: é a capacidade dos bancos de dados serem duplicados para outros servidores. Ela tem o objetivo de aumentar a velocidade e a confiabilidade de uma estrutura de banco de dados.

Para realizar a distribuição de dados, existem vários SGBD's disponíveis; entre eles, podem-se destacar pelos seus recursos, complexidades e desempenho: *Oracle, Microsoft Sql Server, DB2, PostgreSQL, Firebird/Interbase, Informix, MySql e Microsoft Access*. Na tabela 2.5, são comparados recursos dos cinco principais bancos de dados disponíveis atualmente: *Oracle, Microsoft Sql Server, PostgreSQL, MySql e Microsoft Access* (BABCOCK, 2008).

Tabela 2.5: Comparativo das características dos Servidores de Bancos de Dados.

Características importantes	Servidores de Bancos de Dados				
	Microsoft SQL Server	Oracle	PostgreSQL	MYSQL	Microsoft Access
Freeware	Sim	Não	Sim	Sim	Não
Sistema Operacional	Windows	Windows, Mac, Linux, Unix	Windows, Mac, Linux, Unix	Windows, Mac, Linux, Unix	Windows
Tamanho máximo do banco	4GB	Ilimitado	Ilimitado	Ilimitado	2GB
Referential Integrity	Sim	Sim	Sim	Sim	Sim
Transactions	Sim	Sim	Sim	Sim	Sim
Views	Sim	Sim	Não	Não	Não
Trigger	Sim	Sim	Sim	Sim	Não
Function	Sim	Sim	Sim	Sim	Não
Stored Procedures	Sim	Sim	Sim	Sim	Não
Tabela temporária	Sim	Sim	Sim	Sim	Não
Suporte à XML	Sim	Sim	Sim	Não	Não
Busca Textual	Sim	Sim	Sim	Não	Não
Suporte à Replicação	Sim	Sim	Sim	Sim	Sim

Todos os bancos de dados analisados apresentam benefícios, mas, de acordo com a tabela 2.4, selecionamos os bancos de dados: *Microsoft Sql Server* e *Microsoft Access* para desenvolver os trabalhos nesta pesquisa, pois o *Sql Server* é otimizado para aplicações complexas, isto é, que envolvem grandes volumes de dados ou que tratam de informações críticas. Além de fornecer um ambiente eficiente e bem elaborado para todas as necessidades relacionadas aos dados. E o *Access*, por sua vez, é focado na facilidade e rapidez para desenvolver os aplicativos, devido a seu poder de simplificação nas soluções.

CAPÍTULO 3

APLICAÇÕES DE BANCOS DE DADOS NA ÁREA BIOMÉDICA

A biomédica e a área médica estão cada vez mais integradas ao mundo da informática, visto que aplicativos podem ser desenvolvidos para fins específicos em que podem facilitar e agilizar o atendimento do paciente. Considera-se, por conseguinte, que a utilização de tais bancos de dados pode representar um importante avanço nas ações de avaliação e melhoria da qualidade dos serviços de saúde. Assim esses bancos de dados são projetados para organizar as informações dos pacientes e, em alguns casos, permitem a associação das informações textuais com informações gráficas, provenientes de exames. Com o advento das imagens digitais, hoje é possível realizar vários processamentos para mostrar, de forma objetiva, os pontos de interesse, necessários para o diagnóstico médico. Porém, para que isso seja possível, as informações necessitam de um meio de gerenciamento e armazenamento.

3.1 Aplicações de bancos de Dados para imagens médicas

Sistemas de imagens médicas estão entre as mais modernas tecnologias desenvolvidas para a área de diagnóstico médico, tendo, portanto, um expressivo impacto na área de saúde.

O volume de imagens médicas geradas por esses sistemas tem crescido enormemente, devido ao aumento tanto do número de exames executados quanto na quantidade de modalidades disponíveis desse tipo de sistema. Para uma melhor compreensão do grande volume de dados médicos e biomédicos, gerados por vários equipamentos de diagnóstico por imagens, surge a necessidade de desenvolver técnicas adequadas para o processamento e visualização desses dados.

Stensaas e Sorenson (1988) desenvolveram, na Universidade de Utah, em Salt Lake City (EUA), o primeiro programa para educação médica assistida por computador, usando o *HyperCard* (um programa de desenvolvimento de hipertextos) e o videodisco "The Slice of Life", para o treinamento em neurociências. Esse programa, chamado *HyperBrain*, serve para a educação médica em diversos níveis: graduação, residência, pós-graduação e educação médica continuada; e provou seu valor como um incremento aos métodos de educação tradicionais, utilizando o banco de dados *Oracle*. O banco de dados contém, aproximadamente, 30.000 imagens, enfatizando as áreas morfológicas como: histologia, radiologia, neuroanatomia, hematologia, citologia, patologia etc (STENSAAS e SORENSON, 1988).

O aplicativo foi desenvolvido para ser um ambiente educacional para vários níveis e cursos. Utilizando o mesmo material disponível no "Slice of Life", foram desenvolvidos diversos outros softwares hipermídia de ensino, tais como o *HyperHeart* e outros (STENSAAS e SORENSON, 1988).

Na mesma linha, outro software de hipermídia médica, de grande influência, mais genérico que o *HyperBrain*, foi desenvolvido na Universidade de Stanford por Freedman et Chase (1989). O "livro eletrônico", que recebeu o sugestivo nome de Cadáver Eletrônico, é constituído de imagens do corpo humano (macroscópicas e histológicas), textos, gráficos computadorizados, vários tipos de ilustrações dinâmicas (interagindo simulações

computadorizadas animações) e sons (VOLPE e SABBATINI, 1994). O programa foi implementado em *HyperTalk* e pode ser modificado por programadores de *HyperCard*, utilizando o *Sql Server* (VOLPE e SABBATINI, 1994).

Como seria de se esperar, as especialidades médicas que utilizam imagens, como a radiologia, a ultra-sonografia, a medicina nuclear, biomédica etc, têm muito a lucrar com o uso de softwares de multimídia, voltados ao ensino. Para essa finalidade, Mezrich et al. (1989) desenvolveram um banco de dados para área de radiologia, denominada *ImageNet*, com o *software HyperCard*. As aplicações desse sistema incluem: aquisição, demonstração e análise de imagens; inserção de imagens em relatórios e gráficos de pacientes; geração de imagens para palestras e seminários; banco de imagens com rápido acesso para revisão e consulta; arquivo de ensino simplificado, de baixo custo para armazenamento e rápida recuperação; bancos de dados de patologia e imagens de referência. As imagens são adquiridas em segundos e mostradas na mesma resolução e escala de cinza em que são vistos no sistema de imagem médica original (VOLPE e SABBATINI, 1994).

O banco de dados é composto por imagens combinadas com informações clínicas e demográficas que incluem detalhes sobre as imagens usadas, permitindo recuperar informações gerais, para serem usadas em consultas clínicas, estudo comparativo e arquivos de ensino (VOLPE e SABBATINI, 1994).

A existência de um recurso desse tipo é extremamente valiosa para o ensino da imagenologia médica, como se pode detectar. Outro projeto semelhante é o *Image Engine* (LOWE, 1993), um gerenciador de base de dados multimídia, orientado a objeto, para facilitar o armazenamento, recuperação, compartilhamento e manipulação de materiais médicos multimídia, como imagens estáticas biomédicas, vídeos e textos, tendo como característica interessante o uso de um dicionário padronizado para codificação de todo o material, baseado

no vocabulário da *National Library of Medicine (Medical Subject Headings - MeSH)* (VOLPE e SABBATINI, 1994).

Jaffe et al (1989) construiu um programa hipermídia para aprendizagem de ecocardiografia que inclui um ambiente controlado pelo usuário, com um acesso instantâneo a 54.000 quadros de vídeo, ligados a 1.200 questões clínicas. O programa envolve gráficos, animações, texto e sons digitalizados e também foi desenvolvido em *HyperCard*, usando microcomputadores Macintosh e um monitor separado para exibir as imagens. O programa apresenta gráficos de anatomia, diferentes tomadas do coração, quadros relevantes para o diagnóstico, animações, além de conter bases de dados catalogados e áudios digitalizados de ausculta cardíacas de diagnoses selecionadas. Cada diagnóstico é descrito por textos: sobre o quadro clínico, sobre a etiologia e sobre os achados, e é ilustrado por seqüências de vídeo de casos, história desses casos, áudio e animações; também há desenhos anatômicos, um glossário e um help para *software* e *hardware* (VOLPE e SABBATINI, 1994).

Em (2002), Geraldeli et al. criou um banco de dados para o Serviço de Radiodiagnóstico do Hospital Universitário Clementino Fraga Filho (HUCFF) da Universidade Federal do Rio de Janeiro (UFRJ), com o objetivo de evitar a deterioração das imagens pelo tempo e tornar a manipulação dos dados clínicos dos pacientes mais ágeis. O autor armazenou os dados de cada paciente e suas respectivas imagens radiológicas digitalizadas. O arquivo de imagens era composto por exames de tomografia computadorizada (TC), ultrassonografia (US), radiologia convencional (RX), medicina nuclear (MN) e mamografia. Foi utilizado para gerenciamento de banco de dados *Access 2000*. As imagens obtidas foram arquivadas em computador e, posteriormente, gravadas em discos passando, quando necessário, por um processo de edição, no qual foi utilizado o aplicativo *Adobe PhotoDeluxe 2.0* (software com recursos de processamento de imagem). Para a exibição das imagens digitalizadas, utilizou-se um programa de gerenciamento de imagem chamado

Irfanview (software visualizador de imagens com recursos de edição e conversão de formatos de imagens, além de compatibilidade com formatos de áudio e vídeo), que pode ser obtido de maneira gratuita na Internet e que possibilita melhor manipulação das imagens obtidas (GERALDELI, 2002).

Para a coleta e armazenamento de dados e imagens foi criado um formulário de papel, no intuito de facilitar e orientar a coleta das informações relevantes de cada paciente (GERALDELI, 2002).

Os dados foram organizados em duas tabelas: tabela imagem e tabela paciente (GERALDELI, 2002).

Na tabela imagem, constam os seguintes itens:

- prontuário, imagem e descrição.

Na tabela paciente, constam os seguintes itens:

- prontuário, código, nome, sexo, cor, idade, queixa principal, histórico da doença atual, achados laboratoriais, achado cirúrgico, anatomia patológica, laudo, observação, publicação, entrada no arquivo e imagem.

A seguir, apresentamos algumas interfaces do aplicativo sistema de controle de pacientes, desenvolvido para armazenamento de dados e imagens dos pacientes.

A Figura 3.1 mostra a Interface Inicial do aplicativo de Geraldeli (2002), desenvolvido para cadastrar dados e imagens dos pacientes.

O menu de controle possui as opções de acesso ao cadastro de pacientes, cadastro de consultas, cadastro de imagens ou fechar a tela.

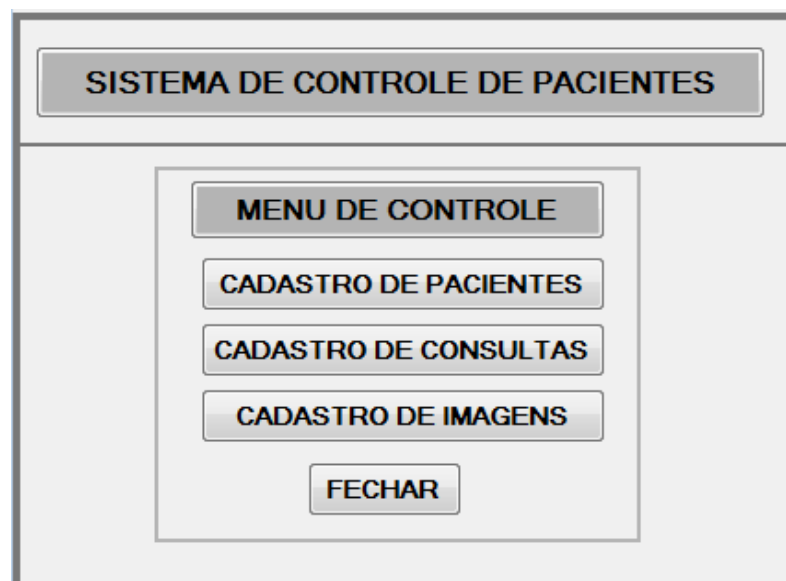


Figura 3.1: Interface Inicial do aplicativo sistema de controle de pacientes para cadastro de dados e imagens. Fonte: Geraldeli et al., 2002, p. 28.

Se o usuário desejar acionar o botão referente ao cadastro de pacientes, ele observará o formulário mostrado na Figura 3.2, contendo as informações necessárias do paciente, como: nome, queixa principal, histórico da doença, anatomia patológica etc.

Figura 3.2: Cadastro das principais informações de pacientes do aplicativo sistema de controle de pacientes. Fonte: Geraldeli et al., 2002, p. 28.

Se o usuário optar pelo cadastro de consultas, será mostrado qual o paciente que será consultado, o prontuário, queixa, exame e o laudo, conforme apresentado na Figura 3.3.

Figura 3.3: Cadastro de consulta de pacientes do aplicativo sistema de controle de pacientes.
Fonte: Geraldeli et al., 2002, p. 29.

O banco de dados foi composto, inicialmente, por aproximadamente 700 exames. A digitalização das imagens permite, inclusive, a sua impressão em papel e posterior distribuição aos alunos para estudo ou uso em aula (GERALDELI, 2002).

Em 2004, Pires, Medeiros e Schiabel desenvolveram um software para treinamento na interpretação de imagens mamográficas em sistemas digitais. O programa foi desenvolvido em *Delphi6*, associado a uma base de dados em *Interbase 6*, com a finalidade de armazenar imagens de exames mamográficos associados aos seus laudos em categorias *BI-RADS* (o software compara o laudo do usuário com as informações do banco de dados, apontando acertos e erros da interpretação) (PIRES; MEDEIROS e SCHIABEL, 2004).

O software permite visualizar as projeções das mamas em diversas posições (individualmente, médio lateral oblíqua e crânio caudal, ou as quatro imagens

simultaneamente), permite acessar as imagens e os laudos existentes ou interpretar as imagens utilizando o sistema de categorias BI-RADS e disponibiliza o laudo, conforme mostrado na Figura 3.4 (PIRES; MEDEIROS e SCHIABEL, 2004).

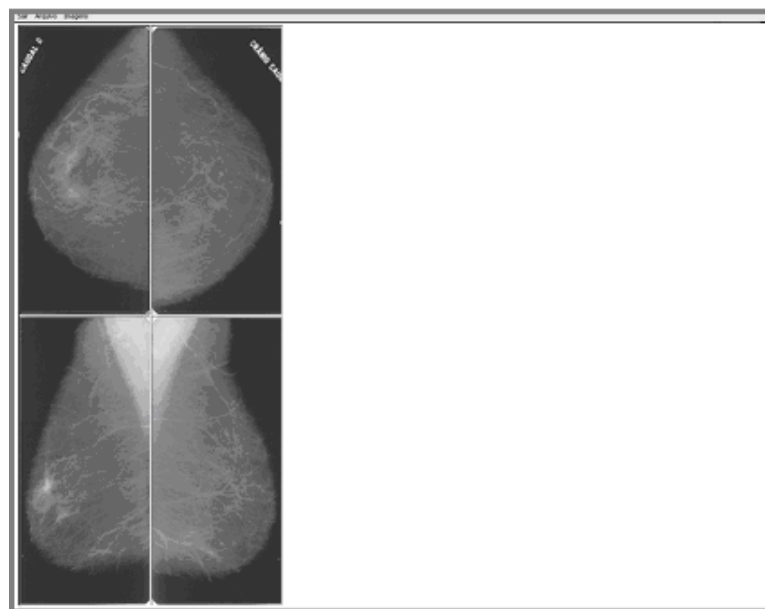


Figura 3.4: Tela que mostra as quatro imagens mamográficas de um exame em diversas posições do software para treinamento na interpretação de imagens mamográficas em sistemas digitais. Fonte: Pires, Medeiros; Schiabel, 2004, p. 244.

No software, foi criada uma função que permite o treinamento de profissionais da área. Com o auxílio desta, o usuário busca um exame de interesse no banco de dados e efetua o diagnóstico, utilizando o mesmo padrão do sistema *BI-RADS*. No final do processo, o software faz a comparação do resultado da interpretação simulada com o armazenado no banco de dados, obtendo uma pontuação ponderada de acertos (PIRES; MEDEIROS e SCHIABEL, 2004).

A tela do software possibilita o treinamento na visualização dos exames mamográficos, juntamente com o seu laudo em sistemas de categorias *BI-RADS*, além da avaliação do desempenho, comparando o laudo simulado com o armazenado no banco de dados. Durante o treinamento, é possível que o usuário visualize os parâmetros técnico-

operacionais e de controle de qualidade do período em que foram geradas as imagens, bem como o índice de qualidade das mesmas (PIRES; MEDEIROS e SCHIABEL, 2004).

Alternativamente, se o usuário não desejar fazer um treinamento, existe uma opção em que ele pode estudar o exame já interpretado no próprio monitor, com a finalidade de se adaptar à tecnologia digital (PIRES; MEDEIROS e SCHIABEL, 2004).

Em qualquer fase do processo de apresentação das imagens no monitor, o usuário pode fazer uma consulta sobre o índice de qualidade da imagem, parâmetros técnicos operacionais e de controle de qualidade sob os quais estão associadas (PIRES; MEDEIROS e SCHIABEL, 2004).

A Figura 3.5 representa a tela do software que mostra o grupo de imagens crânio caudal, no mesmo tamanho físico representado pelo filme mamográfico, com o objetivo de avaliar, de forma mais eficiente, suas estruturas internas. Analogamente, é possível a visualização do grupo de imagens médio lateral (PIRES; MEDEIROS e SCHIABEL, 2004).

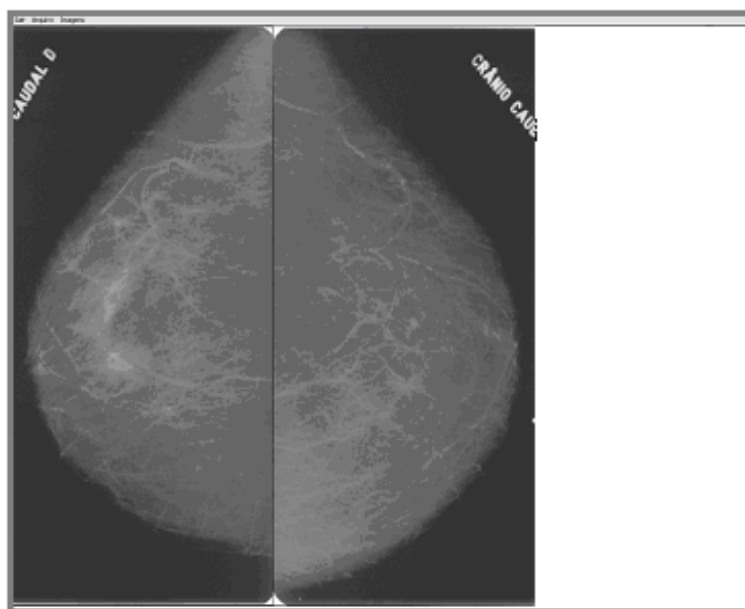


Figura 3.5: Tela que mostra o grupo crânio caudal, no mesmo tamanho físico, com o objetivo de avaliar de forma mais eficiente as estruturas internas, desenvolvido pelo software para treinamento na interpretação de imagens mamográficas em sistemas digitais. Fonte: Pires, Medeiros; Schiabel, 2004, p. 246.

O software possibilita visualizar a imagem do simulador *RMI-156* (utilizado para monitoramento de sistemas de mamografia) e o índice de qualidade, assim como os parâmetros técnicos descritos a seguir:

- dados referentes ao equipamento mamográfico: local de instalação, modelo, fabricante (PIRES; MEDEIROS e SCHIABEL, 2004);
- dados referentes ao processamento: temperatura do fixador/revelador/água, sensitometria, fabricante e modelo da processadora (PIRES; MEDEIROS e SCHIABEL, 2004);
- dados referentes aos insumos: fabricante e modelo das telas, filmes e agentes químicos (PIRES; MEDEIROS e SCHIABEL, 2004).

Esse software apresenta como vantagens, o baixo custo e a facilidade de acesso devido à interface amigável com o usuário; e está disponível na intranet da UNIFESP e será disponibilizado aos profissionais como parte integrante do programa de treinamento do Depto de Diagnóstico por Imagem da Universidade Federal de São Paulo (PIRES; MEDEIROS e SCHIABEL, 2004).

Em 2009, Delfino e Nunes desenvolveram um trabalho utilizando técnicas de PI (Processamento de Imagens) integradas a técnicas de RV (Realidade Virtual). Com a utilização de técnicas de PI, é possível extrair atributos de tamanho, forma e localização nas imagens mamográficas, a fim de fornecer parâmetros para modelagem dinâmica de objetos sintéticos 3Ds, permitindo representar as mamas e os nódulos das imagens reais (DELFINO e NUNES, 2009).

A escolha da tecnologia de software considerou alguns requisitos previamente estabelecidos, de performance, custo e possibilidade de reutilização de código, além, obviamente, de prover facilidades para implementação de ambientes virtuais (AVs) (DELFINO e NUNES, 2009).

A implementação começou com a investigação de técnicas de PI que poderiam ser aplicadas para, extrair características da área que representa a mama nas imagens mamográficas 2Ds (altura, largura, localização do mamilo, entre outras) (DELFINO e NUNES, 2009).

Após a geração do arquivo no formato *Object File* (OBJ), contendo as coordenadas 3Ds, da mama, a implementação prosseguiu com a investigação de técnicas de PI que poderiam ser aplicadas para extrair características de nódulos, (altura, largura e localização do nódulo, entre outras) de imagens mamográficas (DELFINO e NUNES, 2009).

Para a segmentação da mama, foi utilizado o algoritmo proposto por [Goes et al. 2003]. Os autores propuseram a divisão da imagem em nove quadrantes e a análise desses quadrantes em relação à quantidade de *pixels* brancos. O quadrante com maior quantidade de pixels brancos foi selecionado como aquele que contém a localização da mama na imagem. Uma vez obtido o quadrante em que se localiza a mama, é escolhido um *pixel* semente, definido como o ponto médio do quadrante. A partir do *pixel* semente, é aplicado um algoritmo de crescimento de região para delimitar a porção da mama, sendo que os *pixels* pertencentes à região que não foi demarcada são eliminados (DELFINO e NUNES, 2009).

Após a importação dos objetos, sentiu-se a necessidade de aumentar o realismo da cena gerada (DELFINO e NUNES, 2009).

Para isso, foram aplicadas texturas nos objetos 3Ds construídos. Na mama, foi aplicada uma textura de pele, obtida a partir de uma imagem de pele real, e, no nódulo, foi aplicada uma textura na cor vermelha, a fim de facilitar a visualização no modelo *wireframe*. Após a geração do modelo 3D, foi incorporada ao AV (ambiente virtual) uma seringa sintética 3D para permitir ao usuário treinar o procedimento de Punção Aspirativa por Agulha Fina (PAAF) (DELFINO e NUNES, 2009).

A Figura 3.6 mostra a tela de treinamento, na qual são exibidos os dados do paciente e algumas opções de configuração na parte superior. Na parte inferior, são mostradas as visões crânio-caudal (CC) e médio-lateral oblíqua (MLO), do exame mamográfico do paciente, ao lado do AV para o treinamento, onde o usuário tem a possibilidade de usar os 6 graus de liberdade tanto para a mama quanto para a seringa (DELFINO e NUNES, 2009).

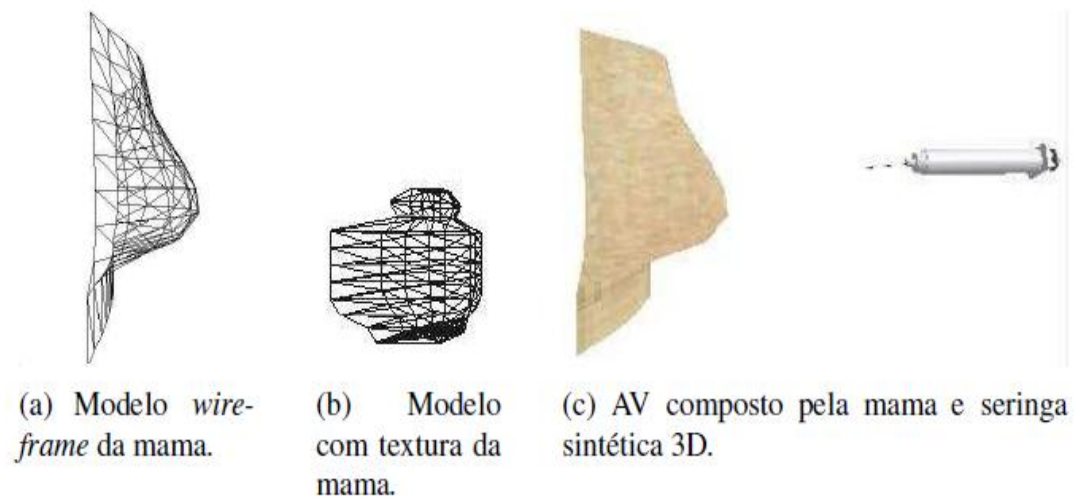


Figura 3.6: Interface para treinamento, na qual são exibidos os dados do paciente e algumas opções de configuração na parte superior. Fonte: DELFINO, S. R e NUNES, F. L. S, 2009, p. 6.

Se a agulha encostar no nódulo, durante o treinamento, ocorrerá a colisão. O fato será avisado para o usuário que está efetuando o treinamento por meio de uma mensagem na tela, conforme exemplifica a Figura 3.7 (DELFINO e NUNES, 2009).

Para ilustrar a utilização da ferramenta serão utilizados dois casos (Figuras 3.8(a) e 3.8 (b) e Figuras 3.8 (c) e 3.8 (d)), em que serão discutidos desde o cadastramento do paciente até o momento em que é efetuado, de modo virtual, o treinamento (DELFINO e NUNES, 2009).

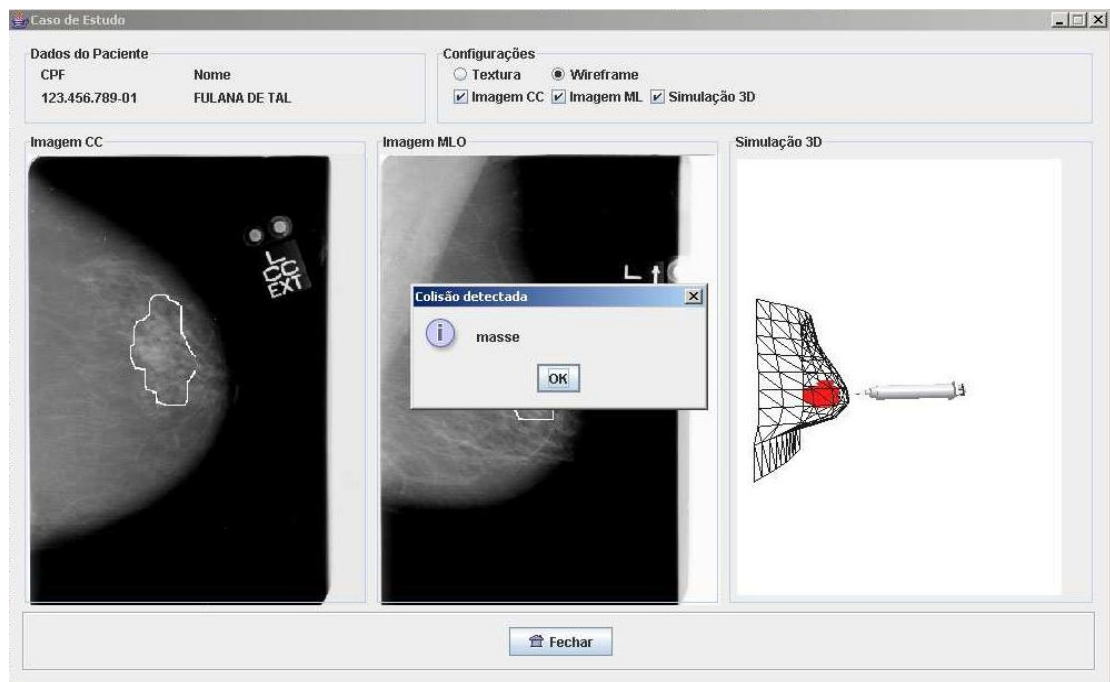


Figura 3.7: Interface para treinamento, na qual mostra a colisão detectada quando o usuário encosta a agulha no nódulo. Fonte: DELFINO, S. R e NUNES, F. L. S, 2009, p. 6.

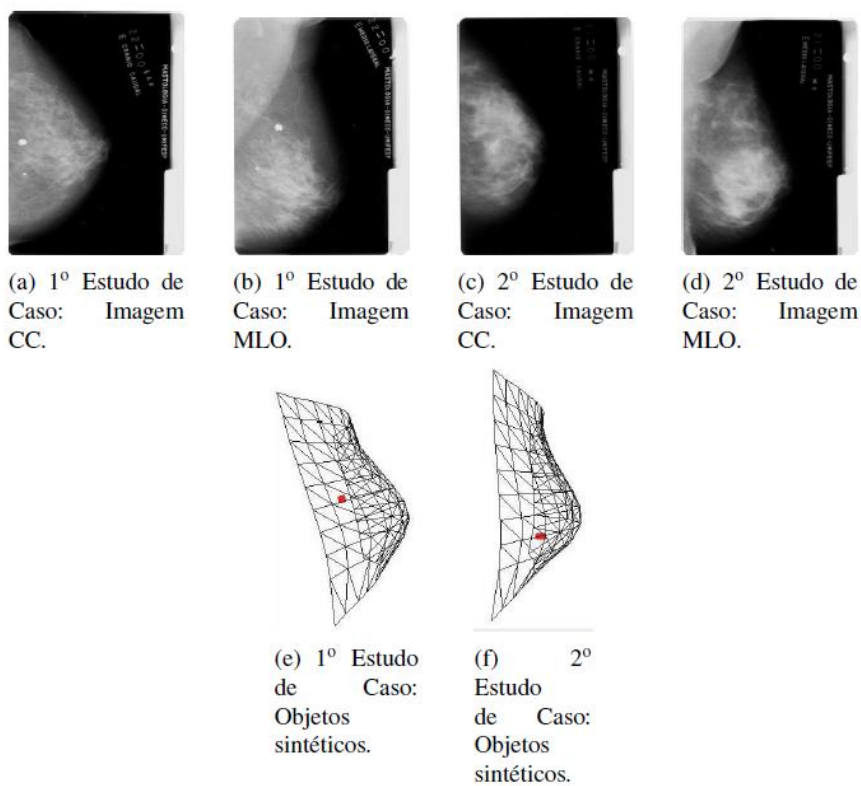


Figura 3.8: Interface utilizada para exemplificação de estudo de caso. Fonte: DELFINO, S. R e NUNES, F. L. S, 2009, p. 7.

O primeiro passo a ser realizado foi submeter as imagens mamográficas a um especialista para que ele analisasse e delimitasse os nódulos, nas visões CC e MLO de cada estudo de caso, em uma ferramenta de editoração gráfica. Após os nódulos terem sido marcados por um especialista, os mamogramas podem ser utilizados como fonte de entrada de informações para o sistema desenvolvido (DELFINO e NUNES, 2009).

A primeira técnica de PI aplicada foi a de redução de escala. Após a operação de escala, foi realizada a binarização da imagem, cujo propósito foi destacar a área da mama do fundo da imagem. Após a fase de binarização, foi aplicada uma dilatação morfológica, para preservar o formato da mama e, em seguida, foram realizadas três sucessivas erosões morfológicas, para isolar a região da mama. O próximo passo consistiu em segmentar a mama. A segmentação da mama foi feita usando de um algoritmo de crescimento de região. Após a fase de segmentação da mama, deu-se início à fase de extração de medidas. Depois da fase de extração de medidas, foi efetuado a descompressão da mama. Depois da fase de descompressão, foi efetuado o alinhamento entre as visões, que é o módulo responsável pelo estabelecimento da correspondência entre as visões CC e MLO. Após a execução das fases anteriores, cujo resultado fornece as medidas necessárias para a construção da mama sintética 3D, os valores obtidos foram normalizados e armazenados em um arquivo no formato OBJ, para posterior recuperação das coordenadas geradas (DELFINO e NUNES, 2009).

Depois de gerado o arquivo no formato OBJ, foram executadas técnicas de PI que permitiram extrair características de nódulos (altura, largura e localização) de imagens mamográficas. A técnica de subtração foi utilizada para segmentar a área da mama do fundo da imagem. Após a eliminação do fundo da mamografia, a imagem mamográfica é binarizada para separar o nódulo do tecido da mama. A partir da imagem binarizada, foi aplicado o algoritmo para extração de medidas do nódulo e, em seguida, o alinhamento entre as visões. Em seguida, foram obtidas as medidas necessárias para a construção do nódulo sintético 3D.

Os valores obtidos foram normalizados e armazenados em um arquivo no formato OBJ. Na Figura 3.8(e), são apresentados os objetos sintéticos 3Ds gerados, referentes ao estudo de caso mostrado nas Figuras 3.8(a) e 3.8(b) e, na Figura 3.8(f), são exibidos os objetos 3Ds gerados, referentes ao estudo de caso mostrado nas Figuras 3.10(c) e 3.10(d) (DELFINO e NUNES, 2009).

Com base nos resultados do algoritmo de extração de medidas, são criados dois arquivos no formato OBJ; um para armazenar a mama sintética 3D e outro para armazenar o nódulo sintético 3D. Após a geração dos arquivos OBJ, os casos podem ser selecionados para treinamento, por meio de uma interface gráfica.

Considerando os parâmetros de desempenho esperados e a facilidade para a construção de interfaces apropriadas, optou-se pela utilização do banco de dados MySQL. A consideração a respeito de possível expansão da base de imagens levou a uma modelagem relacional de banco de dados, de forma a permitir inclusões de novos atributos sem muito esforço de manutenção de programas.

3.2 Aplicações de banco de dados para sinais biomédicos

A análise de sinais biomédicos é um ramo de profundo interesse nas pesquisas de apoio à capacitação física de pessoas com deficiência. Há hoje uma grande tendência de desenvolvimento de tecnologias que, a cada dia, geram uma maior resolução desses sinais. A junção da computação a medicina pode proporcionar maior agilidade e confiabilidade aos exames médicos, planejamento mais eficiente de procedimentos médicos e atos cirúrgicos, além de viabilizar o treinamento de estudantes de medicina por meio de sistemas computacionais baseados nas novas tecnologias.

Carvalho (2003) elaborou um sistema de banco de dados para visualização e armazenamento de sinais biomédicos que consiste no monitoramento remoto de pacientes. A pesquisa teve como objetivo o monitoramento remoto do paciente durante 24 horas por dia. A captura dos sinais é feita por vários sensores ligados ao paciente, que capturam os sinais vitais do paciente, como o ECG, a pressão arterial, a frequência cardíaca, o EMG, o EEG, a temperatura, a oximetria de pulso e a frequência respiratória. Os sinais são capturados por meio de sensores específicos e conectados a um dispositivo que digitaliza e envia os sinais para um computador por meio da porta serial. Esse computador deve verificar, constantemente, esses sinais e, ao encontrar alguma anomalia, decide o que deve ser feito. Essa tomada de decisão não é apenas baseada na análise dos sinais. A visualização dos sinais é um processo que deve ser executado em tempo real. O software para a plotagem dos sinais (Medplot) é um sistema em tempo crítico, pois o atraso na plotagem dos sinais de ECG, EMG e EEG poderá prejudicar a sua análise. No caso de sinais em que a taxa de amostragem é mais baixa, como os sinais de pressão arterial, oximetria de pulso e temperatura, a atualização na tela não afeta a visualização. Foram escolhidos, inicialmente, seis sinais para serem visualizados: ECG, EMG, pressão arterial, frequência cardíaca, oximetria de pulso e temperatura (CARVALHO, 2003).

A interface implementada consiste de uma tela inicial na qual o usuário escolhe qual sinal visualizar. A Figura 3.9 mostra a tela inicial onde o usuário escolhe o sinal que deseja, pressionando com a caneta sobre o botão. Os dados mostrados na tela são simulações ou dados reais capturados (ECG) (CARVALHO, 2003).

No caso do ECG, o sinal é deslizado pela tela da esquerda para a direita. O sinal visualizado corresponde a 2 segundos do ECG no modo retrato (Figura 3.10). Cada divisão do eixo do tempo equivale a 0,2 segundos. A amplitude do sinal segue o padrão de 1mv por 10 mm e cada divisão na tela corresponde a 0,5 mv (CARVALHO, 2003).

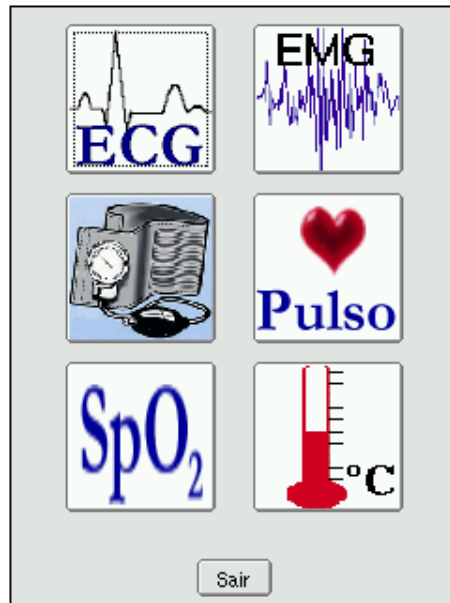


Figura 3.9: Interface Inicial do sistema de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 3.

O sinal do EMG é trabalhado de forma que 3 segundos do sinal sejam mostrados na tela. Para tanto, pode ser necessário interpolar ou dizimar amostras. A Figura 3.11 é a captura da tela com o sinal de EMG (CARVALHO, 2003).

A visualização da pressão arterial (Figura 3.12) mostra, na tela, as últimas 10 amostras, em um gráfico e em uma tabela, de modo a permitir ao profissional da saúde acompanhar o desenvolvimento do usuário ao longo do tempo. No gráfico, o eixo das ordenadas mostra a pressão em mmHg e os valores correspondem aos valores relativos às pressões arteriais sistólica e diastólica das amostras. Os valores do eixo das abscissas correspondem à numeração das amostras. A tabela abaixo do gráfico mostra os valores de cada amostra e a hora na qual a medição foi realizada (CARVALHO, 2003).

Os dados da frequência cardíaca, oximetria de pulso e temperatura do usuário são mostrados nas Figuras 3.13, 3.14 e 3.15, respectivamente. Seguindo o mesmo padrão do gráfico de pressão arterial, eixo das ordenadas mostra-se o valor máximo e mínimo do período

e, no eixo das abscissas, a numeração das amostras. A tabela abaixo de cada gráfico mostra o horário da medição e o seu valor (CARVALHO, 2003).

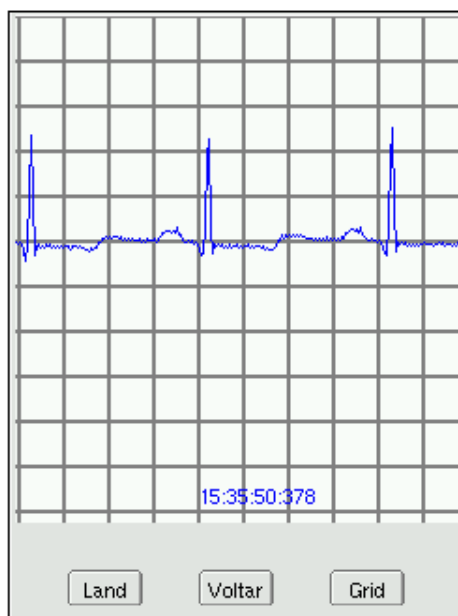


Figura 3.10: Resultado do sinal ECG de um paciente armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 3.

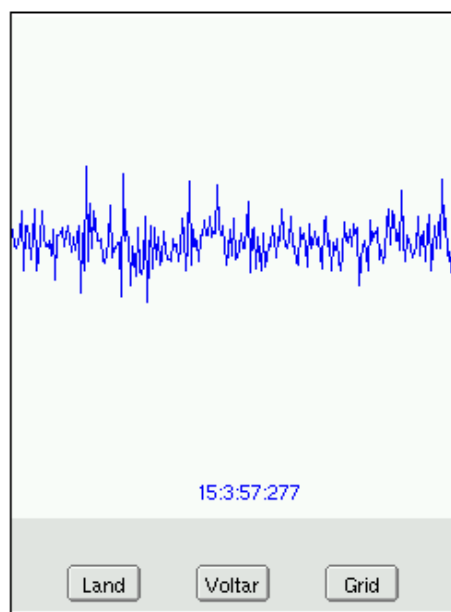


Figura 3.11: Resultado do EMG de um paciente armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 3.

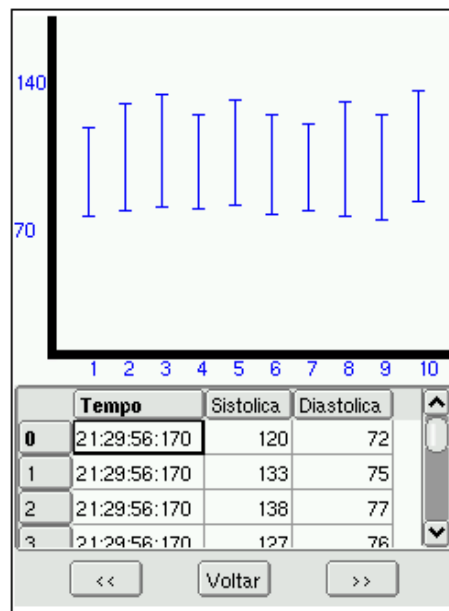


Figura 3.12: Resultado das últimas 10 amostras da Pressão Arterial de um paciente armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.

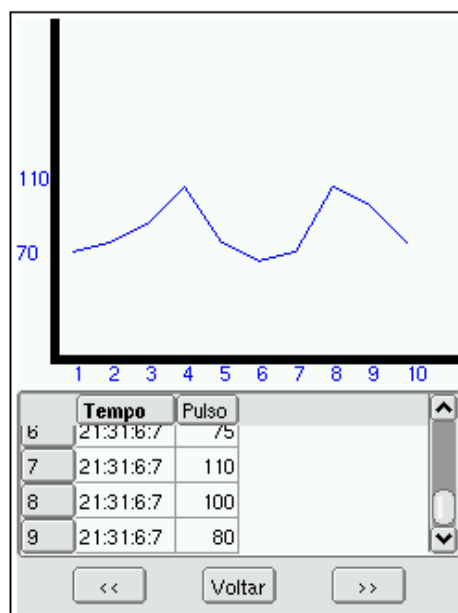


Figura 3.13: Resultado da Frequência Cardíaca de um paciente, armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.

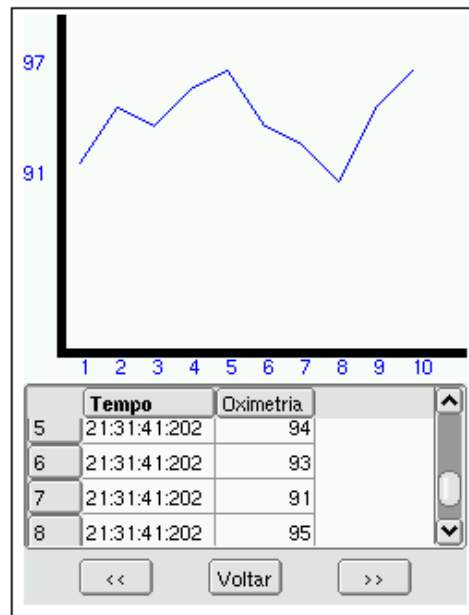


Figura 3.14: Resultado da Oximetria de pulso de um paciente, armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.

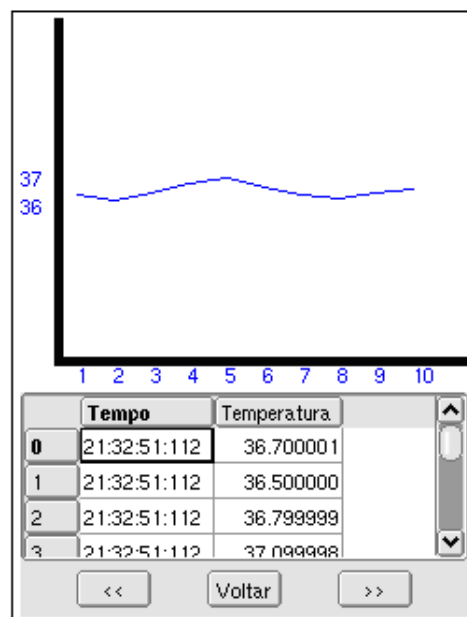


Figura 3.15: Resultado da Temperatura de um paciente, armazenado pelo aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 4.

A Figura 3.16 mostra os resultados do sinal de ECG, EMG e a pressão arterial de um paciente para monitoração remota (CARVALHO, 2003).

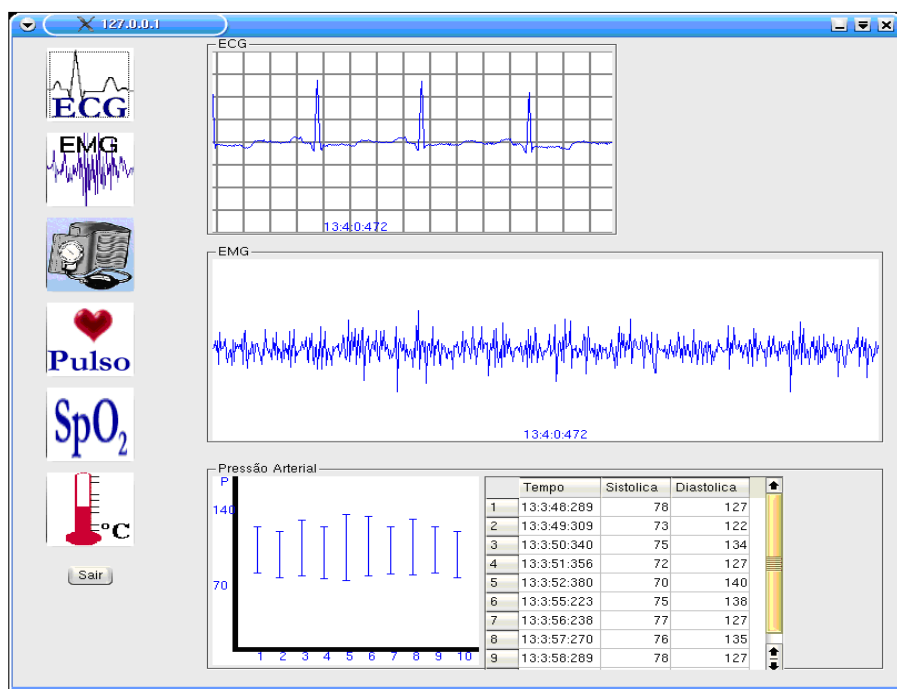


Figura 3.16: Interface para monitorização remota dos sinais do aplicativo de banco de dados para visualização e armazenamento de sinais biomédicos. Fonte: CARVALHO, S. H, 2003, p. 5.

Comprovadamente, as simulações clínicas computadorizadas, mesmo que baseadas exclusivamente em textos, são muito eficazes para promover o ensino da resolução de problemas em Biomédica. As simulações clínicas multimídia, por outro lado, enriquecem ainda mais essa tecnologia, ao permitirem adicionar sons, imagens, animações, sinais etc (CARVALHO, 2003).

SÁ (2004) desenvolveu um sistema em tempo real para relaxamento, utilizando neurofeedback. A linguagem de desenvolvimento deste sistema é o Visual C++, plataforma Windows. O desenvolvimento da modelagem 3D foi feita utilizando o 3D Studio Max (um programa de modelagem tridimensional que permite renderização de imagens e animações) e

usamos o DirectX 8.0 SDK para a renderização em tempo real. O sistema está preparado para fornecer ao paciente os seguintes tipos de biofeedback:

- auditivo: o som se altera de acordo com a variação do sinal;
- visual: ambientes com elementos 3D que são controlados pelos sinais captados pelos sensores e gráficos mostrando os sinais coletados.

Para facilitar o acompanhamento e a evolução do paciente, o sistema também está preparado para realizar as seguintes tarefas:

- cadastrar pacientes: nesta unidade são armazenados os dados pessoais, clínicos (altura, pressão arterial, peso e vícios), anamnese e tratamento proposto;
- armazenar as sessões realizadas: armazena todos os sinais coletados durante a sessão.

O terapeuta pode também armazenar o seu comentário acerca do desempenho do paciente;

- gerar relatórios: os relatórios podem ser referentes a todas as sessões realizadas ou a uma sessão específica;
- mostrar a estatística dos sinais armazenados em uma determinada sessão.

Para armazenar os dados dos pacientes e das sessões realizadas, utilizou-se um banco de dados, *Microsoft Access*. Para tal, foram criadas 3 tabelas: paciente, dados clínicos e sessões realizadas.

A biblioteca de som foi desenvolvida para ser possível fornecer um biofeedback sonoro para o paciente e/ou para proporcionar um som ambiente para a terapia. Ela foi criada utilizando como base as bibliotecas DirectMusic e o DirectSound (ambas são partes do DirectX 8.0 SDK MSWindows) e possuem as seguintes responsabilidades:

- tocar sons wave e midi;
- gerenciar a execução de diversos sons simultaneamente;
- controlar o volume, distorção, eco etc, de acordo com a variação do sinal de feedback.

A biblioteca de vídeo foi criada para tocar vídeos especialmente selecionados para cada terapia animados. Ela foi criada utilizando o DirectShow (parte integrante do DirectX 8.0 SDK MS-Windows) e possui as seguintes responsabilidades:

- tocar vídeos no formato avi, mpeg, mpg e mov;
- controlar a execução de um vídeo, ajustando-a a uma textura 3D;
- controlar o volume da música presente no vídeo.

A biblioteca 3D foi criada utilizando o Direct3D (parte integrante do DirectX 8.0 SDK MS-Windows), e com ela podemos importar os objetos 3D criados no 3Dmax e, juntamente com o Visual C++, fazer a renderização em tempo real do ambiente 3D. Com os objetos em 3D, é possível realizar as animações necessárias para fornecer o biofeedback ao paciente.

Com essas bibliotecas multimídia, foram desenvolvidos os seguintes ambientes:

- Ambiente Circo

O objetivo do paciente é manter a gangorra equilibrada e não deixar a bola cair. Isto só acontecerá quando ele atingir o limiar que o terapeuta definir (Figura 3.17).

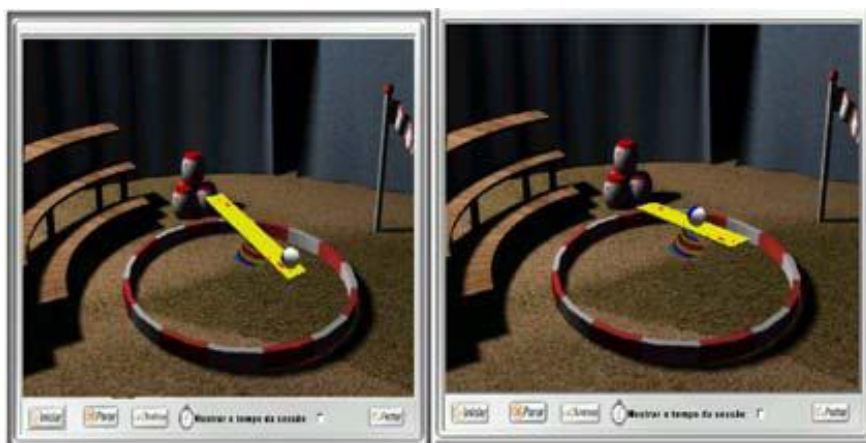


Figura 3.17: Interface para relaxamento através da técnica de neurofeedback usando o ambiente circo. Fonte: SÁ, A. A. R, 2004, p. 3.

- Ambiente Neblina

Para poder visualizar a imagem (ou filme) que está encoberto pela neblina, o paciente deverá atingir o padrão de relaxamento desejado (Figura 3.18).



Figura 3.18: Interface para relaxamento através da técnica de neurofeedback usando o ambiente neblina. Fonte: SÁ, A. A. R, 2004, p. 3.

- Ambiente Quebra-Cabeça

No início da sessão, um quadro se quebra em vários pedaços, como se fosse um quebra-cabeça. Para remontar o quadro o paciente deverá relaxar (Figura 3.21).

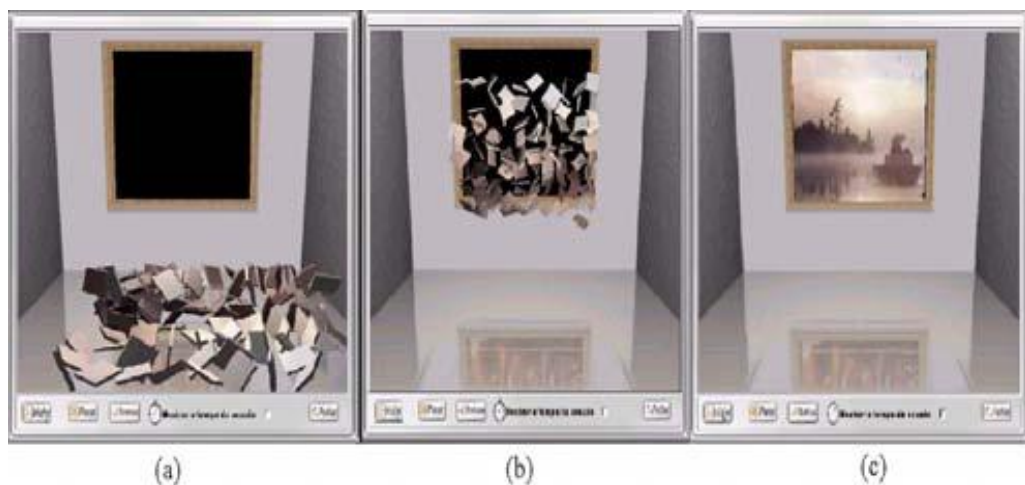


Figura 3.19: Interface para relaxamento através da técnica de neurofeedback usando o ambiente quebra-cabeça. Fonte: SÁ, A. A. R, 2004, p. 3.

O sistema foi desenvolvido visando o relaxamento através da técnica de neurofeedback. Através deste treinamento, o paciente pode aprender a controlar alguns padrões biológicos (auto-regulação), e dessa forma aprender a controlar ou minimizar diversos efeitos gerados pelo estresse diário. Este dispositivo também pode ser aplicado na reabilitação cognitiva de pessoas que sofreram algum tipo de Acidente Vascular Cerebral (AVC), na redução da tensão pré – menstrual, epilepsia etc.

O principal problema das aplicações expostas anteriormente é que foram criadas para se conectarem somente com um determinado banco de dados. Essa realidade de coexistência de bancos de dados autônomos provoca o surgimento de informações inconsistentes que, na maioria das vezes, não podem ser integradas. A identificação dessas divergências e inconsistências que existem em diferentes bancos de dados é fundamental para a integridade dos dados e veracidade das informações manipuladas. A integração dos bancos de dados tem como ponto importante a obtenção de um esquema global, em que ocorre a integração dos dados e os conflitos semânticos podem ser resolvidos.

Para solucionar esse problema foi proposta uma nova arquitetura que será descrita nos capítulos a seguir.

CAPÍTULO 4

TÉCNICAS E FERRAMENTAS PARA DESENVOLVIMENTO DE APLICATIVOS PARA OPERAÇÃO E MANUSEIO DE BANCOS DE DADOS

Um dos objetivos da construção de um *framework* como o que se propõe é criar um sistema com classes reutilizáveis e, usualmente, extensíveis. Diversas técnicas podem ser usadas na construção de sistemas com essas características; como por exemplo, a divisão em camadas. Cada camada agrupa classes que cooperam para a realização de uma mesma responsabilidade de alto nível. Nesse modelo, a comunicação entre as camadas deve ser minimizada, sendo realizada por meio de interfaces bem definidas. Outras técnicas de grande utilidade envolvem recursos como a reflexão e arquiteturas baseadas em *plug-ins*. Neste capítulo, iremos nos ater às técnicas que serão utilizadas no desenvolvimento do *framework* para interface com banco de dados, conforme proposto neste trabalho.

4.1 Framework

São encontradas na literatura várias definições para o eterno *framework*. De acordo com Pree (1995) e Pinto (2000), um *framework* é um software parcialmente completo,

projetado para ser instanciado. Assim, um *framework* define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. *Framework* pode ser classificado, segundo Fayad et al (2000), em *framework* de aplicação, orientado a objetos ou, segundo Szyperski (1997), em *framework* de componentes. O primeiro define uma estrutura para o desenvolvimento de aplicações orientadas a objetos, enquanto que o segundo define uma infraestrutura de execuções onde componentes podem ser conectados.

O principal benefício do desenvolvimento baseado em *frameworks* é a redução do esforço para entender e manter a aplicação (BARRETO JUNIOR e FUKS, 2006).

Framework pode também ser definido sob os aspectos de estrutura e função. Quanto à estrutura, um *framework* consiste em um diagrama de classes e um diagrama de eventos. Quanto à função, um *framework* é o esqueleto de uma aplicação e que pode ser personalizado por um desenvolvedor de aplicação, ou seja, a função do *framework* é extrair e disponibilizar a essência de uma determinada família de aplicações, de modo que o desenvolvedor possa partir de um patamar superior de implementação da arquitetura. Nesse sentido, um *framework* tem a mesma função que uma API (*Application Programming Interface*) (SILVA, 2006).

4.1.1 Framework de componentes

O conceito de *framework* de componentes (*Component Framework* – CF) tem sido aplicado, extensivamente, como estratégia para determinar de que modo componentes, em uma composição, podem se associar e interagir. Um *framework* de componentes é uma entidade de software que provê suporte a componentes que seguem um determinado modelo e possibilita que instâncias desses componentes sejam conectadas (SZYPERSKI, 2002; GOMES e SOARES, 2005).

Um *framework* de componentes contém um conjunto de regras que deve ser obedecido pelos componentes em certo ambiente. Essas regras formam a base em que múltiplos componentes podem coexistir em um único ambiente. O *framework*, sozinho, não implementa, necessariamente, nenhuma funcionalidade. Entretanto, ele deve gerenciar o compartilhamento de recursos e prover um canal de comunicação entre os componentes existentes. Além disso, o *framework* de componentes deve gerenciar os recursos compartilhados pelos componentes e prover mecanismos de suporte que habilitam a comunicação (PRATI; GEROMINI e MONARD, 2003).

Um sistema que permite a introdução de novos componentes, quando necessário, é chamado extensível, ou seja, ele pode lidar com a adição de extensões sem requerer uma checagem global de integridade. Além disso, com a existência de um *framework* de componentes, a construção de novos componentes pode ser projetada tendo em mente a interoperabilidade. Nesse caso, pontes entre os diferentes modelos podem ser desenvolvidas e rapidamente adaptadas para o uso, dando ao sistema uma avançada extensibilidade. Outras funções dos *frameworks* para a integração de componentes são: modularidade, reusabilidade e inversão de controle. A reusabilidade é aumentada, pois as interfaces definidas pelo *framework* definem componentes que podem ser reutilizados e combinados para criar novos componentes. A arquitetura definida pelo *framework* possibilita um processamento padronizado do gerenciamento de eventos que são invocados por meio de um mesmo mecanismo de envio, possibilitando uma inversão de controle (PRATI; GEROMINI e MONARD, 2003).

Assim, o *framework* estabelece as condições necessárias para um componente ser executado e regula a interação entre as instâncias desses componentes. Um *framework* de componentes pode ser único na aplicação, criando uma ilha de componentes ao seu redor, ou pode cooperar com outros componentes (SZYPERSKI, 2002).

De forma geral, um *framework* de componentes permite:

- construir aplicações que possam ser atualizadas ou estendidas com facilidade e rapidez (SZYPERSKI, 2002);
- criar novas partes de um sistema e fazer a publicação delas sem a necessidade de recompilação (SZYPERSKI, 2002);
- desenhar uma aplicação de forma que as peças que a compõem possam ser plugadas em tempo de execução (SZYPERSKI, 2002).

Conforme se observa, um *framework* de componentes pode contribuir para o aumento da qualidade e da produtividade no desenvolvimento de aplicativos. Além disso pode oferecer um esquema de implementação a partir do qual suas classes podem ser estendidas ou instanciadas para desenvolver aplicações específicas. O ambiente criado para dar suporte aos componentes e que funciona como elemento integrador deve ser projetado de forma a permitir a substituição e evolução dos componentes. Para conseguir tal flexibilidade, deve-se investir na arquitetura e no projeto inicial do sistema como um todo, com ênfase no reuso e na flexibilidade individual do componente e na flexibilidade do ambiente de suporte (COSTA, 2008).

4.2 Modelo multicamadas

Seguindo o processo natural da evolução dos *softwares*, temos hoje as chamadas aplicações em camadas. No início da era da computação, as aplicações restringiam-se a unidades monolíticas (*mainframes*) que continham as aplicações e os dados, centralizando todo o processamento. Com o advento da microinformática (terminais inteligentes) e a evolução das redes, o processamento passou a ser dividido, sendo que a lógica da aplicação

ora ficava na interface com o cliente, ora no gerenciador de banco de dados (quando eles eram usados), ou até mesmo se dividia entre eles (AZEVEDO, 2008b).

O uso de camadas é um padrão arquitetural que ajuda na tarefa de separar responsabilidades, promovendo baixo acoplamento e alta coesão em um sistema. Partindo desse princípio, ele conceitua camadas como “divisões lógicas de componentes agrupados por responsabilidades em comum”. Idealmente, camadas se comunicam apenas com camadas adjacentes (CALÇADO, 2005).

Para entender melhor como funciona o modelo multicamadas, pode-se considerar o seguinte exemplo: a Figura 4.1 mostra uma aplicação construída em três camadas e os conceitos envolvidos na tecnologia de multicamadas, bem como os elementos necessários para a composição de um ambiente baseado nesse método e suas características:

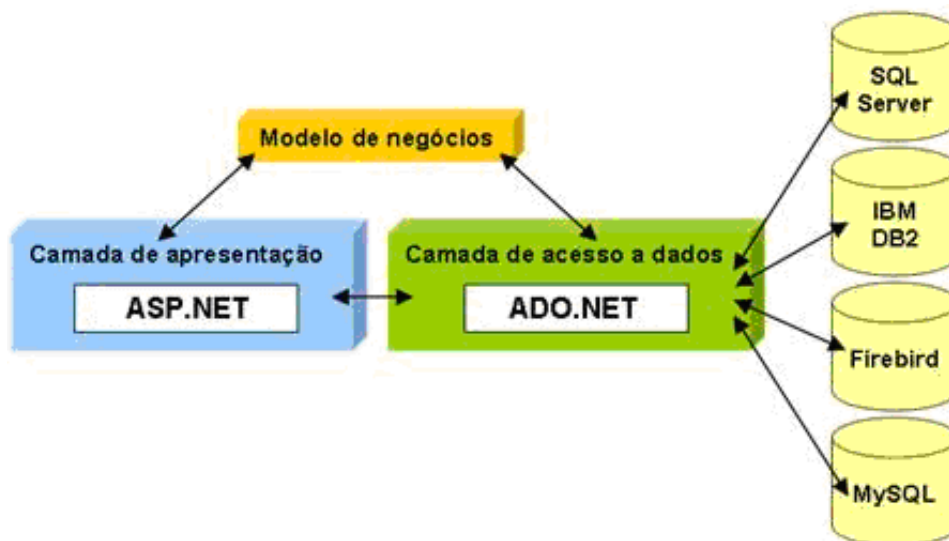


Figura 4.1: Apresentação do modelo multicamadas. Fonte: CARVALHO, R. M, 2008.

- A camada de apresentação é responsável por apresentar o resultado do processamento. Essa camada deve ser o menos dependente possível das demais camadas, de forma que a manutenção não exija o conhecimento da lógica de negócio nem da camada de dados (MACORATTI, 2008b).

- A camada de lógica de negócio é responsável pelas regras e processos de negócio da aplicação (MACORATTI, 2008b).
- A camada de dados representa os dados da aplicação. Ela pode ser dividida em: lógica da persistência e dados propriamente ditos (MACORATTI, 2008b).

4.2.1 Camada de Apresentação

É a chamada de *GUI (Graphical User Interface)*, ou simplesmente interface. Essa camada interage diretamente com o usuário. É por meio dela que são feitas as requisições como consultas, por exemplo. A camada de apresentação é uma parte essencial de um aplicativo, de forma que, se for projetada de forma inadequada, pode resultar em complexidade, falta de flexibilidade, ineficiência e experiência frustrante para o usuário.

Muitos fatores influenciam na decisão sobre que abordagem é a melhor para um aplicativo em particular (requisitos da plataforma cliente, implantação e atualização de aplicativos, experiência do usuário, desempenho, integração do cliente etc). Por exemplo, é possível fornecer uma interface do usuário, avançada, com uma camada de apresentação baseada em navegador. Da mesma forma, é possível fornecer uma interface do usuário, totalmente dinâmica. Nenhuma seria muito fácil de fazer e ambas, provavelmente, resultariam em complexidade desnecessária, falta de flexibilidade e altos custos de desenvolvimento e manutenção (HILL, 2004).

4.2.2 Camada da Lógica de Negócios

Essa camada, também conhecida por *BLL (Business Logic Layer)* contém as funções e regras de todo o negócio.

Seus dados são voláteis, ou seja, para que algum dado seja mantido, deve ser utilizada a camada de dados. Não existe uma interface para o usuário (MSDN, 2008).

O objetivo da camada de negócios é implementar a lógica da aplicação, expondo essa lógica para a camada de apresentação ou para outras aplicações (BARRETO JUNIOR e FUKS, 2006).

Nas aplicações, o uso de persistência possibilita que o desenvolvedor concentre-se na lógica de negócio em vez de concentrar-se na tecnologia em que persistem os dados. Além do mais, o código da aplicação fica independente do SGBD (sistema de gerenciamento de banco de dados), pois todos os comandos para consultar, salvar, atualizar e remover entidades são implementados na lógica de negócios (BARRETO JUNIOR e FUKS, 2006).

No modelo de implementação em camadas, a camada de regras de negócios tem papel fundamental em qualquer sistema que segue essa proposta (AZEVEDO, 2008b).

4.2.3 Camada de Acesso a Dados

A Camada de Acesso aos Dados (*Data Access Layer – DAL*) é a camada responsável pelo acesso aos dados, mantém todo o conhecimento quanto aos comandos *SQL* e acesso ao banco de dados. Essa camada deve ser bem estruturada, de forma a permitir o máximo de transparência quanto ao banco de dados utilizado, seja *SQL Server*, *Oracle*, *Access*, *Firebird* etc. A *DAL* encapsula todo o processo de criação de objetos de conexão, objetos que representam comandos, parâmetros a serem utilizados nos comandos, objetos de transação etc. Além de prover métodos para acessar uma ou mais tabelas relacionadas em um único banco de dados, ou em instâncias de bancos, ou em múltiplos bancos de dados (SERRATO, 2005).

Uma das metas desses componentes é encapsular o processo de chamada, provendo serviços de acesso aos dados para as aplicações. Especificamente, Serrato (2005) explica que os componentes *DAL* implementam os seguintes detalhes:

- gerência e encapsulamento de esquemas;
- gerência de segurança e autorização de maneira apropriada;
- gerência de transações;
- paginação de dados;
- roteamento de dependência de dados, se necessário;
- implementação de estratégia de cache se necessário, para *queries* de dados não transacionais;
- efetivação de serialização de dados.

De acordo com Serrato (2005), o componente de lógica de acesso a dados (*DAL* – *Data Access Layer*) deve prover métodos para executar as seguintes tarefas contra o banco de dados:

- criar registros no banco de dados;
- ler registros no banco de dados e retornar os dados da entidade de negócio ao objeto chamador;
- atualizar registros no banco de dados, utilizando informações de dados da entidade de negócio, já revisados, enviados pelo objeto chamador;
- exclusão de registros no banco de dados.

Além das funções básicas de criação, leitura, atualização e exclusão de registros, esses componentes implementam a lógica de negócio (SERRATO, 2005).

Essa camada recebe as requisições da camada de negócios e seus métodos executam essas requisições em um banco de dados. Assim, alterando-se o banco de dados, estão se alterando apenas as classes da camada de dados, e o restante das camadas não seria afetado por essa alteração. Essa separação facilita a manutenção e a portabilidade da aplicação (MACORATTI, 2008b).

A camada de apresentação não acessa dados, nem a camada de dados faz apresentação. Cada uma cumpre o seu papel. A camada de apresentação não deverá conter qualquer referência a código de acesso a dados, fazendo somente chamadas a *DAL* para qualquer requisição de dados (MACORATTI, 2008b).

4.2.4 Vantagens e desvantagens da utilização do Modelo Multicamadas

A arquitetura multicamadas envolve a separação das funcionalidades, usando camadas, com o objetivo de separar a lógica de apresentação, a lógica de negócio e a conexão com o banco de dados (lógica de acesso a dados).

A separação em camadas torna o sistema mais flexível, de modo que partes podem ser alteradas independentemente. Com o emprego dessa arquitetura, qualquer alteração em uma determinada camada não influi nas demais, desde que o mecanismo de comunicação entre elas permaneça inalterado. Isso permite substituir uma camada inteira por outra (CALÇADO, 2005).

Calçado (2005) enumera algumas vantagens e desvantagens em desenvolver sistemas, utilizando uma arquitetura em camadas:

Vantagens:

- ✓ reduzem complexidade: agrupam componentes e simplificam a comunicação entre eles;

- ✓ reduzem dependência/acoplamento: a regra de comunicação evita dependências diretas entre componentes de camadas diferentes;
- ✓ favorecem a coesão: componentes de responsabilidades relacionadas são agrupados;
- ✓ promovem reusabilidade: camadas podem ser reutilizadas em outros sistemas ou podem ser substituídas;
- ✓ é um padrão arquitetural conhecido: facilita a comunicação e entendimento entre desenvolvedores.

Desvantagens:

- ✓ limitadas pela tecnologia: algumas regras precisam de ser quebradas por limitações tecnológicas;
- ✓ apenas complicam um sistema muito simples: não é qualquer sistema que exige o uso de camadas;
- ✓ possibilidade de overdose: muitos arquitetos acabam criando camadas demais e tornando a aplicação extremamente complexa.

4.3 DB Provider Factory

Os novos conceitos de *provider factory*, contidos na classe *Data.Common.DBProvider.Factories* estão entre as novidades da plataforma .NET (uma plataforma única para desenvolvimento e execução de sistemas e aplicações), versão 2.0. Anteriormente, quando se pretendia efetuar uma conexão com uma base de dados, os objetos eram criados diretamente. Por exemplo, para acessar o Microsoft SQL Server, eram utilizadas as classes *SqlConnection*, *SqlDataAdapter*, *SqlCommand*. Para acessar uma base de dados *Microsoft*

Access, eram utilizados os provedores *OleDbConnection*, *OleDbDataAdapter*, *OleDbCommand* etc. Dessa forma, uma série de classes específicas de provedores era requisitada de acordo com a fonte de dados a ser acessada (MACORATTI, 2008a).

A classe de *provider factory* é derivada de uma classe base abstrata: *System.Data.Common.DbProvider.Factory* e para cada tipo de base de dados a ser acessado tem-se uma nova classe derivada dessa classe base abstrata (MACORATTI, 2008a).

Uma classe base de provedor é um objeto *factory*, que é usado para criar um conjunto de objetos relacionados como *SqlConnection* e *SqlCommand*. Eles retornam um tipo de classe abstrata do tipo *DBConnection* (MACORATTI, 2008a).

Essa classe *DBProviderFactory* permite a realização de duas tarefas:

- obter uma lista de todos os provedores existentes via método estático *GetFactoryClass*;
- criar uma instância de um determinado *Factory*, conforme o seu tipo via método *GetFactoryClass*.

Para determinar todas as classes *DbProviderFactory* disponíveis e criar suas classes adicionais, é fornecida a classe *System.Data.Common.DbProviderFactory* (MACORATTI, 2008a).

Cada provedor de dados faz o registro de uma classe *ProviderFactory* no arquivo *machine.config* da plataforma .NET. A classe base *DbProviderFactory* e a classe *ProviderFactories* podem retornar um *DataTable* de informações sobre os diferentes provedores registrados no arquivo *machine.config* e podem recuperar a *ProviderFactory* conforme a sequência do provedor fornecida ou um *DataRow* de um *DataTable* (MACORATTI, 2008a).

A Figura 4.2 mostra a classe *DBProviderfactory* e suas derivações para cada tipo de base de dados, usando código genérico. As classes concretas, como por exemplo o *SqlCommand*, herdam diretamente da classe *DbCommand*, logo, já conseguem ter o acesso genérico, pois os métodos sempre retornam instâncias dos objetos concretos.

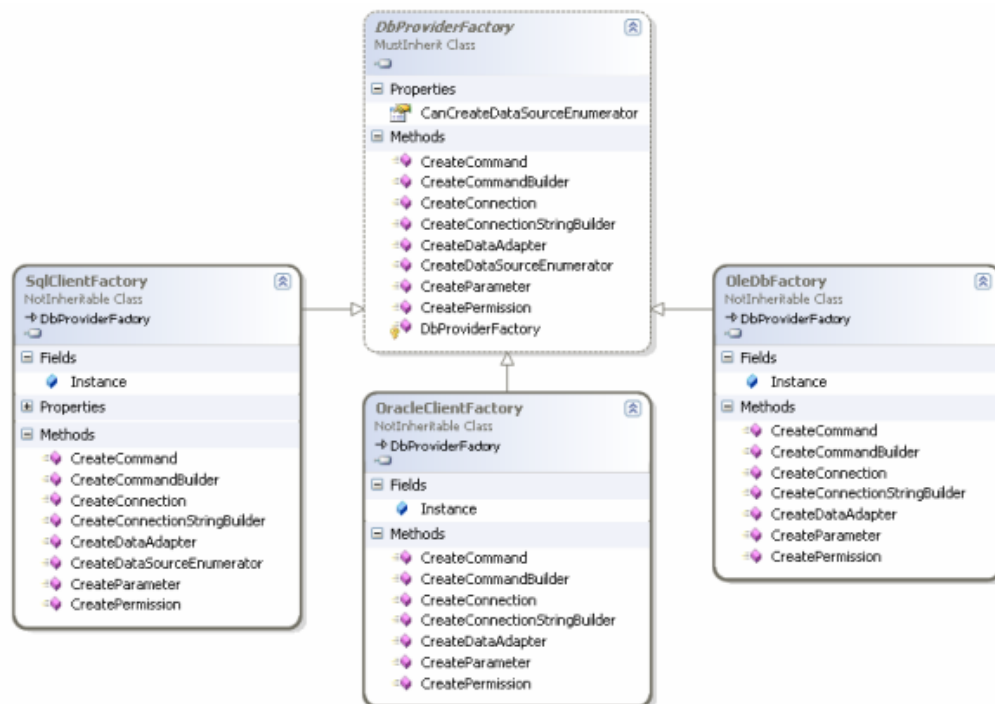


Figura 4.2: Herança da classe *DbProviderFactory*. Fonte: AECE, I, 2005.

A classe *DbDataProvider* define o número de funções que a subclasse precisa de implementar, conforme descrito abaixo:

- *createCommand()*: cria um objeto *Command* derivado de *DbCommand*;
- *createCommandBuilder()*: cria um objeto derivado de *DbCommandBuilder*;
- *createConnection()*: cria um objeto derivado de *DbConnection*;
- *createConnectionStringBuilder()*: cria um objeto derivado de *DbConnectionStringBuilder*;
- *createDataAdapter()*: cria um objeto derivado de *DbDataAdapter*;

- *createDataSourceEnumerator()*: cria um objeto derivado de *DbDataSourceEnumerator*;
- *createParameter()*: cria um objeto derivado de *DbParameter*;
- *createPermission()*: cria um objeto derivado de *CodeAccessPermission*.

A Figura 4.3 exemplifica uma interface genérica, usando também o código genérico (*IDbConnection*, *IDbDataAdapter*, *IDbCommand* etc), implementada pela classe abstrata *System.Data.Common.DbProviderFactory* (MACORATTI, 2008a).

A classe abstrata *DbProviderFactory*, do namespace *System.Data.Common*, representa uma “fábrica” com um conjunto de métodos virtuais, para criar instâncias de classes de uma implementação de fonte de dados específica. Cada provedor de dados projetado para o *.NET Framework* deve fornecer uma especialização da classe abstrata *DbProviderFactory*.

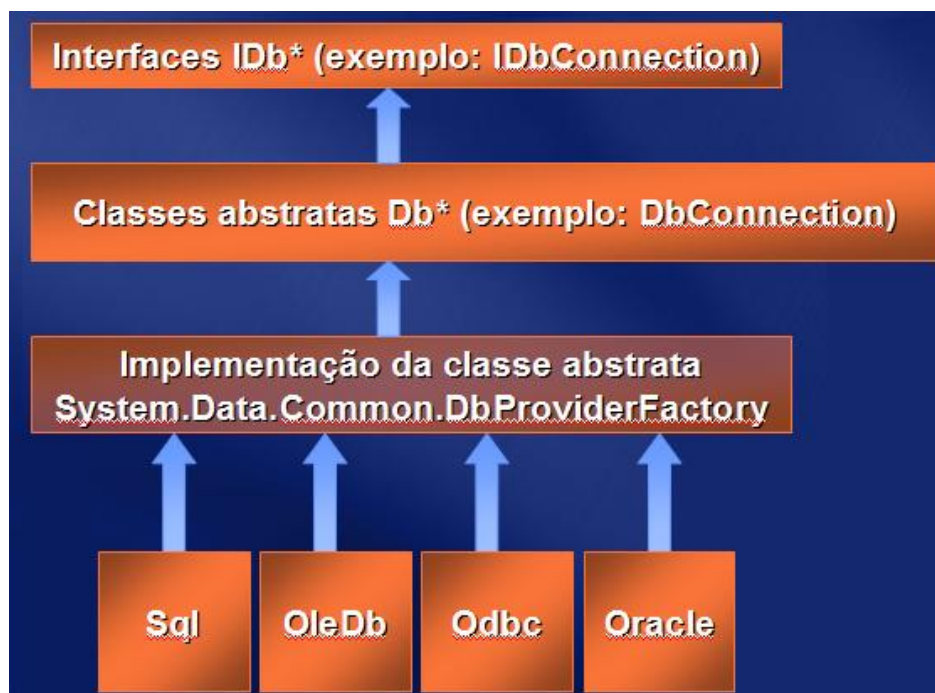


Figura 4.3: Esquema simplificado de classes para acesso a dados no ADO.NET 2.0. Fonte: CARVALHO, R. M, 2008.

De um modo geral, interfaces, nomeadas como IDb (*IDbConnection*, *IDbCommand*, *IDbDataAdapter* etc), do *namespace System.Data*, são implementadas por classes abstratas, nomeadas como Db (*DbConnection*, *DbCommand*, *DbDataAdapter* etc), do *namespace System.Data.Common*. As classes abstratas Db implementam alguns métodos das interfaces correspondentes IDb e passam a responsabilidade de implementação de outros métodos para classes concretas, desenvolvidas em provedores de dados para o .NET *framework*. Conforme a descrição do padrão de projeto *abstract factory*, as classes abstratas declaram uma interface para um tipo de objeto-produto.

4.4 Reflexão

Reflexão ou *Reflection* é o processo de obter informações sobre *assemblies* (um conjunto, é uma coleção de tipos e recursos que são criados para trabalhar juntos e formar uma unidade lógica de funcionalidade) e os tipos definidos dentro dele, e ainda, a criação, invocação e acesso às instâncias de tipos em tempo de execução (FERNANDES, 2003).

A reflexão é, cada vez mais, utilizada em diversos projetos, incluindo várias partes da própria plataforma .NET. Seus recursos facilitam a criação de sistemas adaptativos, permitindo ao programador criar sistemas genéricos, cujas funcionalidades só serão conhecidas e acessadas em tempo de execução (MICROSOFT, 2007).

A riqueza de informações disponíveis sobre os objetos (tipos) em execução, poderá ser navegada e investigada por meio da reflexão disponível através das classes do *namespace* (são usados como um sistema organizacional. Eles fornecem uma maneira de classificar e apresentar os elementos de programação que são expostos a outros programas e aplicativos) *System.Reflection* (MICROSOFT, 2007).

Por meio da reflexão, verifica-se que é possível obter informações sobre os métodos de uma classe e executá-los dinamicamente (MICROSOFT, 2007).

A reflexão também permite inferir sobre o estado (introspecção) e modificar a semântica do sistema em tempo de execução (intervenção). A introspecção permite ao sistema descobrir quais as classes que compõem o código, os atributos e métodos contidos nas mesmas, em plataformas orientadas a objeto. A intervenção permite criar instâncias dessas classes e executar seus métodos (DEMERS e MALENFANT, 1995).

Com a criação de novas classes e métodos, a reflexão permite realizar modificações nas classes existentes, em tempo de execução. Além disso, aquela pode ser implementada em diversas linguagens de programação, como nas plataformas de programação orientada a objeto, Java e .NET (DEMERS e MALENFANT, 1995 e LIMA, 2006).

A reflexão, como técnica de programação, consiste em obter informações dinâmicas sobre objetos: classe, superclasse, campos, seus tipos e valores, métodos, construtores etc. Essas informações são usadas para fazer transformações durante a execução do programa (DEVEGILI, 2000).

Reflexão permite inspecionar, programaticamente, uma montagem e obter informações sobre ela, assim como seus tipos e atributos. É possível que cada um crie as próprias montagens e tipos, utilizando os serviços disponíveis na classe *System.Reflection.Emit* (LIMA, 2006).

O pacote *System.Reflection.Emit* contém classes que permitem criar novos elementos da plataforma e alterar os existentes em tempo de execução. Para isso, existe uma série de classes “*Builder*” que realizam essa função (MICROSOFT, 2007).

A reflexão também permite a inclusão de marcadores especiais em elementos da linguagem (classes, atributos, métodos, parâmetros, *assemblies* etc). Esses marcadores são chamados de atributos e são inseridos na forma de metadados que não influem, diretamente,

no membro ao qual foram associados. Os atributos podem ser obtidos por meio da reflexão, fornecendo informações especiais sobre um membro (DEMERS; MALENFANT, 1995 e MAES, 1987).

A reflexão tem duas categorias: inspeção e manipulação. A inspeção envolve a análise de objetos e de tipos para a obtenção de informações estruturadas sobre a sua definição e o seu comportamento. Normalmente, isso é feito com pouco ou nenhum conhecimento anterior sobre eles, além de algumas provisões básicas (por exemplo, no *.NET Framework*, todos herdam de *System.Object* e uma referência com tipo de objeto será o ponto inicial típico para a reflexão). A manipulação utiliza as informações obtidas por meio da inspeção para chamar o código dinamicamente, criar novas instâncias de tipos descobertos e, até mesmo, reestruturar tipos e objetos rapidamente. É importante observar que, para a maioria dos sistemas, a manipulação de tipos e de objetos em tempo de execução incorrerá em uma penalidade de desempenho quando comparada às operações equivalentes feitas de forma estática no código-fonte (MICROSOFT, 2007).

Essa é uma poderosa ferramenta para o desenvolvimento de solução muito genérica (MICROSOFT, 2007).

Segundo Devegili (2000), uma linguagem ou ambiente de programação que possua uma arquitetura reflexiva deve garantir que:

- as aplicações tenham acesso aos dados que representam aspectos sobre si mesmas (autorrepresentação);
- qualquer mudança nesses dados seja causada por mudanças nos aspectos sobre o sistema, e que mudanças nos aspectos sobre o sistemas sejam refletidas nos dados que os representam.

Os pacotes da plataforma por meio dos quais os recursos de reflexão são acessados e algumas das classes contidas nesses pacotes são apresentados na Figura 4.4.

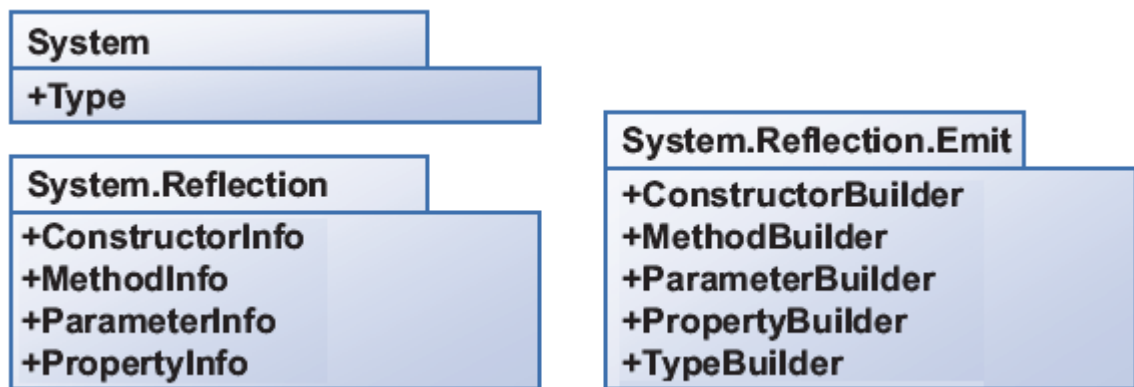


Figura 4.4: Pacotes da plataforma .net utilizados na reflexão com as classes disponíveis nesse pacote.

Funcionalidades encontradas na *Class.System.Reflection*:

- ***constructorInfo ()*** – recebe uma coleção contendo todos os construtores encontrados na classe na qual se realiza a pesquisa;
- ***methodInfo ()*** – recebe informações sobre os métodos contidos na classe;
- ***parameterInfo ()*** – recebe uma coleção contendo os parâmetros encontrados em cada construtor que foi localizado na classe que pesquisamos;
- ***propertyInfo ()*** – recebe informações sobre as propriedades na classe;
- ***fieldInfo ()*** – recebe informação sobre os campos contidos na classe.

A principal classe utilizada para acesso aos recursos de reflexão da plataforma .NET é a classe *Type* do pacote *System*. Trata-se de uma classe base abstrata, utilizada apenas como interface para suas classes derivadas. Assim, não é possível gerar uma instância diretamente da classe *Type*, mas a plataforma fornece meios de obter referências para instâncias das subclasses *Type* (DEMERS e MALENFANT, 1995).

Type possui diversas propriedades que permitem acessar as informações relativas a uma determinada classe. Por meio dessas propriedades, é possível, por exemplo, descobrir o

nome da classe, de qual classe ela herda e se a classe é abstrata (DEMERS e MALENFANT, 1995).

A maioria dos métodos de *Type* é utilizada para obter informações sobre os membros da classe (construtores, atributos, métodos etc). Existem vários métodos com diversas versões com essa função, utilizados para obter todos os membros da classe de um determinado tipo. Como exemplos, podem ser citados o *GetConstructor* (para obter um objeto do tipo *ConstructorInfo* que contém informações sobre um dos construtores da classe) e o *GetProperty* (retorna um objeto do tipo *PropertyInfo* com informações sobre uma das propriedades do objeto) (DEMERS e MALENFANT, 1995).

O pacote *System.Reflection* contém uma classe chamada *Assembly*, utilizada para manipular os arquivos executáveis e bibliotecas da plataforma. Essa classe possui métodos que permitem ao programador carregar, ou até mesmo executar um *assembly*, caso este seja um executável. Assim como a classe *Type*, essa classe possui uma série de propriedades e métodos que permitem ao programa descobrir informações sobre um *assembly* (DEMERS; MALENFANT, 1995 e MAES, 1987).

Antes de realizar qualquer operação com uma instância da classe *Assembly*, é preciso carregar a informação de um arquivo *assembly*. Isso pode ser feito por meio dos métodos estáticos *Load* e *LoadFrom*, que criam um novo objeto da classe e carregam a informação do *assembly*. Caso se deseje obter uma referência para um objeto que represente o *assembly* do programa que está sendo executado, basta utilizar o método *GetExecutingAssembly*. Após carregar um *assembly*, é possível utilizar o método *GetTypes* para obter um vetor de objetos *Type* referentes às classes disponíveis no *assembly*. A partir desse momento, é possível gerar uma instância de uma dessas classes por meio dos métodos descritos anteriormente (DEMERS; MALENFANT, 1995 e MAES, 1987).

Na plataforma *.NET*, quando um programa é compilado (gerando um código de máquina na forma de um arquivo *.dll* ou *.exe*), o compilador inclui, nesse código, os metadados com informações sobre as classes. A plataforma fornece classes que são capazes de acessar esses metadados, permitindo instalar um aplicativo de cliente, avançado, sem destruir outros aplicativos. O *.NET* permite que os aplicativos sejam isolados, mantendo todos os artefatos de aplicativos em um diretório local, de modo que todos os conjuntos de módulos (*assemblies*) sejam mantidos separados. Além disso, tais aplicativos não exigem nenhum processo de registro quando são implantados, reduzindo ainda mais o risco de destruir outros aplicativos. O *.NET framework* permite que várias versões de um conjunto de módulos (*assembly*) sejam implantadas lado a lado. Isso garante que, quando um aplicativo é executado, faça-o com as versões exatas dos conjuntos de módulos (*assemblies*) com os quais foi desenvolvido e testado (HILL, 2004).

Montagem ou *assembly* é um programa feito em *.NET*, exibido como uma unidade única; como por exemplo uma classe, uma *dll*, ou um executável (LIMA, 2006).

4.5 Plug-in

Segundo CEMBRANELLI (2008), *plug-in* é o conjunto de serviços para controlar um grande conjunto de componentes trabalhando conjuntamente.

Plug-ins vêm sendo reconhecidos como um dos aspectos mais relevantes para a determinação da qualidade do software. Servem, normalmente, para adicionar funções a outros programas maiores, provendo alguma funcionalidade especial ou muito específica (CEMBRANELLI, 2008).

O *plug-in* é uma biblioteca para a construção de aplicativos. Ele fornece funcionalidades a um nível maior de abstração para os usuários desses aplicativos. Assim,

esse usuário pode utilizar as operações disponíveis na biblioteca sem ter conhecimento de como são implementadas, podendo se dedicar à implementação de outras funcionalidades; como por exemplo, visualização dos dados, interface com o usuário e ferramentas para análise (CEMBRANELLI, 2008).

Arquitetura de *plug-in* é a forma de facilitar a inclusão de novos tipos de dados, transformando a arquitetura definida e implementada num *framework*. Além disso, pode-se criar um conjunto de *plug-ins* capazes de satisfazer um grande conjunto de aplicações (CEMBRANELLI, 2008).

Os *plug-ins* são utilizados atualmente em diversas aplicações; como por exemplo, nos navegadores da internet (para exibir animações e vídeos em páginas html), em reprodutores multimídia (para reproduzir novos formatos), e em editores de imagem (para adição de novos efeitos). Vários sistemas, de diferentes áreas, desenvolvidos utilizando arquiteturas de *plug-ins*, podem ser encontrados na literatura (PAPAJORGJI, 2005 e MULLICK et al., 1999).

Para tal, o *Eclipse* foi desenvolvido utilizando um sistema de *plug-ins* bastante eficiente, cuja arquitetura é mostrada na Figura 4.5. A figura mostra que a plataforma é composta apenas de componentes básicos, sendo eles uma bancada (*Workbench*) que utiliza as bibliotecas gráficas *JFace* e *SWT* para gerar os componentes visuais relativos à interface com o usuário, uma área de trabalho (*Workspace*) que gerencia os arquivos e projetos em desenvolvimento, um sistema de ajuda ao usuário e um sistema de colaboração para desenvolvimento em equipe. As demais funcionalidades da *IDE* são adicionadas por meio de *plug-ins* que estendem os recursos básicos da plataforma, adicionando, por exemplo, editores para diversas linguagens com verificação de sintaxe, editores gráficos, navegadores de objetos, integração com servidores de aplicação, dentre outros. Com isso, é possível transformar o *Eclipse* em uma *IDE* Java, em uma ferramenta de desenvolvimento para web,

ou em uma suíte para modelagem em *UML*, modificando apenas o conjunto de *plug-ins* utilizados pelo sistema (RIVIERES e WIEGAND, 2004).

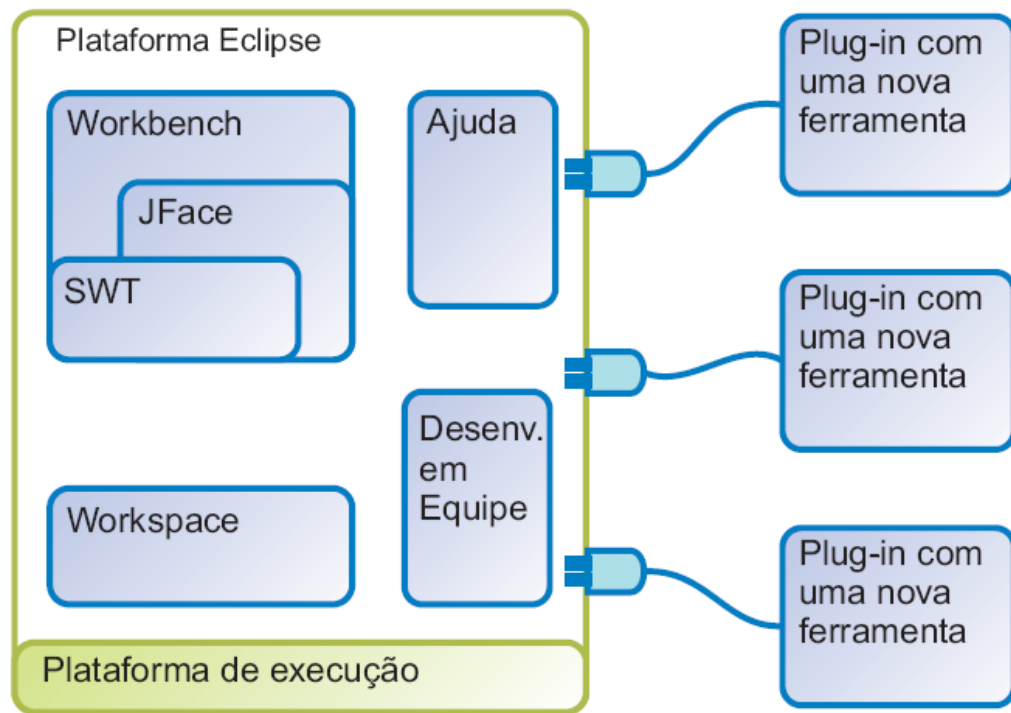


Figura 4.5: Arquitetura da plataforma *Eclipse*.

Neste trabalho, um *Plug-in* será utilizado para a construção do aplicativo e o interfaceamento com os bancos de dados.

CAPÍTULO 5

PROPOSTA DE UM FRAMEWORK PARA INTERFACE E GERENCIAMENTO DE BANCOS DE DADOS

Neste capítulo, as técnicas descritas anteriormente serão utilizadas no desenvolvimento do *framework*, que se caracteriza por utilizar diferentes bancos de dados. Esse *framework* poderá ser utilizado no armazenamento de dados, sinais e imagens para posteriores análises.

O *framework* precisa de ter a capacidade de integrar novos tipos de aplicativos e bancos de dados diferentes. Este *framework* tem por objetivo possibilitar a instanciação de *softwares* (SGBD's) que permitam que um conjunto de sistemas de bancos de dados diferentes seja integrado para consultas e atualizações, de forma transparente à localização dos dados, aos caminhos de acesso e a qualquer heterogeneidade ou redundância que exista.

5.1 Requisitos

O diagrama de blocos de um banco de dados e um aplicativo é apresentado na Figura 5.1. Nele, o aplicativo permite o gerenciamento das tabelas e a execução de funções clássicas,

possibilitando a criação de tabelas, remoção, edição, inserção de conteúdos, entre outras funções dessa linguagem, acessando somente um banco de dados.

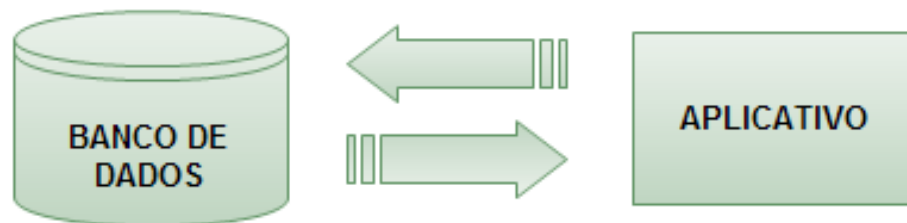


Figura 5.1: Modelo de desenvolvimento tradicional na maioria dos aplicativos.

Diferentes bancos de dados podem utilizar características diferentes. Por exemplo, cada um tem um tipo de arquitetura, podendo diferenciar-se no tipo de sintaxe, no tipo de dados, na conexão, no caminho onde se encontra o banco. Esse número de diferenças entre bancos de dados é o motivo de utilização da arquitetura de *software* proposta na introdução deste trabalho.

Inicialmente, parece mais simples construir um aplicativo para cada instância do banco de dados, em que é possível acessar todos os seus recursos diretamente, como nas arquiteturas clássicas, nas quais cada aplicativo possui um banco de dados específico.

Porém, a arquitetura proposta neste trabalho possibilita o desenvolvimento de aplicativos que, através de um *plug-in*, poderão conectar-se a diversos bancos de dados de forma simples, conforme exemplifica a Figura 5.2. A Figura 5.2 (a) mostra uma variante da arquitetura proposta com um aplicativo conectando com diferentes bancos de dados. Já a Figura 5.2 (b) exemplifica a variante em que aplicativos diversos conectam-se aos bancos de dados diversos.

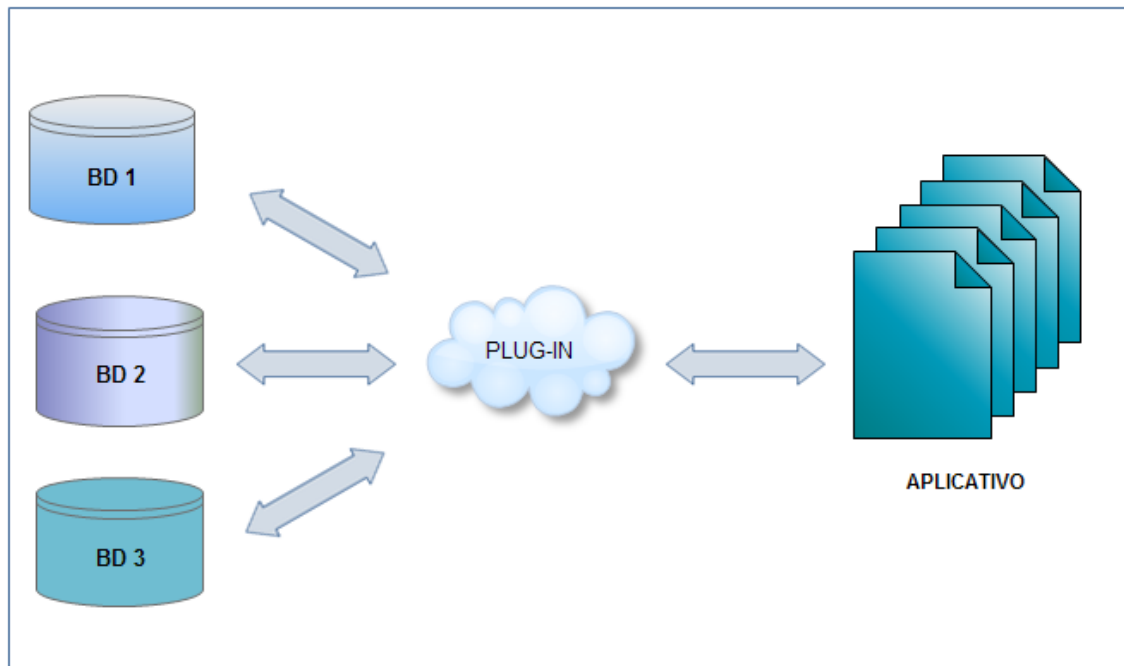


Figura 5.2 (a): Variante da arquitetura proposta, baseada em um aplicativo, um *plug-in* e diversos bancos de dados.

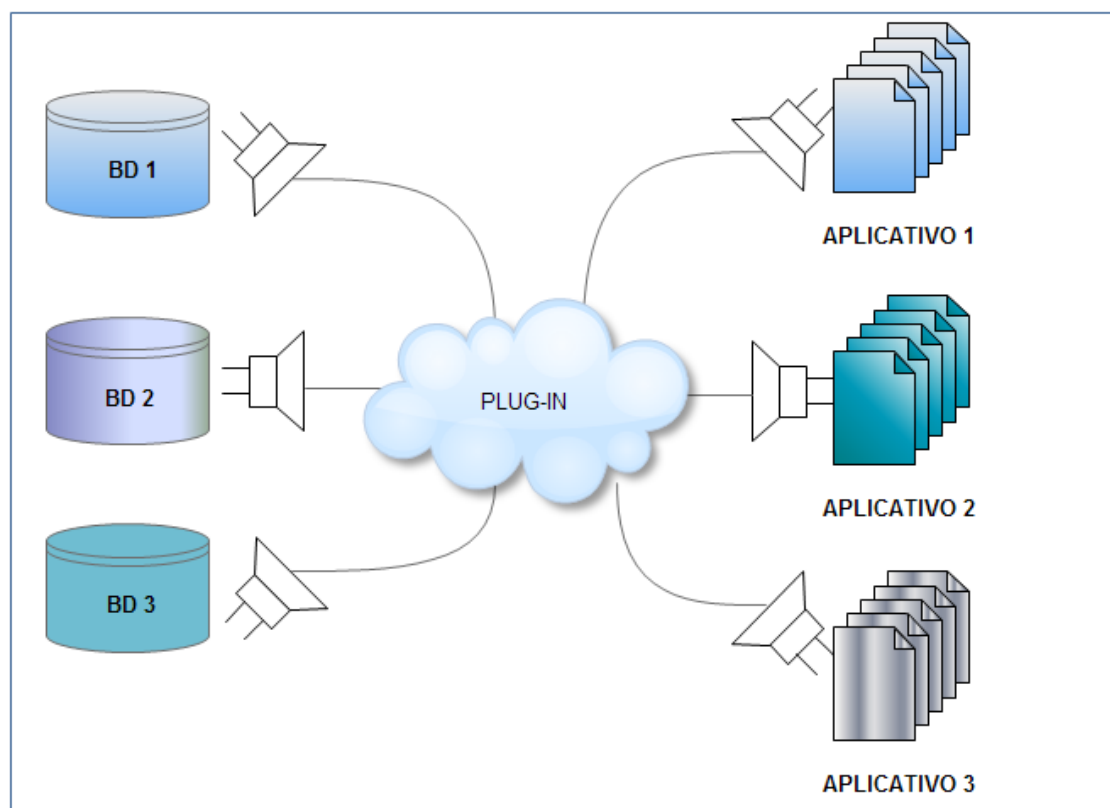


Figura 5.2 (b): Outra variante da arquitetura proposta, baseada em diversos aplicativos, *plug-in* e diversos bancos de dados.

A arquitetura proposta prevê uma interface genérica para diversos bancos de dados. Essa interface é única para todos os bancos de dados compatíveis com o aplicativo. Dessa maneira, o aplicativo torna-se independente do banco de dados utilizado, podendo, durante a sua operação, alternar entre vários bancos de dados diferentes, de maneira transparente para o usuário. Mesmo possuindo uma interface única, ele fornece meios para acessar os recursos específicos de cada banco de dados. Para tal, existe um *plug-in* que funciona como um “tradutor” entre a interface e o banco de dados. Esse *plug-in* deve ser capaz de selecionar qual o tipo de banco de dados a ser utilizado, a partir do conjunto de bancos de dados disponíveis para utilização. O *plug-in* comunica-se com o banco de dados para o qual foi projetado, utilizando-se de métodos para carregá-lo em tempo de execução, através da reflexão.

Para o desenvolvimento do *framework*, foi utilizada uma plataforma de programação que permita suporte a diversas linguagens de programação, pois o modelo *.NET* pode ser exemplificado com o *Visual Studio .NET* (um ambiente integrado que hospeda o *VB .NET*), o *C#* (*C sharp*), e o *C++ .NET*. Subjacente a tudo isso, está a *.NET framework*, conforme descrito por (GOMES; JIN e CREPALDI, 2008).

O processo de desenvolvimento de um aplicativo que utilize o *framework* requer, portanto, que a linguagem utilizada seja compatível com a plataforma *.NET*, e que o *.NET framework* esteja presente no ambiente de execução do aplicativo. As próximas seções apresentam os passos do desenvolvimento do *framework*, utilizando a arquitetura e os requisitos descritos anteriormente.

5.2 Desenvolvimento do *Framework*

O desenvolvimento do *framework* oferece a infraestrutura adequada para o atendimento de tal requisito, já que um *framework* é um (sub) sistema de *software*, de um

domínio particular, que pode ser ajustado para aplicações individuais. Um *framework* consiste em uma grande estrutura, que pode ser reutilizada, como um todo, para a construção de um novo aplicativo; o que representa, no contexto de sistemas de bancos de dados diversos.

Para construir a interface com os bancos de dados, optou-se por utilizar o modelo em camadas, conforme descrito nas Figuras 5.3 e 5.4.

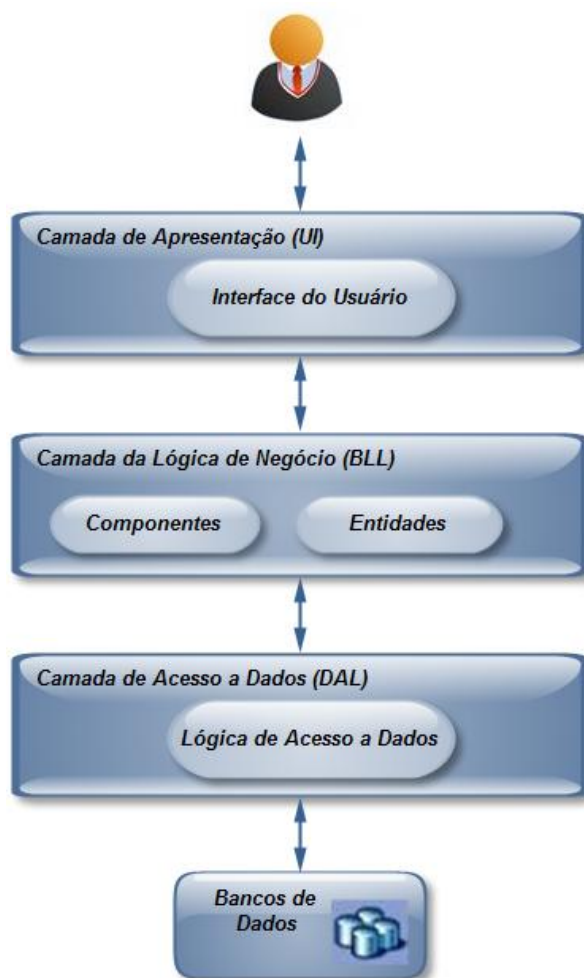


Figura 5.3: Modelo multicamadas da arquitetura proposta.

A camada de acesso a dados deve ser rápida, eficiente e transparente ao desenvolvedor de aplicativos. Essa chave detém a lógica de acesso a dados de todo o *framework*, desde que permaneça sem estado, ou seja, usada por todos os objetos de negócios como ponte para acessar o dado armazenado. A camada de acesso a dados é provisionada independentemente

(SQL Server, Oracle, Access) e isso alavanca muitas características de otimização dos diferentes provedores que garantem uma excelente performance.

Além de utilizar os processos lógicos em camadas, para construir a interface com os diversos bancos de dados, optou-se também por utilizar *plug-in* e reflexão; o que simplifica as tarefas de geração e manutenção da aplicação, conforme exemplificado na Figura 5.4.

A arquitetura proposta é composta pelos seguintes módulos: 1– camada de apresentação; 2– camada de negócios; 3– camada de entidades; 4– camada de acesso a dados; 5– *Plug-in*; 6– Reflexão; 7– *Providers*. A Figura 5.4 mostra, esquematicamente, a estrutura descrita:

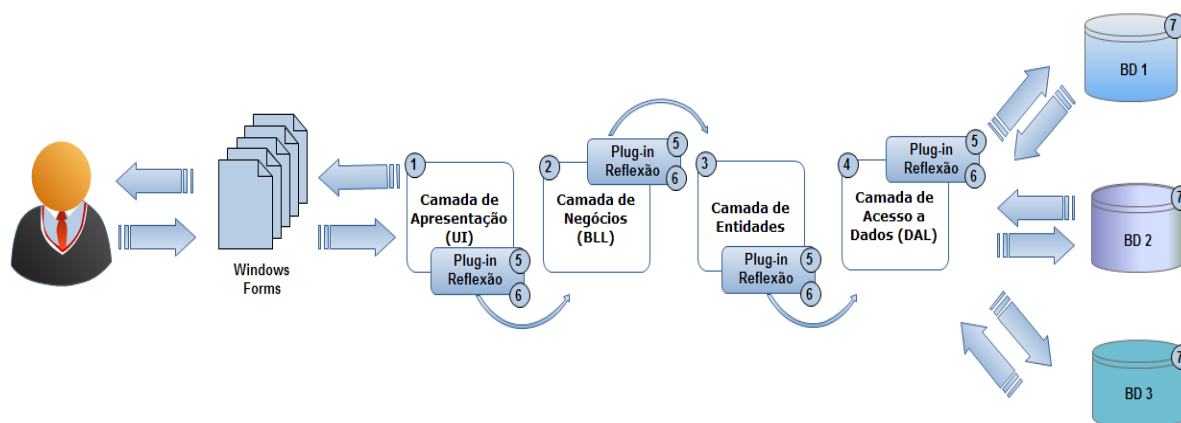


Figura 5.4: Diagrama de blocos da arquitetura proposta.

1. **Camada de Apresentação:** implementada utilizando Winforms, é responsável por fazer a interação do usuário com o sistema. É uma camada que executa os tratamentos de telas e campos e geralmente acessa somente a segunda camada, a qual faz as requisições ao banco de dados e devolve o resultado.
2. **Camada de Lógica de Negócios ou BLL:** classes que encapsulam a lógica de negócios da aplicação. Em um sistema, seguindo esse modelo, a aplicação cliente nunca acessa diretamente a última camada que é a do banco de dados,

pois quem tem essa função é a camada de lógica de negócios, na qual podem se conectar diversas aplicações. Essa parte do sistema é responsável por fazer as requisições ao banco de dados e todo o seu tratamento, ou seja, somente ela tem acesso direto ao banco de dados.

3. **Camada de Entidades:** é basicamente um mapeamento das tabelas do banco. Para cada tabela do banco, teremos uma classe na camada de entidades, utilizada como parâmetro nos comandos enviados para a camada de acesso a dados.
4. **Camada de Acesso a Dados ou DAL:** fornece um mecanismo de persistência para os bancos de dados. É a última divisão do modelo, na qual fica localizado o sistema gerenciador de banco de dados. É também conhecida como camada de dados.
5. **Plug-in:** funciona como um “tradutor” entre a interface e o banco de dados, capaz de selecionar o tipo de banco de dados desejado no sistema, a partir de um conjunto de bancos de dados disponíveis para utilização. O plug-in comunica-se com o banco de dados para o qual foi projetado, utilizando-se de métodos para carregá-lo em tempo de execução. Os plug-ins são, na verdade, bibliotecas representadas por arquivos .dll, que devem estar disponíveis em um diretório especificado pela aplicação. O sistema utiliza essas informações para gerar uma instância da classe, fazendo uso dos recursos de reflexão. O plug-in é mais bem desenvolvido com uma linguagem de programação que suporte reflexão, uma vez que esse recurso permite escolher e instanciar objetos em

tempo de execução. Ou seja, a reflexão examinará os tipos em um assembly, interagindo com eles, instanciando e criando tipos, no momento de execução. Utilizando-se esse recurso, é possível acessar as funções individuais de cada banco de dados implementado. Com isso, o sistema examinará o conteúdo do assembly, encontrará tipos associados com os métodos, campos, propriedades, eventos etc, relacionados em todas as funções específicas, bem como as assinaturas de cada um dos métodos.

6. **Reflexão:** é o processo de obter informações sobre *assemblies* e os tipos definidos dentro dele, e ainda, a criação, invocação e acesso às instâncias de tipos em tempo de execução. A reflexão tem duas categorias: inspeção e manipulação. A inspeção envolve a análise de objetos e de tipos para a obtenção de informações estruturadas sobre a sua definição e o seu comportamento. A manipulação consiste em utilizar as informações obtidas por meio da inspeção para chamar o código dinamicamente, criar novas instâncias de tipos descobertos e, até mesmo, reestruturar tipos e objetos rapidamente. Essa é uma poderosa ferramenta para o desenvolvimento de soluções muito genéricas.
7. **Providers:** oferece a funcionalidade para conexão com banco de dados, execução de comandos e recuperação de resultados.

Para construir a interface para os diversos bancos de dados, foi criada uma classe abstrata, denominada *DB Provider Factory*, responsável pela implementação dos bancos de dados que o sistema irá utilizar.

A classe *DB Provider Factory* apresentada para configurar os *databases*, exposta no diagrama de classes da Figura 5.5, contém os recursos descritos anteriormente.

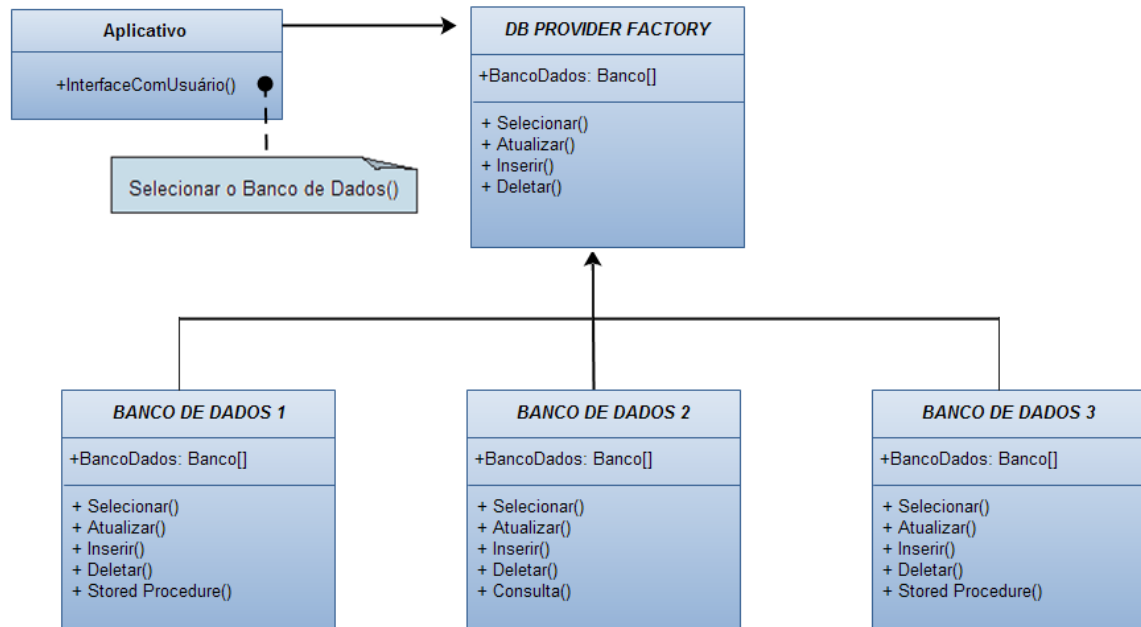


Figura 5.5: Diagrama da classe *DbProviderFactory* com os diferentes bancos de dados.

A classe *DbProviderFactory* representa um conjunto de métodos que cria instâncias de classes, a partir de uma implementação de fonte das interfaces genéricas *IDbConnection*, *IDbDataAdapter*, *IDbCommand*. Nesta classe, são inseridos os objetos genéricos para acesso aos dados, criando-se uma aplicação que contém as interfaces para permitir o acesso simultâneo aos diversos bancos de dados.

A classe *DbProviderFactory* tem a finalidade de fornecer ao desenvolvedor um construtor de *strings de conexão*, fortemente tipado (é uma *string* que contém informações a respeito de uma determinada base de dados; é dentro dela que estão contidos os dados necessários para se conectar ao servidor de banco de dados, ou até mesmo a um arquivo de banco de dados), no qual se tem propriedades exclusivas para cada *provider*. Isso possibilitará

definir os valores para cada uma dessas propriedades, de acordo com a base de dados que irá acessar.

Os métodos e propriedades utilizados foram:

- ✓ **inclusão e alteração de dados:** esses métodos são os mais comuns em qualquer aplicação que utilize banco de dados. Eles serão chamados pela camada de apresentação para realizar a devida operação no banco de dados. Nesses métodos, podem-se encontrar procedimentos comuns em operações de inclusões e alterações;
- ✓ **exclusões:** esses métodos realizam operações de exclusão de dados do banco. Internamente, possuem procedimentos comuns ao processo;
- ✓ **seleções de linhas da tabela:** esses métodos vão dar suporte a consultas e listagens de que a camada de apresentação possa necessitar e também para outros objetos de negócio;
- ✓ **configurações CRUD (INSERT, UPDATE e DELETE):** são críticas quando completam com êxito e aperfeiçoam inserções, atualizações e apagam performance entre a *DAL* e o banco de dados. Em alguns casos, o desenvolvedor usa *store procedures* em toda interação *CRUD*. Nessa situação, as opções *CRUD* não são somente úteis; elas são vitais. Uma *store procedure* pode ser especificada para cada operação *CRUD* através de propriedades nos objetos de negócios, que usam essas configurações de volta, quando interagindo com a *DAL*.

Todo o código específico, relacionado com a fonte de dados (criar conexão, selecionar, incluir, atualizar e deletar dados), será implementado na *BLL* e executado pela *DAL*. A

interação entre o negócio e a *DAL* recebe o comando pronto, ou seja, criado na *BLL* e passado para a *DAL* executar.

A Figura 5.6 apresenta o diagrama para a ferramenta de habilitação ou desabilitação dos bancos, com suas configurações.

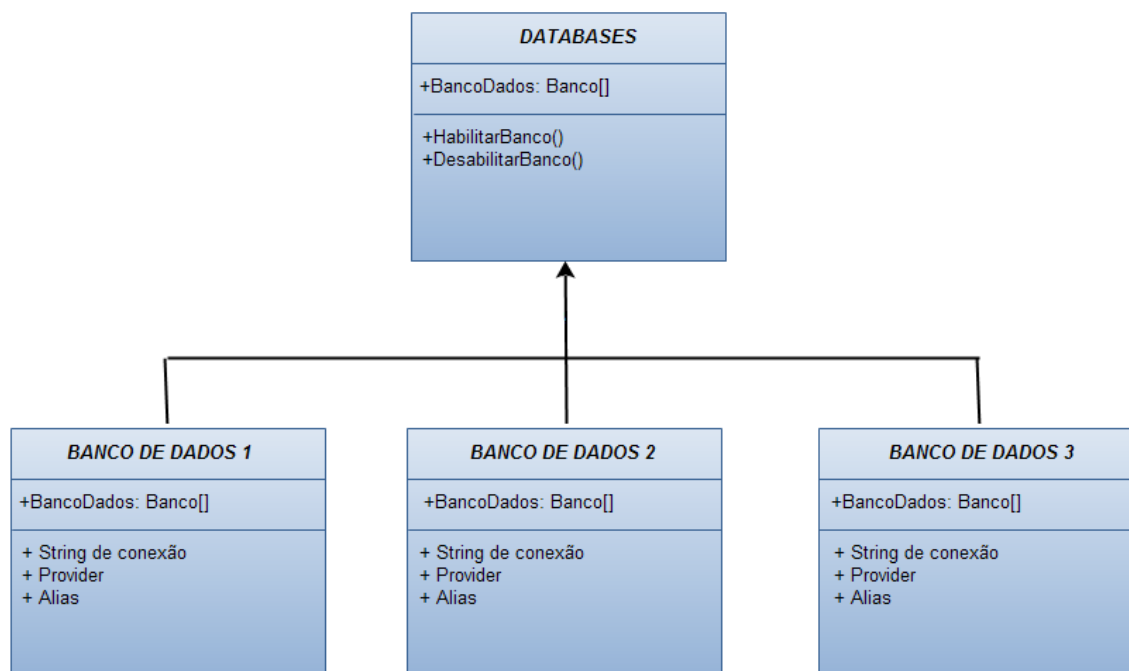


Figura 5.6: Diagrama de classes para habilitar e desabilitar os diversos bancos de dados.

A criação da *interface Databases* permite a seleção dos bancos de dados que serão manipulados mediante a sua habilitação ou desabilitação pelo usuário. Há, ainda, a possibilidade de introdução de novos bancos de dados, adicionando-se novas configurações de acordo com as funcionalidades de cada um deles.

5.3 O Gerenciador do *plug-in*

No *framework*, os *plug-ins* são, na verdade, bibliotecas representadas por arquivos `.dll`, que devem estar disponíveis em um diretório especificado pela aplicação. Essas bibliotecas contêm classes que permitem a comunicação com os bancos de dados.

O sistema de gerenciamento facilita o carregamento dessas classes e é implementado na classe denominada *GPlugin*. O gerenciador de *plug-in* apresenta-se como argumento ao diretório de armazenamento e procura por todas as "dll's" no caminho específico. Quando uma biblioteca solicitada é encontrada, o sistema lista as classes presentes na biblioteca. A partir daí, o sistema carrega o *assembly* para cada um dos tipos dentro dele e gera lista contendo as instâncias do *plug-in*, que deverá ser executado. A localização da biblioteca na qual a classe se encontra contém informações necessárias para gerar uma instância dessa classe. O sistema utiliza essas informações para gerar uma instância da classe, fazendo uso dos recursos de reflexão, retornando uma instância dessa classe, que pode ser utilizada pelo aplicativo.

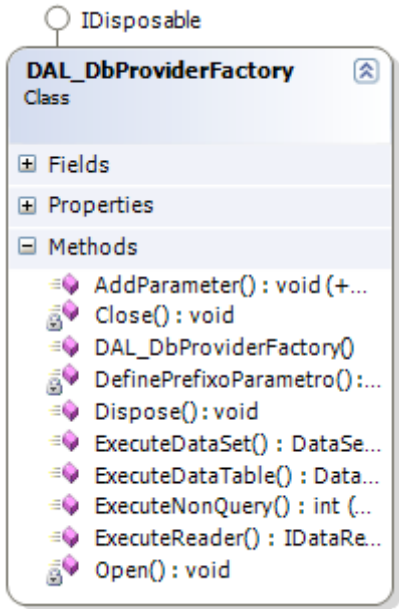
5.4 A implementação do *framework*

O *framework* foi implementado na forma de uma biblioteca, utilizando a versão 2.0 da plataforma .NET. A Figura 5.7 mostra o diagrama de classes gerado pela IDE.

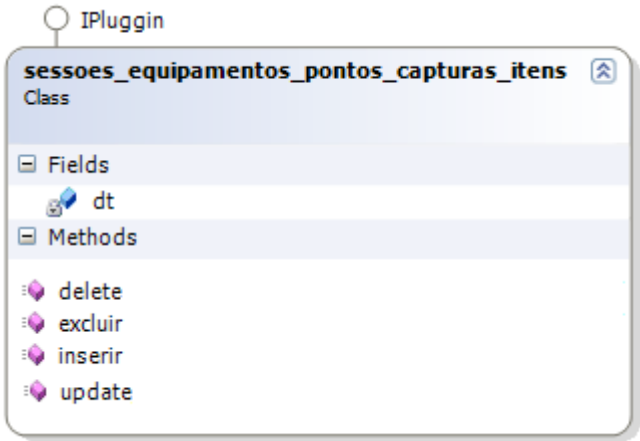
O diagrama da Figura 5.7 mostra as classes implementadas, descritas nas seções anteriores, modificando apenas a sua nomenclatura. Por exemplo, a classe *DAL* recebe o nome de *DAL_DBProviderFactory* no *framework*, já a classe *BLL* representa a classe *SESSOES_EQUIPAMENTOS_PONTOS_CAPTURAS_ITENS*, a classe do *PLUG-IN* recebe o nome de *PLUGGINS*, a classe *ENTIDADES* representa a classe *ENTITYS* e ainda a classe da *INTERFACE*.

A camada de acesso a dados foi feita mantendo as informações sobre os bancos de dados conectados ao sistema e é utilizada pela classe *DAL_DBProviderFactory* para fornecer a lista de bancos de dados encontrados, responsável pelo conhecimento do banco de dados.

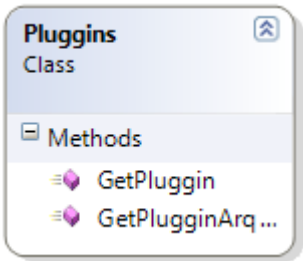
Quando a classe *DAL_DbProviderFactory* for instanciada, ela precisará, de imediato, de descobrir com qual *provider* ela deve trabalhar, checando as configurações do ambiente.



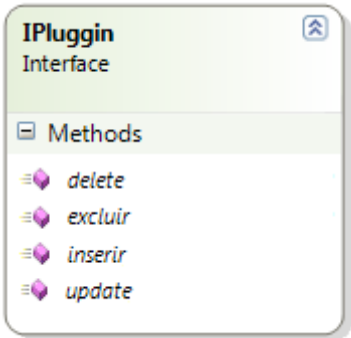
DAL



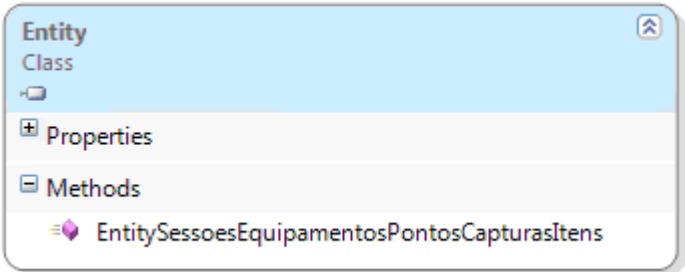
BLL



PLUG-IN



INTERFACE PLUG-IN



ENTITY

Figura 5.7: Diagrama de classes do *framework*.

A classe *PLUGINS* gera o atributo que será utilizado no desenvolvimento dos *plug-ins* para indicar qual o construtor da classe será utilizado.

A camada de lógica de negócios que é a classe *SESSOES_EQUIPAMENTOS_PONTOS_CAPTURAS_ITENS* possui chamadas para a classe de acesso a dados que, por sua vez, retornam dados a esta, que, por sua vez, retorna os referidos dados à camada de apresentação para exibição. É nessa classe que ocorrem as chamadas a vários métodos. Os métodos da *BLL* realizam a busca dos dados a partir de métodos da *DAL*, para os quais são passados parâmetros do tipo instrução *SQL*. Nessa camada, vão-se implementar os métodos de inserção, atualização, exclusão e listagem, referentes a todas as tabelas existentes no projeto. Para isso, utiliza-se o componente de Entidades para objetos de instância, que representam campos da tabela envolvida. Essa classe é formada basicamente por métodos e propriedades. Os métodos serão chamados na camada de apresentação para realizar as operações de negócio. As propriedades irão formar parâmetros de entrada ou saída da classe para determinar algum comportamento do método.

A classe da interface é aquela que contém a assinatura dos métodos que a classe implementa. Essa interface funciona como um contrato que estabelece como “as partes” deverão se comunicar. O uso de interfaces permite a modelagem de objetos e de relacionamentos entre eles, construindo, assim, uma planta do sistema, de uma forma consistente e clara. A partir dessa modelagem, é possível iniciar a construção, seguindo os moldes já existentes.

O fato de todas as classes estarem vinculadas a interfaces comuns permite que os componentes da aplicação apenas façam referência às interfaces e não às classes. Com isso, os componentes podem utilizar uma *classe connection* sem saber exatamente a que *Data Provider* pertence, apenas utilizando as definições de métodos existentes na interface.

Na Figura 5.8, mostra-se a classe da *BLL* que possui chamadas para a classe *DAL* que, por sua vez, retornam dados a esta, e esta retorna os referidos dados à camada de apresentação para exibição; o *plug-in* procura por todas as "dll's" no caminho específico.

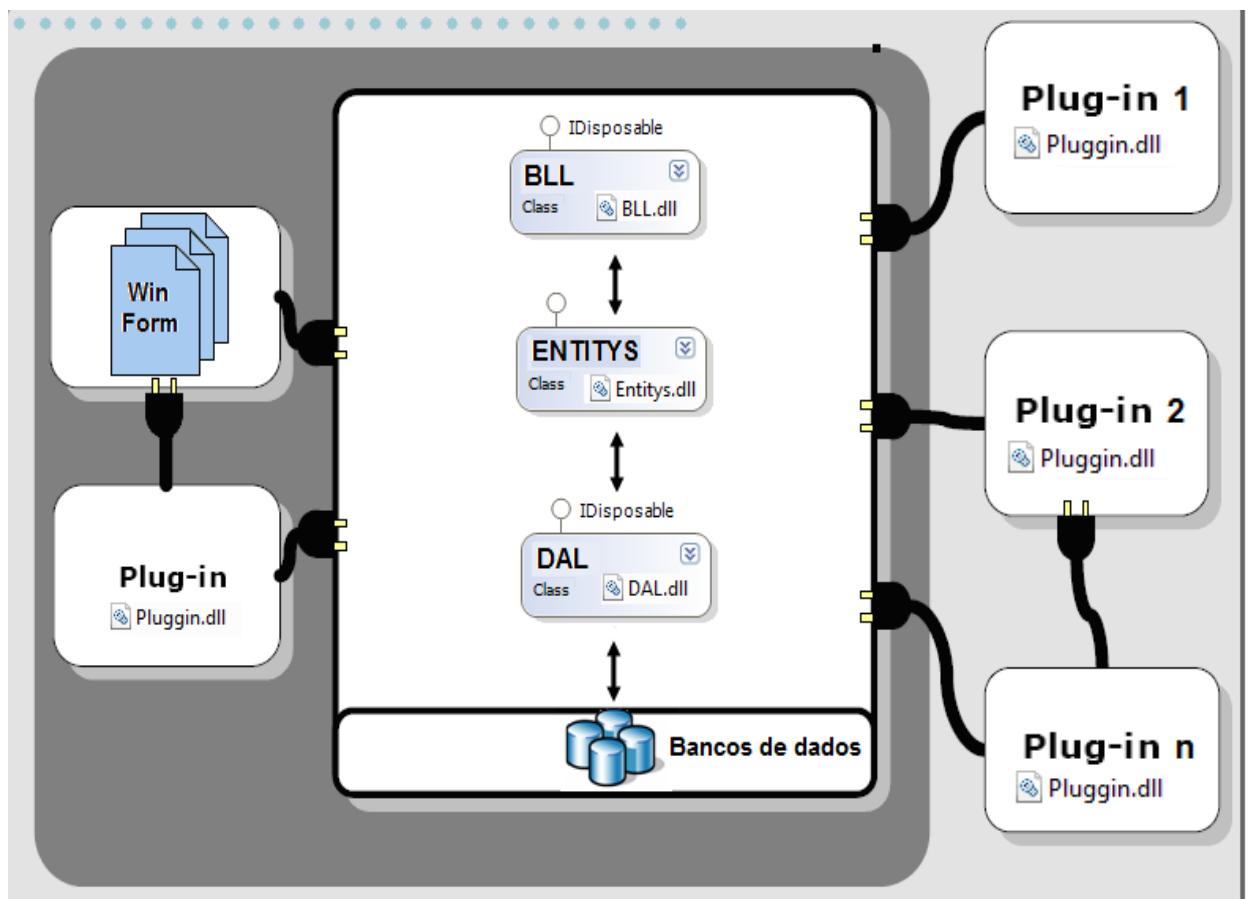


Figura 5.8: Arquitetura do *framework* com suas respectivas dll's.

5.5 Desenvolvendo aplicativos com o *framework*

A construção de um aplicativo que utilize o *framework* deve ser feita utilizando-se uma das linguagens da plataforma *.NET*. As *DLL's*, mostradas na Figura 5.9, representam as classes da arquitetura proposta e, se necessário, devem ser adicionadas ao projeto, permitindo acesso às classes do *framework*.

No modelo proposto, o aplicativo acessa o banco de dados sempre que necessitar. A aplicação realizará o acesso através de um objeto de conexão de um dos *Data Providers* do *Ado.NET*.

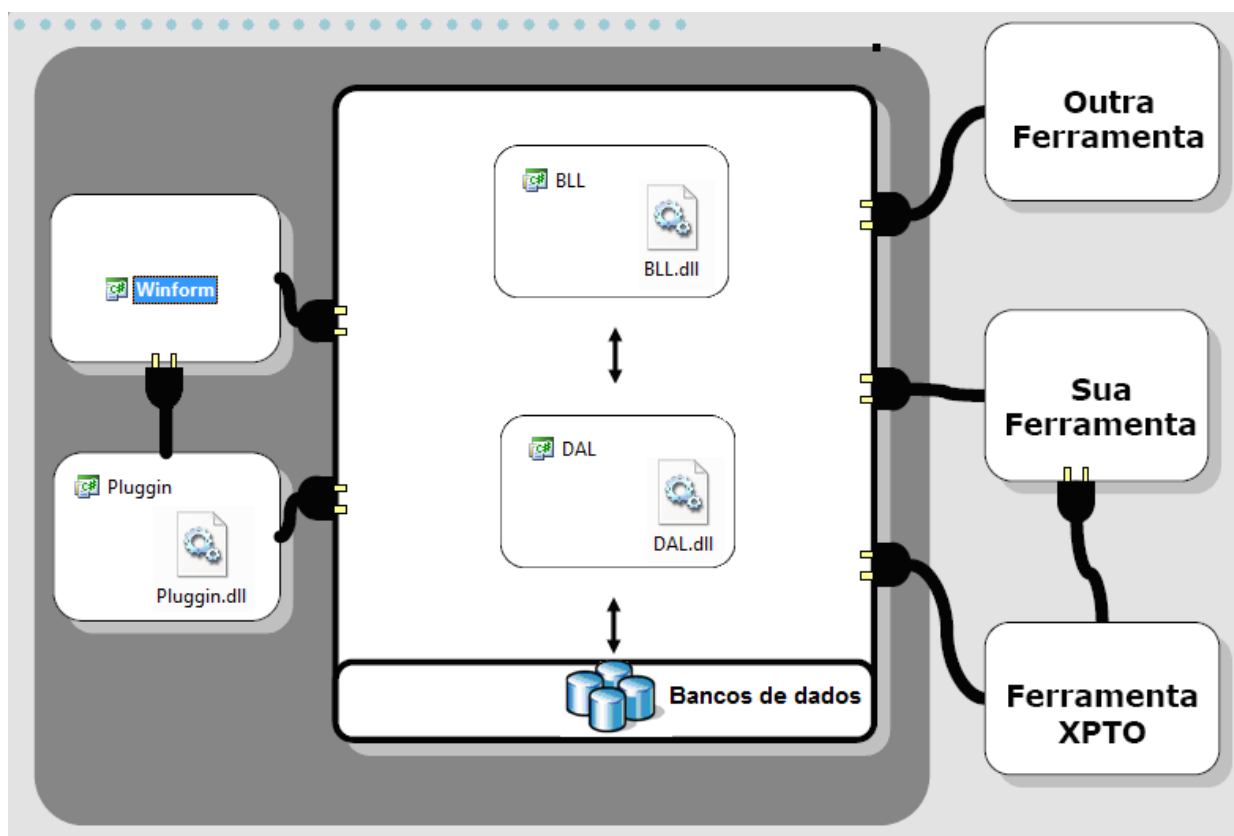


Figura 5.9: Arquitetura do *framework* com outros aplicativos.

O objetivo da classe a ser criada será fornecer uma forma única para ler a estrutura de uma tabela, independente do banco de dados que esteja sendo utilizado. A classe deverá ser implementada para ler a estrutura de uma tabela do banco de dados, utilizando polimorfismo para disponibilizar métodos com nomes iguais, recebendo parâmetros diferentes, tornando transparente seu uso para o desenvolvedor.

Novamente, a principal vantagem é que a mesma implementação passa a servir para diferentes tipos de banco de dados, sem a preocupação com as diferentes bibliotecas de

acesso. Isso foi feito através das novas classes que foram adicionadas ao *.NET Framework* e ao *ADO.NET 2.0* em que foram criados recursos novos e aperfeiçoados que afetam várias classes e diversos cenários de desenvolvimento de aplicativos diferentes. Há, assim, melhorias substanciais, principalmente no que diz respeito à criação do código genérico para múltiplos bancos de dados.

Essa classe fornece uma forma de criar e gerenciar as *ConnectionStrings* que serão utilizadas para criar a conexão com as bases de dados. Todas essas classes foram herdadas de uma classe base chamada *DbConnection*, que está contida dentro do *namespace System.Data.Common*, conforme exemplifica a Figura 4.13 do capítulo 4.

Após a seleção do banco de dados, os demais recursos do sistema tornam-se disponíveis para utilização.

Os métodos descritos acima fornecem a base para a homogeneidade e escalabilidade das aplicações. O *software* pode utilizar essas classes de acordo com a suas necessidades.

5.6 Criando os plug-ins

A criação de plug-ins consiste no desenvolvimento de uma biblioteca da plataforma. Esta biblioteca deve conter uma ou mais classes derivadas do framework. Estas classes devem seguir algumas regras na sua construção como:

- conter um construtor que receba uma string. Este construtor será utilizado para gerar a instância da classe;
- implementar todos os métodos abstratos da classe *DAL_DBProviderFactory*. A classe também deve implementar as propriedades abstratas herdadas;
- utilizar propriedades para implementar as funções de configuração únicas dos bancos de dados.

5.7 Conclusão

Os métodos descritos acima fornecem a base para a construção de um sistema para interface e gerenciamento de diferentes bancos de dados com sistemas de sinais. A construção se dá a partir de modelo em camadas que foram utilizadas no desenvolvimento do *framework*, implementado na forma de bibliotecas de classes de procedimentos. O *software* pode utilizar essas classes de acordo com suas necessidades. Por exemplo, um *software* que necessite obter e armazenar dados oriundos de bancos diferentes poderá utilizar as funções do *framework*.

Detalhes dessa implementação encontram-se no código disponível no CD que acompanha este documento.

O próximo capítulo apresenta a construção de um sistema de dados e visualização de imagens e sinais.

CAPÍTULO 6

PROVA DE CONCEITO

Este capítulo tem por objetivo avaliar se a arquitetura proposta permite a integração de diferentes bancos de dados a diferentes aplicações. O aplicativo exemplo foi desenvolvido para possibilitar o acesso aos bancos de dados *Microsoft Sql Server* e *Microsoft Access*. A mesma estratégia de desenvolvimento poderá ser empregada para uso de outras bibliotecas de acesso a dados, como o *ODBC*, *Oracle* etc. Para essa avaliação, foi criado um aplicativo para armazenamento e acesso a vários tipos de dados obtidos a partir de trabalhos na área biomédica.

O desenvolvimento das várias sub-unidades do sistema exemplo foi realizado, conforme descrito no capítulo 5.

6.1 Estrutura do *framework*

A estrutura do *framework*, conforme exemplifica a Figura 6.1, apresenta um *plug-in* entre a aplicação e os bancos de dados. A aplicação dessa estrutura inclui: inserção de dados, imagens e sinais coletados de pacientes; e, para armazenamento e gerenciamento desses dados, foram utilizados os bancos de dados *Access* e *Sql Server*. O *plug-in* é utilizado para a

construção do aplicativo e o interfaceamento com o banco de dados para facilitar a comunicação e tornar um sistema compatível com diversos bancos de dados e *frameworks*.

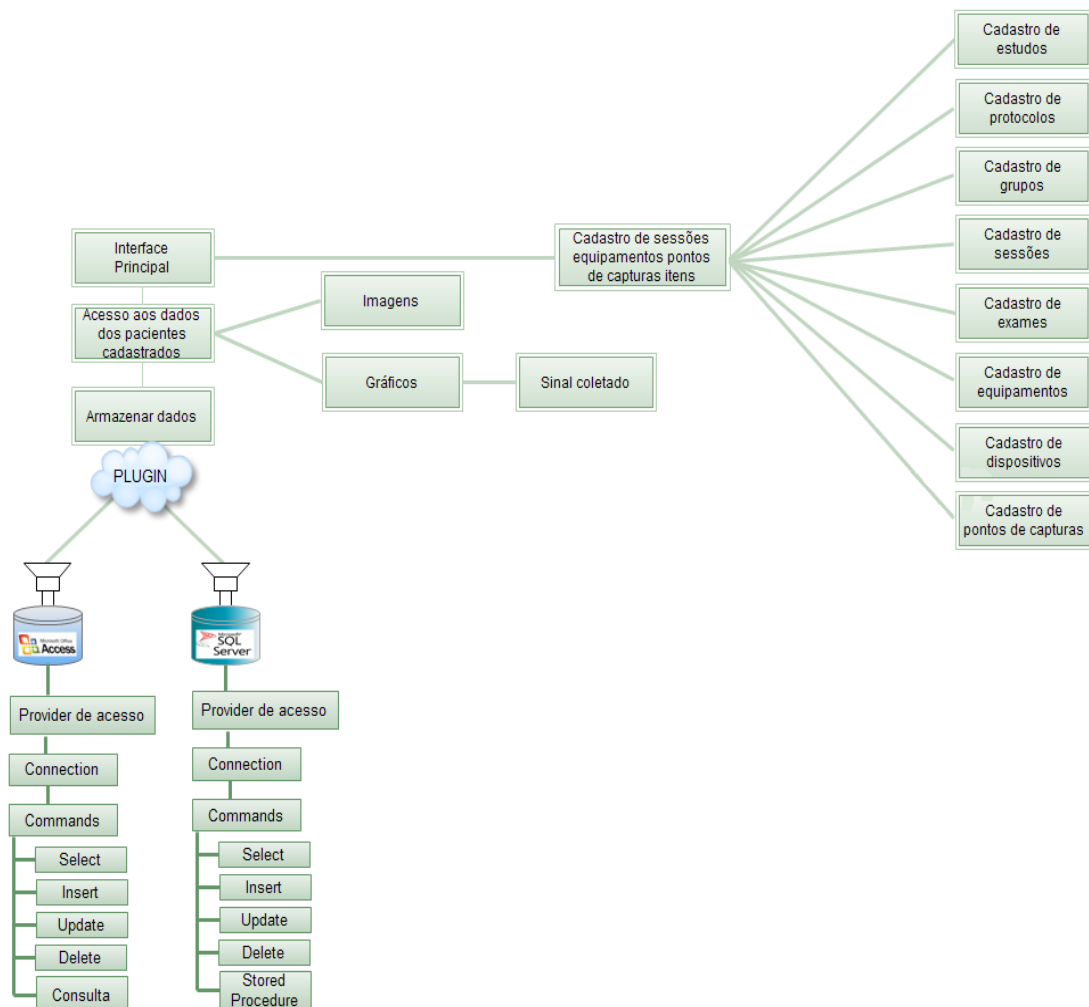


Figura 6.1: Estrutura do *framework*.

A seguir, serão apresentadas as etapas para o desenvolvimento do sistema exemplo, bem como as técnicas utilizadas para a implementação das mesmas.

6.2 Modelagem de Dados

Inicialmente, foi feito um levantamento das principais aplicações de bancos de dados na área biomédica, procurando reunir o maior número possível de informações para a

modelagem de dados. O desenvolvimento do banco de dados iniciou-se com a elaboração da modelagem de dados representada pelo programa *Erwin*. Em seguida, procedeu-se ao processo de identificação das tabelas a serem criadas nos bancos de dados *Access* e *Sql Server* e, posteriormente, desenvolvimento do *plug-in* para o interfaceamento dos bancos de dados com o aplicativo exemplo.

O planejamento da modelagem de dados é exemplificado na Figura 6.2.

Os itens a seguir exemplificam a descrição sucinta das entidades envolvidas e procedimentos adotados no *framework* mostrados na Figura 6.2.

Cidades: registrar o identificador de cidade e sua descrição. Exemplo: Uberaba.

Dependências Físicas: Registrar o identificador das dependências e suas descrições; serão cadastrados os lugares onde serão feitos cada exame. Exemplo: Hospital São João, laboratório biomédica.

Equipamentos: registrar o identificador dos equipamentos relacionados em cada exame e sua descrição. Exemplo: Heart laser, oxímetro, esfigmomanômetro.

Dispositivos de Equipamentos: registrar o identificador dos dispositivos relacionados em cada equipamento e a descrição de cada dispositivo. Exemplo: dispositivo (canal 3, canal 2, eletrodo Ag).

Itens dos Dispositivos de Equipamentos: registrar o identificador de cada item dos dispositivos relacionados em cada equipamento e a descrição do item. Exemplo: item (filtro passa alta, filtro passa baixa).

Estudos: registrar o identificador dos estudos relacionados a cada doença, o número de sessões, a data de início e de término, status do estudo e a descrição do comitê de ética. Exemplo: estudos (angina, distúrbio do sono, Creutzfeldt-Jakob, tremor).

Protocolos de Estudos: registrar o identificador de cada protocolo relacionado a cada estudo e a descrição do protocolo. Exemplo: protocolos (foram analisados 46 pacientes com transplante cardíaco).

Grupos dos Protocolos de Estudos: registrar o identificador do grupo envolvido relacionado a cada estudo, o critério de inclusão e exclusão (quais serão as pessoas (voluntários) que serão incluídas e excluídas em cada estudo). Exemplo: grupo de estudo (pessoas dementes), critério de exclusão (participantes portadores de disfunções neurológicas serão excluídos desse estudo).

Grupos de Pessoas dos Protocolos de Estudos: registrar o identificador da pessoa relacionada em cada grupo de estudo. Exemplo: Maria, José.

Status dos Estudos: registrar o identificador do andamento de cada estudo envolvido e sua descrição. Exemplo: em andamento, concluído, paralisado, aguardando aprovação e cancelado.

Exames: registrar o identificador do exame em que cada pessoa será envolvida e sua descrição. Exemplo: polissonografia, ECG, EEG, EMG, EOG.

Equipamentos dos Exames: registrar o identificador do exame e do equipamento para serem relacionados nas tabelas.

Itens dos Exames: registrar o identificador de qual será o tipo de exame, o item e sua descrição. Exemplo: item do exame (oxigenação e frequência cardíaca, pressão arterial, esteira e ecocardiograma de esforço).

Medicamentos: registrar o identificador dos medicamentos que serão passados para cada doença e sua descrição. Exemplo: Digoxina, Antagonistas do cálcio, mepacrina e fenotiazínicos.

Pessoas: registrar qual o nome da pessoa envolvida, com todos os seus dados pessoais (data de nascimento, sexo, tipo sanguíneo, endereço, bairro, cep, telefone, peso e altura).
Exemplo: Ana, João.

Pontos de Captura: registrar o identificador dos pontos de captura onde serão posicionados o dispositivo e sua descrição. Exemplo: distal valva tricúspide e ante braço esquerdo.

Sessões: registrar o identificador de cada sessão, qual o tipo de estudo, protocolo, grupo, pessoa, exame, dependência, data, hora, termo de consentimento e anamnese.

Sessões dos Pontos de Captura dos Equipamentos: registrar o identificador do tipo de estudo, protocolo, grupo, sessão, pessoa, exame, equipamento, dispositivo do equipamento, ponto de captura.

Configurações das Sessões dos Equipamentos dos Pontos de Captura: registrar o identificador do tipo de estudo, protocolo, grupo, sessão, pessoa, exame, equipamento, dispositivo do equipamento, item, valor 1, valor 2, valor 3, valor 4 e valor 5. Exemplo: valores (exibe os valores da frequência de cada equipamento: 15, 30, 75, 100, 150).

Itens das Sessões dos Equipamentos dos Pontos de Captura: registrar, em cada sessão, os identificadores de: estudo, protocolo, grupo, sessão, pessoa, exame, equipamento, dispositivos, ponto de captura, sinais e imagens.

Eventos das Sessões: registrar, em cada sessão, os identificadores de: sessão, pessoa, exame, evento e a sua descrição. Exemplo: paciente muito nervoso.

Sessões de Medicamentos: registrar, em cada sessão, os identificadores de: sessão, pessoa, exame, medicamento.

Tipos de Pessoas: registrar o identificador: tipo de pessoa. Exemplo: pesquisador, voluntário, administrador do sistema, usuário do sistema.

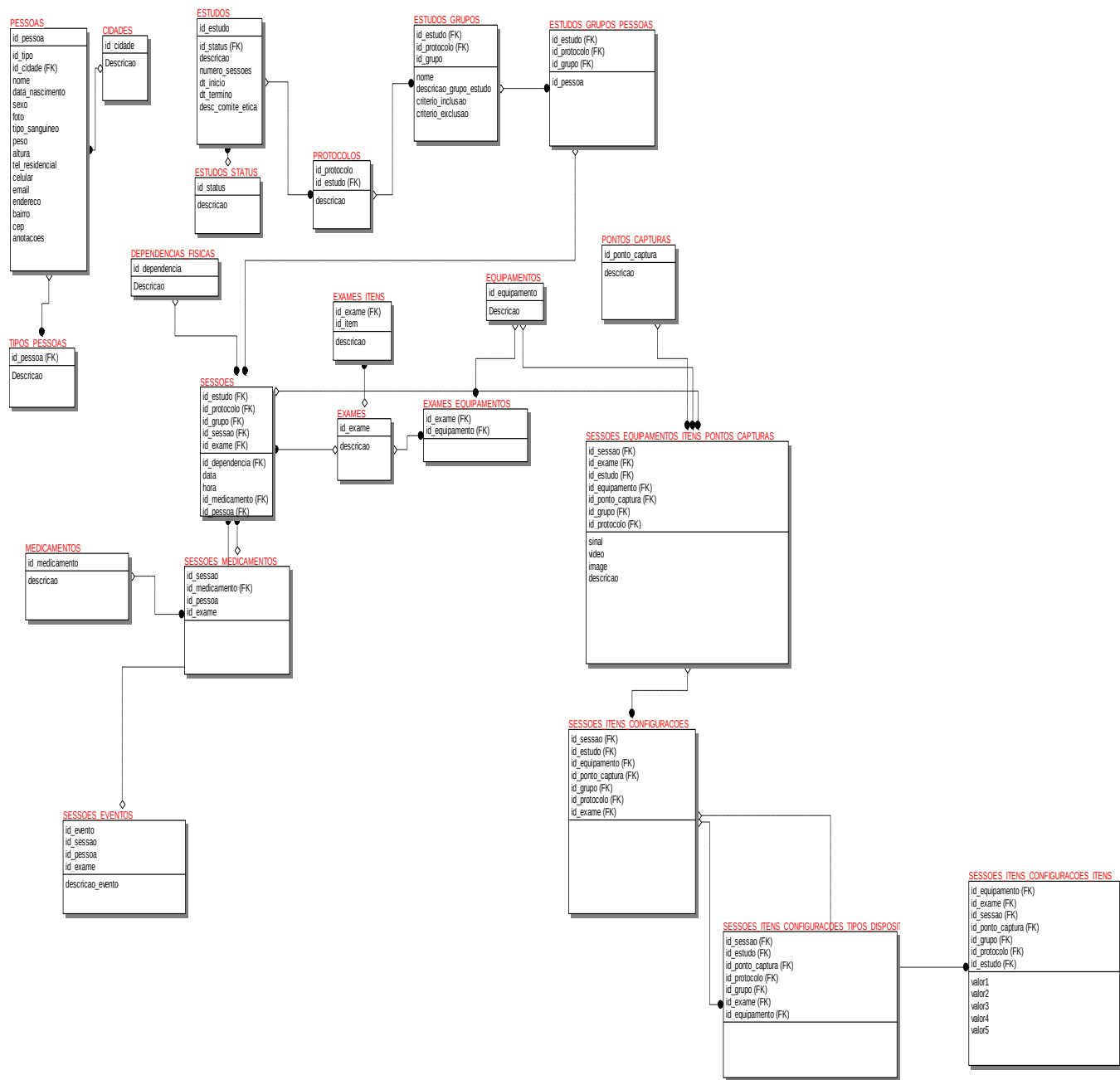


Figura 6.2: Visão geral da modelagem de dados do aplicativo exemplo.

6.3 Criação dos registros nos bancos de dados

O próximo passo executado foi criar os registros clínicos, de cada paciente, nos bancos de dados.

Com a linguagem de pesquisa (*SQL*) integrada à linguagem de programação *C #*, é possível modelar um banco de dados através de classes, para realizar consultas no banco de dados, como também inserir, atualizar ou deletar dados. A linguagem executará as instruções *SQL* apropriadas de inserir, atualizar e deletar, quando o método equivalente for invocado.

A linguagem de programação permite o modelamento de classes que mapeiam os bancos de dados. Dessa forma, tais classes representam tabelas dos bancos de dados.

Assim, para cada tabela no banco de dados, uma classe é criada. Esta classe permite a pesquisa dos dados provenientes dos bancos de dados e fornece toda a infraestrutura necessária para aplicar mudanças nos bancos de dados.

Será adotado um mapeamento entre classe e tabela, na relação de um para um, no qual cada classe terá uma tabela correspondente de mesmo nome.

6.4 Acesso ao *plug-in*

A Figura 6.3 mostra o diagrama de classe do *plug-in* gerado pela *IDE*, conforme foi visto no capítulo 5. A classe chamada *Pluggins* procura por todas as "*dll's*" num caminho específico e, por meio de reflexão, carrega o *assembly* para cada um dos tipos dentro do próprio *assembly*;

- verifica se a *interface IPluggin* está devidamente implementada;
- cria uma instância do *plug-in* e a adiciona à lista de *plug-ins* disponíveis e à lista contendo instâncias de todos os *plug-ins* que poderão ser executados pelo aplicativo.

A interface *IPluggin* é um protocolo que define métodos que deverão ser obrigatoriamente implementados pelas classes que a utilizarem.

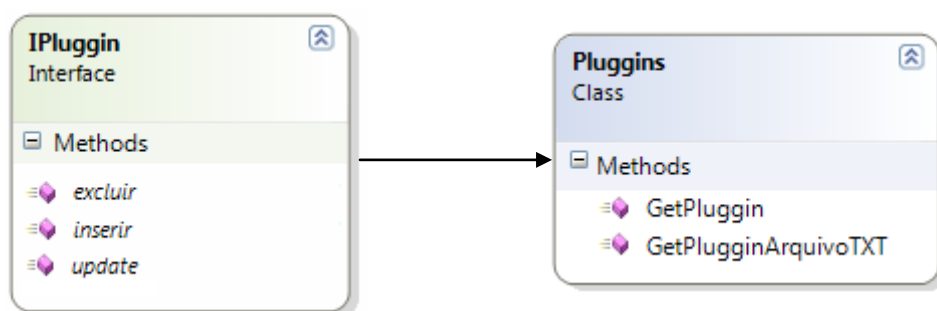


Figura 6.3: Classe *plug-in* do *framework*.

6.5 Aplicativo

Após a conclusão dos passos anteriores, procedeu-se o desenvolvimento do aplicativo para avaliar a capacidade de conexão do aplicativo com os bancos de dados *Microsoft Access* e *Microsoft Sql Server*. O objetivo é desenvolver um aplicativo capaz de acessar ambos os bancos de dados em um único módulo.

6.5.1 Interface Principal

O aplicativo tem como finalidade principal, registrar pacientes, doenças, estudos científicos, exames, sessões realizadas no decorrer do tratamento, medicamentos, equipamentos utilizados como EMG, ECG, EEG, posicionamento dos sensores, imagens (fotos, tomografia etc) e sinais coletados de equipamentos como EMG, ECG e EEG.

Para avaliar a operação e eficiência do *framework*, foram criados registros clínicos de pacientes, no banco de dados *Microsoft SQL Server* e *Microsoft Access*. A Figura 6.4 mostra a tela principal do aplicativo.



Figura 6.4: Interface principal do *framework* para interface e gerenciamento de bancos de dados.

6.5.2 Menu do programa

6.5.2.1 Menu Auxiliar

O menu auxiliar é o local do programa no qual o usuário deve optar pelo banco de dados que será utilizado.

O aplicativo é capaz de acessar os bancos de dados através do *plug-in*, podendo alternar a utilização entre os bancos de dados *Microsoft Access* e *Microsoft Sql Server*. Nesse contexto, vale notar que a programação, utilizando reflexão e algumas outras classes, permitiu o desenvolvimento, de forma independente, do banco de dados a ser utilizado.

Para ativar um banco de dados, o usuário deverá marcar a sua respectiva caixa de checagem, conforme exemplificado na Figura 6.5.

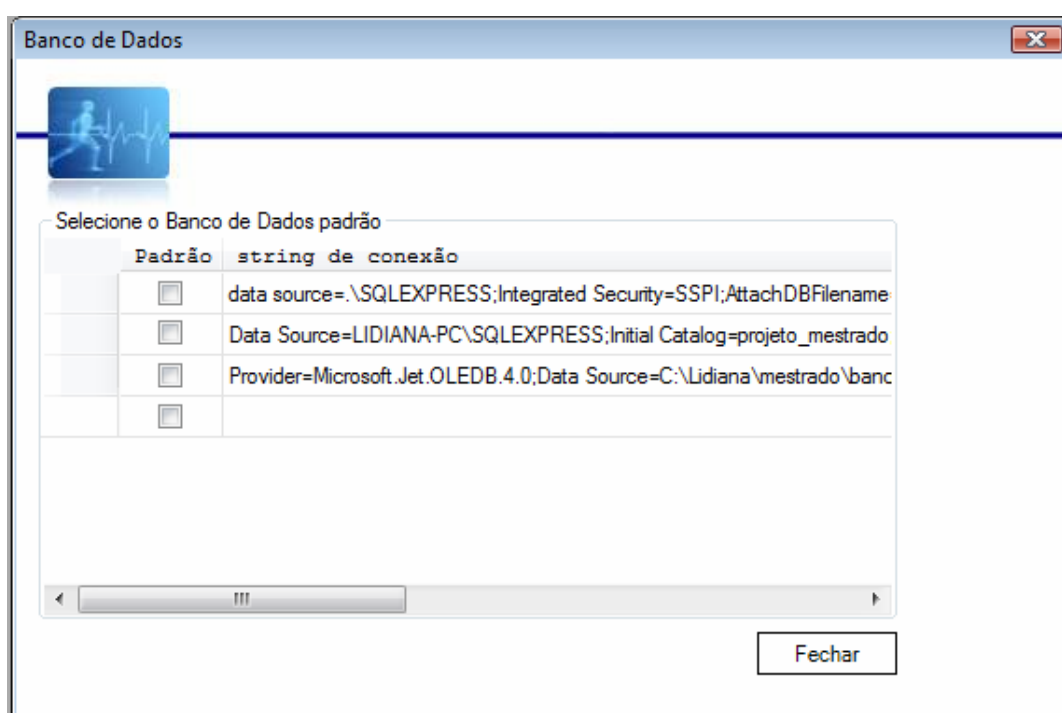


Figura 6.5: Interface para escolha do banco de dados a ser utilizado do *framework* para interface e gerenciamento de bancos de dados.

6.5.2.2 Menu Cadastros

No menu cadastros, o usuário poderá acessar a tela com diversas opções para cadastramento de sessões, equipamentos, dispositivo, pontos de capturas, protocolos, grupos, estudo, exame, sinal coletado, imagem e informações relativas aos pacientes, conforme exemplifica Figura 6.6.

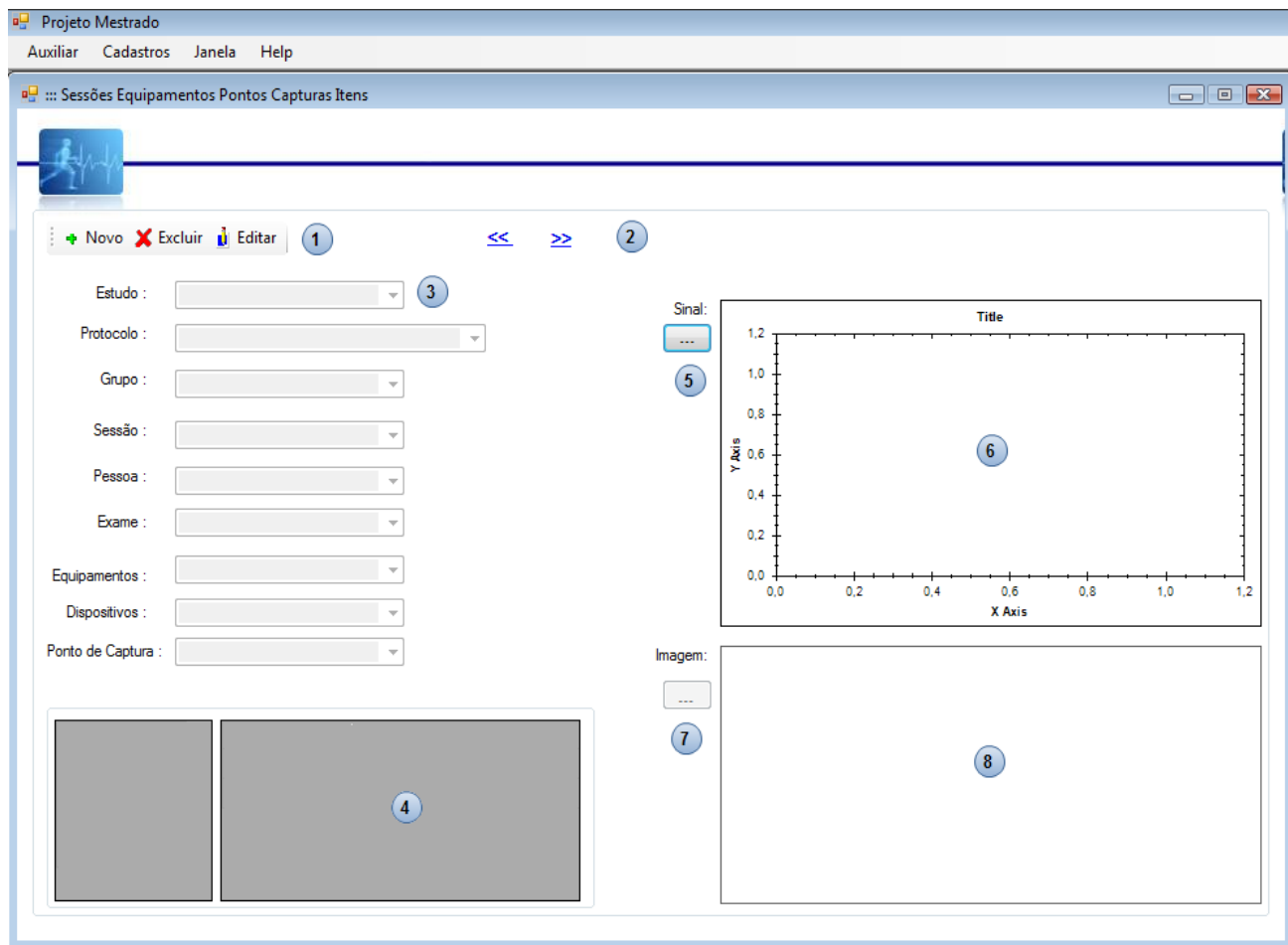


Figura 6.6: Interface de cadastramento do sinal coletado, imagem e informações relativas aos pacientes do *framework* para interface e gerenciamento de bancos de dados.

A Figura 6.6 mostra uma interface composta por cadastrar imagens e sinais coletados, combinada com informações dos pacientes permitindo recuperar informações gerais, para serem usadas em consultas, estudo comparativo e arquivos de ensino. Essa interface integrará diversos tipos de informação, como:

1. Os botões de ação: no botão novo poderão ser inseridos novos registros para cada *combobox*. O botão excluir deleta informações do paciente. Com o botão editar, poderá ser editada uma nova imagem ou um novo sinal.
2. As teclas de navegação estão relacionadas à movimentação por entre os registros.

3. Registros clínicos dos pacientes, como estudo, protocolo, grupo, sessão, pessoa, exame, equipamentos, dispositivos e pontos de capturas.
4. Registros da sessão, data, hora e descrição relacionado ao paciente.
5. Botão para buscar um arquivo TXT com dados de sinais biológicos de pacientes, para posteriormente serem plotados.
6. Exibir sinais biológicos (ECG, EEG e EMG).
7. Botão para buscar os arquivos para a visualização das imagens.
8. Exibir as imagens médicas (radiografias, tomografias, sonografias, fotos etc).

A Figura 6.7 exemplifica a edição dos dados que foram cadastrados no banco de dados para exibição na *combobox*.

Projeto Mestrado

Auxiliar Cadastros Janela Help

Sessões Equipamentos Pontos Capturas Itens

Salvar Excluir Editar

Estudo : Distúrbio do sono

Protocolo : Distúrbio do sono

Grupo :

Sessão :

Pessoa :

Exame :

Equipamentos :

Dispositivos :

Ponto de Captura :

nome	id_sessao	data	hora
Maria	26	01/05/2008	12:00
Pedro			
Maria			

Sinal:

Y Axis

X Axis

Imagem:

Figura 6.7: Interface fornecida pelo sistema ao usuário para edição de dados.

6.6 Exemplos de utilização

Com a grande utilização de sistemas de bancos de dados e a diversidade das aplicações, cresce a necessidade de se permitir ao usuário o acesso integrado a informações, de forma heterogênea, autônoma e transparente. Faz-se necessário, portanto, prover de informação um ambiente aberto e integrado, buscando-se promover a interoperabilidade dos sistemas envolvidos. Essa arquitetura desenvolvida permite acessar bases de dados diferentes com provedores específicos de cada base, utilizando o mesmo modelo. Esse enfoque permite que os usuários possam acessar os dados sem tomar conhecimento de como estes estão distribuídos; deixando, porém, o usuário com liberdade para escolher que bancos de dados devem ser acessados e como os dados devem ser transformados.

A) O Aplicativo acessando o banco de dados *Microsoft Access*

A Figura 6.8 exemplifica a seleção do banco de dados *Microsoft Access* e, logo após, mostra os dados, a imagem e o sinal do paciente, já cadastrados os seus dados.

A visualização dos dados na tela será feita da seguinte maneira. O usuário selecionará o nome do paciente onde já foram cadastrados seus dados pessoais (nome, qual o tipo da pessoa, data de nascimento, sexo, tipo sanguíneo, cidade, endereço, bairro, cep, telefone, celular, *e-mail*, foto, peso e altura) nos bancos de dados *Microsoft (Sql Server e Access)*. Logo após a seleção do nome do paciente feita pelo usuário, mostrará as sessões cadastradas para essa representada. O usuário pode consultar id da sessão, data, hora e descrição, cadastradas para cada sessão de cada pessoa, clicando na linha correspondente ao id da sessão. Após o clique, são automaticamente atualizados, nas *combobox's*, os dados necessários de cada paciente.

Além dos dados dos pacientes poderão ser vinculados suas respectivas imagens digitalizadas e sinais coletados como o EMG, ECG e EEG, contendo os dados de avaliações do indivíduo, com o propósito de auxiliar o ensino e a pesquisa.

Na região direita, são exibidos os arquivos dos sinais coletados, gerando o gráfico corresponde ao sinal obtido.

O aplicativo foi criado para permitir a visualização gráfica de sinais captados por um equipamento qualquer. Para isso, os gráficos são desenhados à medida que as amostras dos sinais são buscadas de um arquivo TXT e, posteriormente, plotadas, fornecendo uma visualização do sinal coletado. O eixo das abscissas representa o tempo, enquanto que o eixo das ordenadas corresponde à amplitude.

A proposta é a digitalização das imagens, dos sinais e o gerenciamento dos dados a eles relacionados.

B) O Aplicativo acessando o banco de dados *Microsoft Sql Server*

O aplicativo é capaz de acessar um único banco de dados em um determinado momento; entretanto, pode alternar a utilização entre diferentes bancos de dados, conforme exemplifica a Figura 6.9, que anteriormente acessava o banco de dados *Access* e agora alterna para o *Sql Server*.

A Figura 6.9 exemplifica o mesmo aplicativo apresentado para o banco de dados *Access*, permitindo acessar também o banco de dados *Sql Server*.

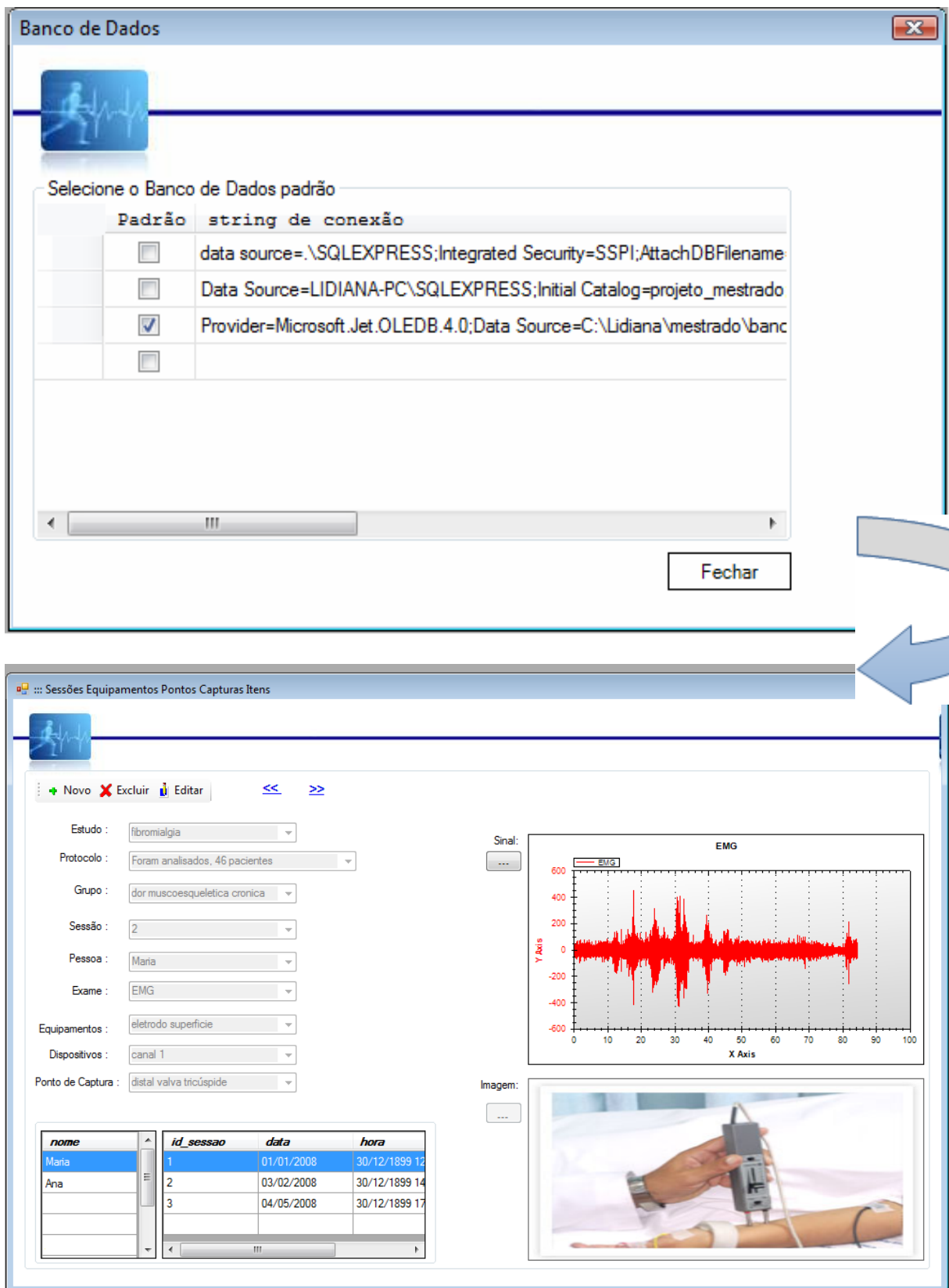


Figura 6.8: Aplicativo exibindo os dados, a imagem e o sinal do paciente no banco de dados *Microsoft Access*.

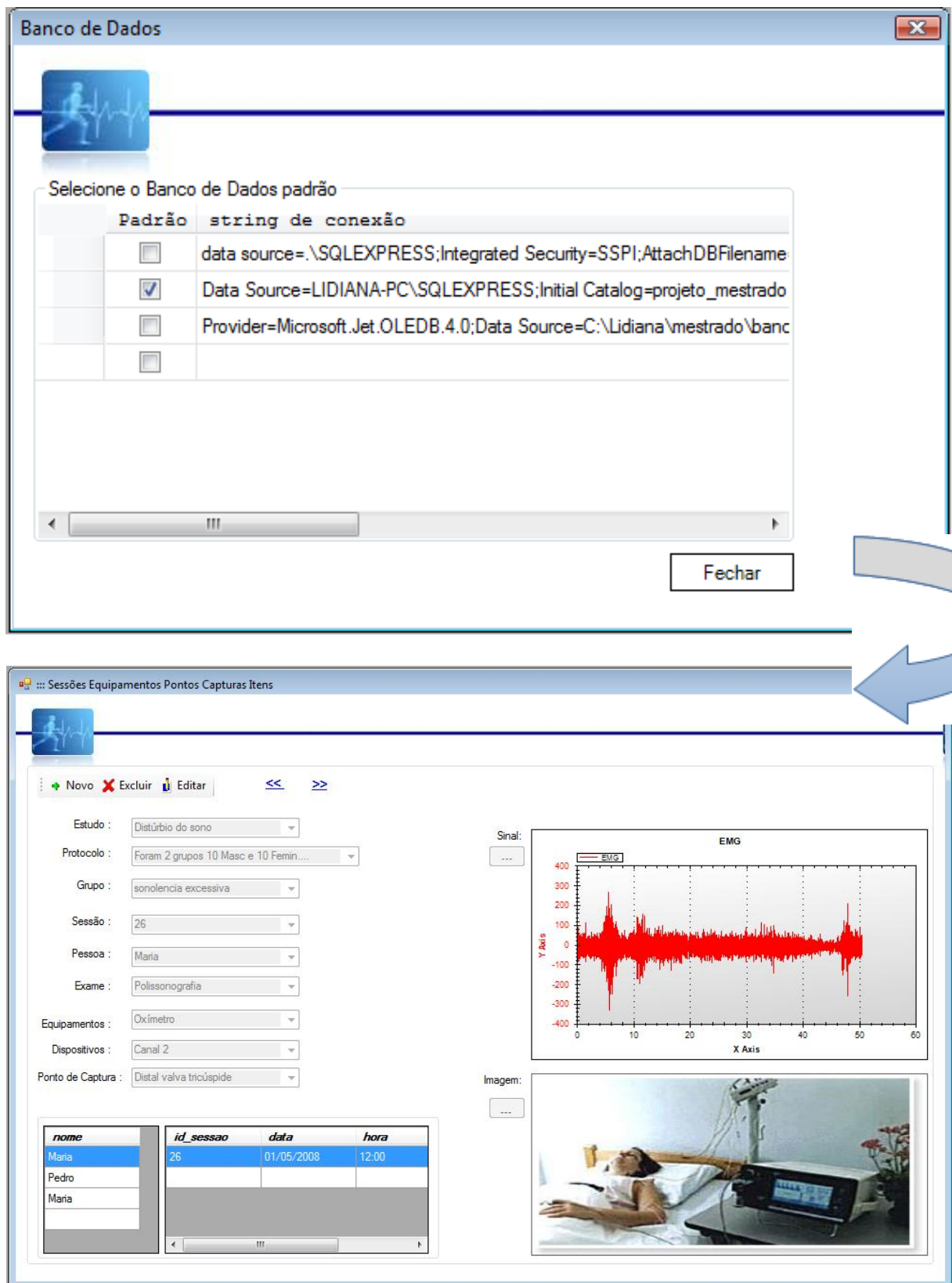


Figura 6.9: Aplicativo exibindo os dados, a imagem e o sinal do paciente no banco de dados *Microsoft Sql Server*.

6.7 Atualizações no sistema para outros bancos de dados

Foi exemplificado os dados de duas bases diferentes com o mesmo código. A aplicação recuperou os dados de uma base de dados dentro de um servidor *SQL Server* e também de dados provenientes de um arquivo do *Microsoft Access*, com o uso do código genérico usando as *classes base* que passam a ser herdadas pelos objetos concretos e lá foram implementados os métodos de acordo com o *provider* específico. Depois das classes de acesso a dados de bases diferentes, cria-se uma *factory*, que em tempo de execução, utilizando a reflexão, vai identificar qual das classes de acesso a dados irá instanciar, baseando-se em uma chave definida no arquivo *app.Config* da aplicação. Feito isso, a *factory* se encarrega de verificar qual dos objetos instanciar, verificando no *app.Config* qual a base de dados a ser utilizada, e através da reflexão, carrega e cria a instância da mesma através do método *factory*.

Uma das dificuldades no desenvolvimento de *software* ocorre quando surge alguma alteração após o seu desenvolvimento, exigindo a manutenção de código. Para que outros bancos de dados utilizem o *framework* é necessário modificar o *plug-in*, para incluir o novo recurso. Isso ocorre porque o aplicativo “enxerga” todos os bancos de dados, como sendo do mesmo tipo. As regras para criação de *plug-ins* foram descritas no capítulo anterior.

Para customizar outro banco de dados, tem que herdar a classe base e implementar os métodos e propriedades de acordo com a sua base de dados e também criar o *provider factory*. Para suprir essa necessidade tem que criar a instância do objeto concreto uma classe abstrata chamada *DbProviderFactory*, que contém vários métodos que devem ser sobrescritos nas classes concretas, retornando os objetos específicos de cada *provider*. Isso quer dizer que, para cada *provider*, temos uma classe responsável por retornar uma instância dos objetos concretos. Estas classes retornam conexões, os comandos, entre outros.

Requisitos para que a aplicação atenda a mais um tipo diferente de base de dados, por exemplo banco de dados nuvem, teríamos que seguir o seguinte procedimento:

1. Instalação do banco de dados.
2. Criação da estrutura de tabela de banco de dados (as tabelas, seus campos e tipos de dados devem ser iguais entre todas as bases de dados utilizadas, para que não ocorram erros que provavelmente somente serão detectados em tempo de execução da aplicação).
3. Criar classes de acesso a dados, utilizando o banco de dados desejado.
4. Compilar estas novas classes.
5. Alterar a configuração no arquivo *app.config* para chamá-la.

6.8 Conclusão

O estudo de caso do *framework* tem como objetivo reunir dados de diferentes bancos de dados e torná-los compatíveis. O processo de integração de dados visa possibilitar ao usuário acesso uniforme a várias fontes de dados heterogêneas, através de um único esquema. Construir um esquema global, a partir de várias fontes locais, foi uma tarefa trabalhosa, já que estas fontes armazenam diferentes tipos de dados, variedade de formatos, diversidade de semântica associada aos elementos, além dos problemas clássicos de nomenclatura, como sinônimos e homônimos.

Assim, a tarefa principal que foi tratada em integração dos bancos de dados diferentes é a identificação de conceitos equivalentes.

CAPÍTULO 7

CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho apresentou, inicialmente, uma introdução sobre o desenvolvimento de banco de dados, levantando questões como a falta de compatibilidade entre bancos de dados diferentes, dificuldades para acessar os recursos específicos de cada banco de dados e utilização dos mesmos, simultaneamente ou não, em um único *software* e vice-versa. Como solução para esses problemas, foi proposta uma arquitetura baseada em *plug-in*, para realizar o interfaceamento entre aplicativos de *software* e bancos de dados. A estrutura básica da arquitetura foi implementada na forma de um *framework*, sobre o qual podem ser desenvolvidos bancos de dados e aplicativos para armazenar e acessar informações variadas. O *framework* foi desenvolvido de forma a isolar o desenvolvimento do aplicativo do desenvolvimento do banco de dados, permitindo maior flexibilidade e reusabilidade. Tal arquitetura pode diminuir, consideravelmente, o impacto de alterações em uma unidade no funcionamento da outra. O *framework* também fornece uma interface única para conexão com bancos de dados distintos em um único aplicativo de *software*. As técnicas de reflexão com o *DB Factory*, utilizadas neste trabalho, permitem ao *framework* realizar a conexão entre o aplicativo e o banco de dados, de maneira transparente e com total acesso a seus recursos.

Durante o desenvolvimento do *framework*, houve uma preocupação maior com a compatibilidade entre bancos de dados, uma vez que cada um possui características específicas quanto ao modo como os dados são armazenados e indexados, tipos de dados disponíveis e capacidades de armazenamento. Para possibilitar o acesso a diferentes tipos de bancos de dados, utilizou-se o recurso da orientação a objetos, que possibilita que uma mesma classe possua métodos com o mesmo nome, até mesmo com a mesma quantidade de parâmetros, desde que os parâmetros sejam diferentes. Porém, os métodos foram usados com nomes iguais, não modificando o *namespace* utilizado, eliminando a necessidade de realizar essa transformação que é diferente para cada banco de dados. Foi empregada ainda a classe *DB Provider Factory*, que é usada para criar um conjunto de objetos em que cada tipo de base de dados a ser acessado; tem-se, assim, uma classe derivada de uma classe abstrata, em que uma série de classes específicas de provedores é requisitada com a fonte de dados a ser acessada.

O sistema desenvolvido foi avaliado através de experimentos com os bancos de dados *Microsoft Access* e *Microsoft Sql Server*, em tarefas como, por exemplo, armazenamento e visualização de sinais e dados de pacientes. O aplicativo foi capaz de armazenar os dados, utilizando bancos de dados diferentes, acessando a mesma informação de dados com todos os seus recursos, bastando para isso a instalação do *plug-in* e sem a necessidade deste ser modificado. Essa integração ocorreu de maneira bastante simples, devido à interface única com os bancos de dados, fornecida pelo *framework*.

A adaptação ocorreu em tempo de execução, não sendo necessário recompilar ou reiniciar o aplicativo para que ela ocorresse, comprovando que o sistema é capaz de adaptar-se ao substituir, alterar ou incluir um banco de dados.

Dessa forma, pode-se dizer, a arquitetura proposta, através do uso de sistemas adaptativos e de *plug-ins*, possui grande potencial para minimizar os problemas de falta de compatibilidade entre bancos de dados, as dificuldades de manutenção de *software* e de integração entre aplicativos e bancos de dados.

Certamente, com o tempo, novas operações devem ser acrescentadas. Mas, é importante ressaltar que novas inclusões devem ocorrer sem que haja necessidade de mudança substancial no modelo atual. Sendo assim, as facilidades que esse tipo de implementação traz são muitas, pois, quando um programador estiver desenvolvendo um aplicativo para conectar com um determinado banco de dados, ele contará com a interoperabilidade com outros tipos de bancos de dados. Outra vantagem é a possibilidade de realizar consultas em mais de um banco de dados a partir da mesma interface de aplicativo. Permite-se, assim, um alto desempenho no acesso a dados de qualquer tipo de fonte. Isso é possível, pois ele oferece uma visão unificada dos diferentes fornecedores de dados existentes na atualidade.

O desenvolvimento deste trabalho oferece aos sistemas públicos ou privados a possibilidade de integração de informações, oriundas de diferentes sistemas gerenciadores de bancos de dados, sem a necessidade de junção material dos dados em um único sistema. Com o desenvolvimento de *plug-ins* e aplicativos apresentados neste trabalho, é possível acessar diferentes fontes de informações de interesse dos profissionais de saúde, desde cadastros e fichas de *anamnese* até resultados de exames complexos com imagens e sinais, armazenados em diferentes sistemas de instrumentos e *softwares*. De posse de informações mais completas e facilmente acessíveis, podem-se alcançar diagnósticos mais precisos, tratamentos mais adequados e maior eficiência em serviços de saúde.

7.1 Trabalhos Futuros

O protótipo desenvolvido atende aos objetivos inicialmente traçados, mas ainda existem vários pontos em que é possível realizar melhorias e otimizações. Dentre esses, destacam-se:

- criar novos ambientes para que o *framework* se torne um sistema completo no armazenamento dos dados;
- avaliar se as potencialidades do *framework* em relação a diferentes bancos de dados atende à maior acessibilidade possível e, caso contrário, modificá-lo, tornando-o ainda mais genérico;
- verificar a aplicação do *framework* em sistemas comerciais, como, por exemplo, aqueles destinados à coleta e análise de dados biomédicos etc.

REFERÊNCIAS

AECE, I. ADO.NET - Generic Code, 2005. Disponível em www.israelaece.com/2005/11/default.aspx. Acesso em 12 jan 2009.

AZEVEDO, C. Especial sobre .Net. Artigonal. Diretório de artigos. 14/02/2008b. Disponível em: <http://www.artigonal.com/programacao-artigos/fundamentos-teoricos-sobre-c-333204.html>. Acesso em 18 ago 2008.

BABCOCK, C. The survey compared Oracle with DB2, MySQL, Informix Dynamic Server, PostgreSQL, Microsoft SQL Server, and Sybase Advanced Server Enterprise. Revista Information Week de 4 de Março de 2008. Disponível em www.informationweek.com/news/software/enterpriseapps/showArticle.jhtml?articleID=206901483. Acesso em 12 Jul 2008.

BARBIERE, C. Modelagem de Dados. Rio de Janeiro: IBPI Press, 1994. 350 p.

BARRETO J; GOMES. C e FUKS, H. Agregando frameworks de infra-estrutura em uma arquitetura baseada em componentes [recurso eletrônico]: um estudo de caso. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. 2006. Disponível em: http://www2.dbd.puc-rio.br/pergamum/tesesabertas/0410823_06_cap_03.pdf. Acesso em 02 jan 2009.

CALÇADO, P. Arquitetura de Camadas em Java EE. Mundo Java. Rio de Janeiro, v.3, n.15, p.34-43, 2005.

CARVALHO, R. M. Criando uma camada genérica de acesso a bancos de dados, 2008. Disponível em www.devmedia.com.br/articles/viewcomp.asp?comp=3811. Acesso em 12 fev 2009.

CARVALHO, S. H; Monitorização de Sinais Biomédicos em Assistentes Pessoais Digitais, 2003. Disponível em: www.sbis.org.br/cbis9/arquivos/735.doc. Acesso em 15 dez 2008.

CEMBRANELLI, F. Revista .net Magazine Edição 49 artigo .net Magazine 49 - Arquitetura Plugável em .Net, 2008.

COSTA, B. S. J. Desenvolvimento de componentes visuais para monitoração de processos na indústria. Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Departamento de Engenharia de Computação e Automação. Natal, Junho de 2008. 45 P. Disponível em: <ftp://ftp.engcomp.ufrn.br/publicacoes/ECC/ECC9.pdf>. Acesso em 13 abr 2009.

COUGO, P. Modelagem conceitual e projeto de Banco de Dados. Rio de Janeiro: Campus, 1997.

DELFINO, S. R e NUNES, F. L. S Geração de Estudos de Caso para Treinamento Médico Virtual a partir de Técnicas de Realidade Virtual e Processamento de Imagens, 2009. Disponível em www.sbc.org.br/bibliotecadigital/download.php?paper=1481. Acesso em 12 ago 2009.

DEMERS, F. N. e MALENFANT, J. Reflection in logic, functional and object-oriented programming: a short comparative study. In: IJCAI'95, Workshop on Reflection and Metalevel Architectures and their Applications in AI. Montreal, Quebec, Canadá: [s.n.], p. 29–38, 1995.

DEVEGILI, A. J. Tutorial sobre Reflexão em Orientação a Objetos. 2000 Florianópolis: Universidade Federal de Santa Catarina, 2000, Disponível em: <http://devegili.org/docs/reflexaooo.html>. Acesso em 30 nov 2008.

FAYAD, M. E e JOHNSON, R. E. Domain-specific application frameworks: frameworks experience by industry. New York: J. Wiley, 2000. p.681 .

FERNANDES, Acauan P. Reflexão Computacional. Revista do CCEI v. 7, n. 11, p. 38-42, 2003.

FERREIRA, S. P. Produção e disponibilização de estatísticas: uma abordagem institucional. São Paulo Perspec., São Paulo, v. 17, n. 3-4, p. 17-25, 2003 .

GERALDELI, F. E. et al. Produção de material instrucional para o ensino da radiologia por meio da digitalização de imagens. Radiol Bras , São Paulo, v. 35, n. 1, p. 27-30, 2002 .

GOMES, A. T. A e SOARES, L. F. G. Gerência de Reconfiguração em Frameworks de Componentes. Departamento de Informática. PUC-Rio, 2005. Disponível em: http://www.lncc.br/~atagomes/files/2005_10_wdbc_gomes.pdf. Acesso em 13 abr 2009.

GOMES, C. S.; JIN, N. K e CREPALDI, T. F. Máquinas Virtuais Java e .NET. Instituto de Computação. Universidade Estadual de Campinas-UNICAMP. Disponível em: http://www.ic.unicamp.br/~rodolfo/Cursos/mc722/2s2007/trabalhos/g04_texto .pdf. Acesso em 30 nov 2008.

HEUSER, C. A. Projeto de Banco de Dados. 2 ed. Porto Alegre: Sagra-Luzzato. 1999. 279 p.

HILL, D. Escolhendo a arquitetura da camada de apresentação correta. Microsoft Corporation. Agosto de 2004. Disponível em: <http://www.microsoft.com/brasil/msdn/Tecnologias/arquitetura/ChoosingPresentationLayer.mspx>. Acesso em 14 fev 2009.

ILHA, J. O. O Registro Clínico Computadorizado: Funções e Vantagens. Revista Informédica, v. 1, n. 7, p. 5-10, 1994.

ILHA, J. O. O Registro Clínico Computadorizado: Padronização e Codificação. Revista Informédica, v. 1, n. 5, p. 5-8, 1993.

JOHANSTON, H. Sistemas de Informação Hospitalar: Presente e Futuro. Revista Informédica, v. 1, n. 2, p. 5-9, 1993.

KORTH, H. F., SILBERCHATZ, A. e SUDARSHAN, S. Sistemas de Bancos de Dados. 3 ed. São Paulo: Makron Books. 1999. 778 p.

LIMA, L. Attribute Banco Reflection. Parte 02. out. 2006. Disponível em: http://imasters.uol.com.br/artigo/4946/bancodedados/attribute_banco_reflection_-_parte_02/imprimir/. Acesso em 10 mai 2009.

MACHADO, F. N. R. Banco de Dados: projeto e implementação. São Paulo: Érica, 2004. 398 p.

MACORATTI, J. C. ADO.NET 2.0. Factories de Provedores, 2008. Disponível em: http://www.macoratti.net/adn_fp1.htm. Acesso em 01 abr 2009.

MACORATTI, J. C. Camada de dados e TableAdapters em Windows Forms, 2005. Disponível em: http://www.macoratti.net/vbn5_cdt.htm. Acesso em 18 fev 2009.

MAES, P. Concepts and experiments in computational reflection. In: Object-oriented programming systems, languages and applications - OOPSLA. Orlando, FL: [s.n.], 1987. p. 147–155.

MICROSOFT. Reflection .NET Framework Class Library Reference. Disponível em: [http://msdn2.microsoft.com/pt-br/library/system.reflection.emit\(VS.80\).aspx](http://msdn2.microsoft.com/pt-br/library/system.reflection.emit(VS.80).aspx), 2007. Acesso em: 22 abr 2009.

MSDN. Microsoft Developer Network. Práticas de programação <http://msdn2.microsoft.com/pt-br/library>. Acesso em 10 ago 2008.

MULLICK, R. et al. Plug-ins: A software model for biomedical imaging & visualization research. In: Biomedical Imaging Symposium: Visualizing the Future of Biology and

Medicine, National Institutes of Health (NIH). [S.l.: s.n.], 1999.

PAPAJORGJI, P. A. Plug and play approach for developing environmental models. *Environmental Modelling & Software*, v. 20, n. 10, p. 1353–1357, 2005.

PINTO, S. C. C. S. Composição em WebFrameworks. Tese [doutorado]. Departamento de Informática PUC-Rio, 2000.

PIRES, S. R; MEDEIROS, R. B e SCHIABEL H. Banco de imagens mamográficas para treinamento na interpretação de imagens digitais. *Revista Radiol Bras* v.37, p. 239-244, 2004.

PRATI, R. C; GEROMINI, M. R e MONARD, M. C. Um Framework Baseado em Patterns para Integração de Componentes de Mineração de Dados. Laboratório de Inteligência Computacional. Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo, 2003. 10 p. Disponível em: <http://www.icmc.usp.br/~mcmonard/public/ataiRM2003.pdf>. Acesso em 13 abr 2009.

PREE, W. Design Patterns for Object-Oriented Software Development, Addison-Wesley, 1995.

RIVIERES, J.D; WIEGAND, J. Eclipse: A plataforma for integrating development tools. *IBM Systems Jornal*, v. 43, n. 2, p. 371–383, 2004.

SÁ, Â. A. R. de. Uma Proposta de Sistema Computacional em Tempo Real para Biofeedback. Dissertação (Mestrado) — Universidade Federal de Uberlândia, Maio 2004.

SABBATINI, R. M. E. O Computador no Processamento de Sinais Biológicos. *Revista Informédica*, v. 2, n. 12, p. 5-9, 1995.

SABBATINI, R. M. E. e ORTIZ, J. M. Um Sistema para a Informatização Integrada de Clínicas. *Revista Informédica*, v. 2, n. 10, p. 5-12, 1994.

SANCHES, A. R. Fundamentos de Armazenamento e Manipulação de Dados, 2005. Disponível em: <http://www.ime.usp.br>. Acesso em 12 dez 2008.

SERRATO, F. Arquitetura.Net. Camada de componentes de acesso a dados. 2005. Disponível em: http://imasters.uol.com.br/artigo/3763/dotnet/arquitetura_net_-_camada_de_componentes_de_acesso_a_dados/. Acesso em 18 fev 2009.

SILVA, E. Q. Um Framework baseado em componentes para o desenvolvimento de aplicações Web e um processo de instanciação associado. 2006. 167 p. Tese (Doutorado em Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, São Carlos, São Paulo. Disponível em: <<http://www.teses.usp.br/teses/disponiveis/55/55134/tde-26092006-15265>>2/ Data de Acesso: 13 abr 2009.

SILVA, R. Bancos de Dados Geográficos: Uma Análise das Arquiteturas Dual (Sprin G) e Integrada (Oracle Spatial). Dissertação Mestrado [Engenharia] Escola Politécnica da Universidade de São Paulo. São Paulo, 2002. 139 p.

STENSAAS, S. S e SORENSON, D. K. “HyperBrain” and “Slice of Life”: An Interactive Hypercard and Videodisc, PubMed p. 416-420, 1988.

SZYPERSKI, C. Component software: beyond object-oriented programming. 2. ed. Reading MA, EUA: Addison-Wesley. 2002. 589 p.

SZYPERSKI, C.. Independently extensible systems—software engineering potential and challenges. Proceedings of the 19th Australian Computer Science Conference, Melbourne, Austrália. 1997. p. 19–32.

UNICAMP. Framework Baseado em Componentes, 2009. Disponível em: <http://www.dca.fee.unicamp.br/projects/sapiens/Reports/ExplorAnot2/componentFramework.htm>. Acesso em 13 abr 2009.

VOLPE, R. M. e SABBATINI, R. M. E. Aplicações da Multimídia no Ensino Médico. Revista Informédica, v. 2, n. 9, p. 5-12, 1994.

WATSON, R. T. Data Management. Banco de Dados e Organizações. 3 ed. Rio de Janeiro: LTC, 2004. 370 p.

ZAIDAN, F. Ferramenta de modelagem conceitual de banco de dados, 2003. Disponível em www.fernandozaidan.com.br/ined/bd/outros/brModelo.pdf. Acesso em 12 dez 2008.