FRAMEWORK PARA GERENCIAR DADOS DE INTERAÇÃO DO USUÁRIO EM AMBIENTES HIPERMÍDIA DE APRENDIZAGEM

Universidade Federal de Santa Catarina
Programa de Pós-graduação em
Engenharia de Produção

FRAMEWORK PARA GERENCIAR DADOS DE INTERAÇÃO DO USUÁRIO EM AMBIENTES HIPERMÍDIA DE APRENDIZAGEM

Cláudio Luiz Ferreira

Dissertação apresentada ao
Programa de Pós-Graduação em
Engenharia de Produção da
Universidade Federal de Santa Catarina
como requisito parcial para a obtenção
do título de mestre em
Engenharia de Produção,
(Área de Concentração: Mídia e Conhecimento)

Florianópolis
2003

Cláudio Luiz Ferreira

FRAMEWORK PARA GERENCIAR DADOS DE INTERAÇÃO DO USUÁRIO EM AMBIENTES HIPERMÍDIA DE APRENDIZAGEM

Esta dissertação foi julgada e aprovada para a obtenção do título de **Mestre em Engenharia de Produção** no **Programa de Pós-Graduação em Engenharia de Produção** da Universidade Federal de Santa Catarina

Florianópolis, 16 de julho de 2003.

_____

Prof. Edson Pacheco Paladini, Dr.

Coordenador

BANCA EXAMINADORA

Profa. Vania Ribas Ulbricht, Dr.
Orientadora

Prof. Luiz Fernando Gonçalves de Figueiredo, Dr.

Prof. João Bosco da Mota Alves, Dr.

À minha família, pelo apoio
incondicional e as lições de
honestidade e perseverança.

AGRADECIMENTOS

Aos meus pais, pelo exemplo de vida, dedicação e apoio incondicional.

Aos meus irmãos, pela abertura de horizontes e tantas outras formas em que me ajudaram.

À minha orientadora, professora Dra. Vania Ribas Ulbricht, pela amizade e as valiosas contribuições para a conclusão deste trabalho.

Ao Conselho Nacional de Pesquisa (CNPq).

Ao prof. Dr. Luiz Fernando Gonçalves de Figueiredo, pelas contribuições práticas e teóricas do dia-a-dia.

Ao amigo Daniel Wyllie Lacerda Rodrigues, que colaborou direta e indiretamente em diversos aspectos desta dissertação.

Aos amigos de todas as horas, em especial aos colegas da Pós-Graduação e do HiperLab (Laboratório de Ambientes Hipermídia para Aprendizagem).

Aos abnegados autores e inventores, que inspiraram e tornaram este trabalho possível.

À Universidade Federal de Santa Catarina, por todas as oportunidades que me proporcionou.

A todos que contribuíram de forma direta ou indireta para a realização desse trabalho.

“Estou sempre fazendo algo que não sei fazer, para que eu possa aprender a fazê-lo.”

Pablo Picasso

“Jamais desencoraje alguém que faz progressos contínuos, não importa o quanto sejam lentos.”

Platão

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

1FN	Primeira Forma Normal
2FN	Segunda Forma Normal
3FN	Terceira Forma Normal
ADO	ActiveX *Data Objects*
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
ASP	*Active Server Pages*
BNF	*Backus Naur Form*
CAE	*Computer Aided Education*
CASE	*Computer Aided Software Engineering*
CE	Chave Estrangeira
CERN	*Conseil Européen pour la Recherche Nucléaire*
CGI	*Common Gateway Interface*
CNPq	Conselho Nacional de Pesquisa
COM	*Component Object Model*
CORBA	*Common Object Request Broker Architecture*
CP	Chave Primária
CSS	*Cascade StyleSheet*
DARPA	*Defense Advanced Research Project Agency*
DCOM	*Distributed Component Object Model*
DOM	*Document Object Model*
DTD	*Document Type Definition*
DW	*Data Warehouse*
EaD	Ensino a Distância
EGR	Departamento de Expressão Gráfica
EIAC	Ensino Inteligente Auxiliado por Computador
ER	Entidade-Relacionamento
FNBC	Forma Normal de Boyce-Codd
FTP	*File Transfer Protocol*
GUI	*Graphics User Interface*
HTML	*HyperText Markup Language*

HTTP	*HyperText Transfer Protocol*
IBM	*International Business Machines*
IDMS	I*ntegrated Data Management System*
IDS	*Integrated Data Store*
IHC	Interface Homem-Computador
IIS	*Internet Information Server*
IMS	*Information Management System*
ISAPI	*Internet Server Aplication Programming Interface*
ISO	*International Standards Organization*
JSP	*Java Server Pages*
ODBC	*Open Database Conectivity*
OLAP	O*n-Line Analytical Processing*
OLE	*Object Linking and Embedding*
OLE DB	*Object Linking and Embedding for DataBases*
OTM	*Object Transaction Monitor*
PC	*Personal Computer*
PHP	*Personal Home Page*
RISC	*Reduced Instruction-Set Computer*
SEQUEL	*Structured English Query Language*
SGBD	Sistema de Gerenciamento de Banco de Dados
SGBDR	Sistema de Gerenciamento de Banco de Dados Relacional
SGML	*Standard Generalized Markup Language*
SQL	*Structured Query Language*
UDA	*Universal Data Access*
UDESC	Universidade do Estado de Santa Catarina
UFSC	Universidade Federal de Santa Catarina
URL	*Universal Resource Locator*
WWW	*World Wide Web*
XML	*eXtensible Markup Language*
XSL	*XML Stylesheet Language*

RESUMO

FERREIRA, Cláudio Luiz. **FRAMEWORK PARA GERENCIAR DADOS DE INTERAÇÃO DO USUÁRIO EM AMBIENTES HIPERMÍDIA DE APRENDIZAGEM.** Florianópolis, 2003. 130f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-graduação em Engenharia de Produção, UFSC, 2003.

Este trabalho propõe um *framework*, ou seja, uma infra-estrutura reutilizável de *software* para gerenciar dados de interação de usuários em ambientes hipermídia de aprendizagem. Apresenta-se uma arquitetura de três camadas para o monitoramento das interações do usuário bem como seu armazenamento e recuperação em banco de dados relacional. Os resultados desse trabalho servem de base para a realização de diferentes pesquisas visando o aprimoramento dos ambientes de EIAC, especificamente as hipermídias pedagógicas.

Palavras-chave: hipermídia, banco de dados, *framework*.

ABSTRACT

FERREIRA, Cláudio Luiz. **FRAMEWORK PARA GERENCIAR DADOS DE INTERAÇÃO DO USUÁRIO EM AMBIENTES HIPERMÍDIA DE APRENDIZAGEM.** Florianópolis, 2003. 130f. Dissertação (Mestrado em Engenharia de Produção) – Programa de Pós-graduação em Engenharia de Produção, UFSC, 2003.

This work proposes a framework for user interaction data management in hypermedia learning environments. A three-tier software architecture is presented for user interaction monitoring as well as for data storage and retrieval. The outcomes of this work generate a software foundation for different research issues, aiming at the improvement of ICAI systems, notably the educational hypermedia applications.

Key words: hypermedia, database, framework.

1. INTRODUÇÃO

1.1 Considerações iniciais

Atualmente, as instituições acadêmicas estão em transição. A maior parte das mudanças se deve a pressões econômicas geradas pelos custos escalantes e a demanda do mundo comercial por pessoas capazes de se comportarem adequadamente na "sociedade do conhecimento" (Palof & Pratt, 1999).

> "O termo 'nova economia' refere-se a um conjunto de mudanças qualitativas e quantitativas, que nos últimos 15 anos transformaram a estrutura, o modo de funcionamento e as regras do mundo. No lugar dos recursos materiais, as idéias e o conhecimento passaram e reger a nova economia."
>
> (HODGINS, 2000, p. 3)

A nova economia, associada às vertiginosas evoluções tecnológicas dos últimos anos, gera um quadro de profundas transformações sociais que alteram o papel dos aprendizes e profissionais das mais diversas áreas. Nesta nova conjuntura, o "aprendizado torna-se parte do trabalho e o trabalho torna-se parte do aprendizado" (Hodgins, 2000, p. 16).

Este cenário de mudanças gera uma demanda por conhecimento e para o que Valente (1993) definiu como "educar para a informática" e "educar pela informática". Ulbricht (1997) afirma que o uso de computadores na educação representa uma transformação no modo de pensar e educar e implica na redução dos custos da educação, viabilizando sua democratização.

No sentido de democratizar o conhecimento, merecem destaque as aplicações educacionais em redes de computadores. Biuk-Aghai (1998) afirma que nos últimos anos, a utilização cada vez mais freqüente da Internet como uma rede financeiramente acessível levou ao desenvolvimento de vários sistemas educacionais à distância para a rede. Em particular, a WWW tem sido a escolha

favorita para funcionar como base para o desenvolvimento de sistemas educacionais.

No entanto, a maioria das aplicações de aprendizagem para WWW não passa de uma rede de páginas estáticas de hipertexto. O desafio é o desenvolvimento de aplicações educacionais avançadas, com um bom grau de interatividade e adaptatividade (Brusilovsky, 1998).

Essas aplicações educacionais avançadas são chamadas de sistemas de Ensino Inteligente Assistido por Computador (EIAC). Segundo Ulbricht (1997), os EIAC´s possibilitam melhorias na capacidade do aluno para a solução de problemas; na otimização do seu processo de aprendizagem; na democratização do ensino; na motivação do estudante; na redução de custos e barreiras geográficas quando conectados a uma rede de comunicações; na supressão da hora e lugar de estudo; na redução do tempo de estudo e na qualidade do material instrucional a ser apresentado.

Uma classe particular de sistema de EIAC é a das hipermídias pedagógicas. Este trabalho propõe um *framework*, ou seja, uma infra-estrutura de *software* para gerenciar dados de interação de usuários em ambientes hipermídia de aprendizagem. Apresenta-se uma arquitetura de três camadas para o monitoramento das interações do usuário bem como seu armazenamento e recuperação em banco de dados relacional.

Os resultados desse trabalho servem de base para a realização de diferentes pesquisas visando o aprimoramento dos ambientes de EIAC, especificamente as hipermídias pedagógicas.

1.2 Origem do trabalho

O presente trabalho teve origem nos resultados apontados na tese de título “Modelagem de um Ambiente Hipermídia de Construção do Conhecimento em

Geometria Descritiva". Trata-se da tese de doutorado da Profa. Dra. Vânia Ribas Ulbricht, orientadora desse trabalho.

Ulbricht (1997) afirma que um ambiente hipermídia para a construção do conhecimento deve manter registro das variáveis de interação do usuário com o ambiente hipermídia de aprendizagem, com o objetivo de tratar as diferenças individuais dos aprendizes através de uma análise detalhada dessas variáveis.

Partindo dessa premissa, passam a ser necessários à modelagem e o desenvolvimento de uma infra-estrutura de *software*, capaz de armazenar e recuperar essas variáveis de interação do usuário em ambientes hipermídia de aprendizagem. Projetar e implementar essa infra-estrutura é o objetivo principal deste trabalho.

1.3 Justificativa e relevância

A motivação inicial para este trabalho foi o desenvolvimento da infra-estrutura de *software* para o ambiente hipermídia de aprendizagem Geometrando.

> "Geometrando – Caminhando no Tempo com a Geometria é o título de um *software* educacional em desenvolvimento na UFSC – Universidade Federal de Santa Catarina, com a participação da UDESC – Universidade do Estado de Santa Catarina, fruto de um projeto de pesquisa em Informática na Educação, amparado pelo programa CNPq/PROTEM."
>
> (ULBRICHT et al., 2001, p. 19)

O financiamento desse projeto pelo CNPq através do programa PROTEM permitiu a criação de um núcleo de pesquisas, com uma boa quantidade de colaboradores de graduação e pós-graduação. O Geometrando possibilitou também, a continuidade das pesquisas em ambientes hipermídia de ensino-aprendizagem, coordenadas pela professora Vania Ribas Ulbricht.

A participação do autor neste projeto foi a implementação da parte do *software* responsável pelo monitoramento das variáveis de interação do usuário com o ambiente, bem como seu armazenamento em banco de dados relacional, viabilizando a recuperação dessas informações para diversos fins.

Apesar de ter sido concebido para o Geometrando, este trabalho gera resultados que podem ser aplicados em outros ambientes de ensino-aprendizagem voltados para a *web*, sendo relevantes também para os desenvolvimentos na área de ensino à distância.

A disponibilização dessa infra-estrutura de *software* serve como base para a realização de diferentes pesquisas com o objetivo de sofisticar os ambientes de EIAC, particularmente as hipermídias pedagógicas. Em longo prazo, tais pesquisas deverão culminar com a criação de um modelo de desenvolvimento de hipermídias de aprendizagem que levam em consideração aspectos individuais do aprendiz e facilitam a construção do conhecimento.

1.4 Objetivos

1.4.1 Objetivo geral

- Conceber e implementar um *framework* para armazenagem e recuperação de variáveis de interação do usuário em ambientes hipermídia de aprendizagem.

1.4.2 Objetivos específicos

- Efetuar uma revisão bibliográfica sobre banco de dados, tecnologias de rede e arquitetura de sistemas computacionais.
- Conceber a arquitetura e implementar o *framework* para armazenamento e recuperação das variáveis de interação do usuário com ambientes hipermídia de ensino-aprendizagem para rede.

- Modelar e implementar o banco de dados para armazenamento e recuperação das variáveis de usuário.
- Garantir a persistência das variáveis do usuário com o ambiente de ensino aprendizagem.
- Sugerir pesquisas que podem ser realizadas utilizando este *framework* como base.

1.5 Procedimentos metodológicos

Uma pesquisa pode ser classificada como descritiva, segundo Costa (2001, p.31), quando o pesquisador decide realizar uma pesquisa e:

- já conhece algo sobre o assunto de interesse;
- quer divulgar o que já conhece e buscar, de certo modo, adesões de outros pesquisadores;
- busca também críticas construtivas que lhe possibilitem ampliar o conhecimento sobre o tema.

Este trabalho caracteriza-se, portanto, como uma pesquisa descritiva e aplicada, pois seus resultados foram imediatamente utilizados para o desenvolvimento do *framework* que permite o gerenciamento das informações de interação do usuário em ambientes hipermídia de aprendizagem, particularmente o ambiente Geometrando (Ander-Egg, Rummel *apud* Lakatos & Marconi, 1996).

1.6 Descrição e organização dos capítulos

Este trabalho divide-se em cinco capítulos, conforme descrição a seguir:

1. Introdução: este capítulo apresenta uma introdução resumida do tema abordado, bem como seus objetivos geral e específicos, a origem, justificativa e relevância

do trabalho, além de uma breve apresentação da metodologia empregada e a estrutura do trabalho.

2. Banco de Dados: este capítulo de revisão bibliográfica apresenta um histórico e diversos conceitos de banco de dados, dando ênfase aos bancos de dados relacionais e a abordagem entidade-relacionamento.
3. Tecnologias de Rede: consiste na apresentação de um histórico e conceituação das redes de computadores e tecnologias recentes da Internet, em especial a *World Wide Web*. Apresenta-se também uma breve revisão a respeito da arquitetura dos sistemas computacionais.
4. *Framework*: apresenta diversas questões de projeto e implementação do *framework* para gerenciar dados de interação do usuário em ambientes hipermídia de aprendizagem.
5. Conclusão: este capítulo apresenta as conclusões e sugestões para trabalhos futuros.

São apresentadas ainda, as referências bibliográficas utilizadas no corpo deste trabalho e por fim, o anexo com o código SQL para geração do banco de dados criado para o *framework*.

2. BANCO DE DADOS

2.1 Histórico

A modelagem e projeto de banco de dados sofreram evoluções significativas nos últimos anos desde que o modelo relacional passou a dominar a maioria das aplicações comerciais de banco de dados. Entretanto, antes do advento do banco de dados relacional, predominavam os modelos hierárquico e de redes[1].

Segundo Hayes (2002), o pesquisador Charles Bachman da *General Eletric Co*. desenvolveu o primeiro sistema de gerenciamento de banco de dados funcional em 1961. O sistema criado por Bachman chamava-se *Integrated Data Store* (IDS) - Armazenamento Integrado de Dados - e possuía esquemas de dados e *logging*[2]. Entretanto, o IDS funcionava apenas nos computadores de grande porte da *General Eletric*, permitia somente um único arquivo no banco de dados e todas as tabelas de dados tinham de ser programadas uma-a-uma. Um cliente do IDS, a *BF Goodrich Chemical Co*. chegou a reescrever todo o sistema para torná-lo viável. O resultado foi chamado de I*ntegrated Data Management System* (IDMS) - Sistema Integrado de Gerenciamento de Dados.

Em 1968, a IBM lançou o *Information Management System* (IMS) – Sistema de Gerenciamento de Informações – um sistema de banco de dados hierárquico para os computadores de grande porte da IBM. Segundo Korth & Silberschatz (1995), esta primeira versão do IMS foi desenvolvida pela IBM e pela *North American Aviation* (*Rockwell International)* para o programa de aterrissagem lunar Apollo.

---

[1] Korth & Silberschatz (1995) explicam que no modelo de redes os dados são representados por coleções de registros e os relacionamentos entre os dados são representados por ligações que podem ser vistas como ponteiros. Similarmente, no modelo hierárquico os dados e relacionamentos também são representados por registros e ligações. Contudo, no modelo hierárquico os registros são organizados como coleções de árvores em vez dos grafos arbitrários do modelo de redes.

[2] Logging é uma técnica de gravar as operações efetuadas sobre os dados bem como sua data, hora e usuário responsável. Este método é importante para recuperar a consistência dos dados em caso de falha e para manter informações sobre as atualizações feitas no banco de dados.

Já em 1973, a *Cullinane Corp*. (mais tarde seria conhecida como *Cullinet Software Inc*.) iniciou as vendas de uma versão bastante melhorada do IDMS e estava a caminho de se tornar a maior empresa de *software* do mundo naquela época.

Nesse ínterim, o pesquisador da IBM Edgar F. Codd estava procurando um modo melhor de organizar os bancos de dados. Em 1969, Codd introduziu o conceito de um banco de dados relacional inteiramente organizado em tabelas simples.

> “Codd criou um modelo que permitia aos projetistas quebrar seus bancos de dados em tabelas separadas, porém relacionadas de modo a otimizar a eficiência de execução enquanto mantinha a mesma aparência externa do banco de dados original para os usuários finais. Desde então, Codd é considerado o pai do banco de dados relacional.”
>
> (LOSHIN, 2001)

Figura 1: Em 1969 o pesquisador Edgar F. "Ted" Codd inventa o banco de dados relacional
Fonte: Hayes (2002)

Hayes (2002) afirma que naquele momento, a IBM colocou mais pesquisadores no projeto com o codinome *System/R* nos laboratórios em San José na Califórnia. Apesar desse investimento, o comprometimento da IBM com o seu sistema hierárquico de banco de dados (IMS) impediu que o *System/R* se tornasse um produto antes de 1980.

Korth & Silberschatz (1995) afirmam que aproximadamente ao mesmo tempo em que o Laboratório de Pesquisa San Jose da IBM desenvolvia o protótipo do *System/R*, um grupo na Universidade da Califórnia desenvolvia um sistema de banco de dados experimental chamado Ingres.

> "Em 1973 os pesquisadores de *Berkeley* Michael Stonebraker e Eugene Wong usaram informações publicadas a respeito do *System/R* para iniciar os trabalhos de desenvolvimento do Ingres. O projeto Ingres acabaria sendo comercializado pela *Oracle Corp.*, *Ingres Corp.* e outras empresas do Vale do Silício. Em 1976, a *Honeywell Inc.* começou a vender o *Multics Relational Data Store*, o primeiro banco de dados relacional comercial."
>
> (HAYES, 2002)

Desde o lançamento do *System/R* da IBM em 1980, os sistemas de banco de dados relacionais vêm dominando o segmento de gerenciamento de dados. Nos dias de hoje há diversos produtos comerciais de bancos de dados relacionais tais como IBM DB2, Oracle, Informix, Microsoft Access, Microsoft SQL Server, Borland Interbase, MySQL, PostGreSQL sendo os últimos três disponíveis gratuitamente.

Chung (1995) confirma que atualmente a tecnologia de banco de dados relacionais atingiu um estado maduro e possui diversos produtos consagrados no mercado. A maioria das aplicações comerciais e industriais da atualidade utiliza sistemas de gerenciamento de banco de dados relacionais (SGBDR) e há uma quantidade muito grande de dados armazenados nesses sistemas.

2.2 Sistema de Gerenciamento de Banco de Dados - SGBD

Para Teorey (1999), um sistema gerenciador de banco de dados (SGBD) é um s*oftware* genérico para manipulação de banco de dados. Um SGBD é capaz de tratar os aspectos de visão lógica e física dos dados, suportar linguagens para definição e manipulação de dados, bem como prover utilitários para o gerenciamento de transações e controle de concorrência. Leite (1980) resume os requisitos fundamentais que um SGBD deve cumprir:

- Permitir a independência de dados;
- controlar a redundância de dados;

- oferecer estruturas de dados compatíveis com as características dos diversos tipos de usuários;
- permitir o acesso concorrente;
- oferecer facilidades que permitam estabelecer o controle de acesso;
- permitir e manter o relacionamento entre os dados;
- garantir a integridade das informações armazenadas;
- ter bom desempenho.

Segundo Korth & Silberschatz (1995), o grande objetivo de um Sistema de Gerenciamento de Banco de Dados é prover os usuários com uma visão abstrata dos dados. Similarmente aos conceitos de abstração e encapsulamento em sistemas orientados a objeto, cabe ao SGBD omitir certos detalhes de como os dados são armazenados e mantidos para esconder a complexidade das estruturas de dados utilizadas internamente pelo sistema.

Divide-se em níveis crescentes de abstração toda a complexidade de um sistema de banco de dados. O físico é o nível mais baixo de abstração no qual descreve-se detalhadamente as estruturas de dados que armazenam os dados fisicamente. No nível conceitual abstrai-se as complexidades do nível físico e determina-se apenas quais dados são armazenados no banco e quais as relações entre eles. Por último, no nível de visões, parte-se do princípio que os usuários de um SGBD não estão interessados em todos os dados armazenados e, portanto descreve-se apenas parte dos dados estruturados no nível conceitual.

2.3 Banco de dados relacional

O modelo relacional de banco de dados foi definido inicialmente pelo Dr. E. F. Codd na década de 1970 e posteriormente bastante estendido por outros autores. Ele pode ser definido por pelo menos três características significativas (Codd, 1982):

- Suas estruturas de dados são simples. Tratam-se de relações que são tabelas de duas dimensões cujos elementos são os itens de dados. Essa característica

permite um alto grau de independência da representação física dos dados (e.g. arquivos e índices);

- Provê uma fundação sólida para a consistência de dados. O projeto de banco de dados é auxiliado pelo processo de normalização[3] que elimina anomalias de dados. O estado consistente do banco de dados pode ser definido e mantido através de regras de integridade;
- Permite a manipulação das relações sob a forma de conjuntos. Esta característica levou ao desenvolvimento de poderosas linguagens não-procedurais baseadas na teoria de conjuntos (álgebra relacional) ou na lógica (cálculo relacional).

2.4 Ciclo de vida do banco de dados

Para Teorey (1999), o ciclo de vida de um banco de dados incorpora as etapas básicas de projeto do esquema lógico global do banco de dados, alocação dos dados em rede e definição de esquemas locais específicos ao SGBD utilizado. Depois da fase de projeto, o ciclo continua com a implementação e manutenção do banco de dados.

2.4.1 Análise dos Requisitos

Os requisitos do banco de dados são determinados através de entrevistas com as pessoas que geram os dados e seus usuários. Posteriormente, elabora-se uma especificação formal dos requisitos que inclui os dados necessários, as relações naturais entre esses dados e a plataforma de *software* para implementação do banco de dados.

2.4.2 Projeto Lógico – abstração

O esquema global do banco de dados apresenta todos os dados e suas relações e é desenvolvido usando uma técnica de modelagem conceitual de dados tal como o

[3] O processo de normalização será visto em detalhe no item 2.6.

modelo Entidade-Relacionamento[4]. O projeto lógico inicia portanto com a modelagem Entidade-Relacionamento (ER), passa pelo processo de transformação do modelo ER para tabelas SQL[5] e finalmente pelo processo de normalização das tabelas.

2.4.3 Projeto Físico

Uma vez concluído o projeto lógico é necessário gerar uma estrutura física para o banco de dados com o propósito de aumentar sua eficiência. O projeto físico envolve a definição dos índices e *clustering* (agrupamento) dos dados. Teorey (1999) chama atenção para outra fase importante do projeto físico – a denormalização. Este processo pode ser utilizado quando há ganhos significativos em eficiência ao efetuar-se refinamentos no esquema global para cumprir requisitos de processamento de consultas e transações.

2.4.4 Implementação, monitoramento e modificação

Terminado o projeto, inicia-se a fase de implementação do banco de dados com a criação do esquema formal através da linguagem de definição de dados do SGBD. A partir daí a linguagem de manipulação de dados pode ser usada para consultar e atualizar o banco de dados bem como definir índices e estabelecer restrições tais como a integridade referencial[6].

A linguagem SQL possui tanto as construções de definição quanto as de manipulação de dados. Desse modo é possível criar, consultar e atualizar uma tabela em SQL.

---

[4] O modelo ER é visto em detalhe no item 2.5.

[5] *Structured Query Language* (SQL) – Linguagem de Consultas Estruturadas – é a linguagem padrão ISO-ANSI para definição e manipulação de dados em sistemas de bancos de dados relacionais. No item 2.5.8 descreve-se em maior detalhe esta linguagem.

[6] No item 2.5.6 define-se as restrições de integridade referencial.

Quando o banco de dados entra em operação, o monitoramento indica se os requisitos de eficiência estão sendo cumpridos. Se eles não estão sendo satisfeitos, deve-se fazer modificações para melhorar a eficiência. Outras modificações podem ser necessárias quando os requisitos mudam ou a expectativa do usuário final aumenta com a eficiência do banco de dados. Conseqüentemente, o ciclo de vida continua com o monitoramento, reprojeto e a implementação das modificações.

2.5 Modelo entidade-relacionamento

> "O conhecimento de técnicas para modelagem de dados e projeto de bancos de dados é fundamental para os criadores de banco de dados. Dentre uma variedade de técnicas de modelagem de dados, o modelo Entidade-Relacionamento (ER) é atualmente o mais popular em uso devido à sua simplicidade e legibilidade. Na maioria das ferramentas CASE[7] utiliza-se uma forma simplificada do modelo ER e a técnica é de fácil aprendizado além de ser aplicável a uma variedade de aplicações industriais e comerciais."
>
> (TEOREY, 1999, p.10)

A abordagem entidade-relacionamento (ER) para a modelagem conceitual de banco de dados foi descrita pela primeira vez em 1976, por Peter Chen e posteriormente estendida por outros autores. O autor Toby J. Teorey juntamente com outros pesquisadores, é responsável pela metodologia de projeto lógico para banco de dados relacionais usando o modelo ER estendido e suas publicações são referência no assunto. Por esse motivo, a descrição do modelo ER neste trabalho é baseada principalmente nos textos de Teorey.

---

[7] Para LENDING & CHERVANY, as tecnologias de *Computer-Aided Software Engineering* (CASE) - Engenharia de software auxiliada por computador – são ferramentas que provém auxílio automatizado ao desenvolvimento de *software*. O objetivo dessas ferramentas é a redução do tempo e custo de desenvolvimento de *software* bem como o aprimoramento da qualidade dos sistemas desenvolvidos.

Para Korth & Silberschatz (1995), o modelo entidade-relacionamento é baseado na percepção do mundo real que consiste em um conjunto de objetos básicos chamados *entidades* e nos *relacionamentos* entre esses objetos.

2.5.1 Entidades

Para Teorey (1999), as entidades são os principais elementos de dados sobre os quais deve-se manter informações. Korth & Silberschatz (1995) definem como entidade um objeto que existe e é distinguível dos outros objetos.

Para exemplificar, “Cláudio Luiz Ferreira” com o número de matrícula 20012R0120 é uma entidade, visto que este número identifica unicamente um aluno em particular de uma universidade. Do mesmo modo, a disciplina com o código “EPS5001” do Departamento de Engenharia de Produção e Sistemas é uma entidade que identifica unicamente uma disciplina específica.

2.5.2 Relacionamentos

Os relacionamentos representam associações do mundo real entre uma ou mais entidades. Por exemplo, pode-se definir um relacionamento que associe o aluno “Cláudio Luiz Ferreira” à disciplina “EPS5001”.

> “Os relacionamentos são descritos em termos de grau, conectividade e existência. O significado mais comum associado ao termo relacionamento é quanto à conectividade entre as ocorrências das entidades: um-para-um, um-para-muitos e muitos-para-muitos”.
>
> (TEOREY, 1999, p.15)

2.5.3 Atributos

Os atributos são características das entidades e fornecem detalhes descritivos a seu respeito. Uma ocorrência específica de um atributo de uma entidade ou de um relacionamento é chamada de valor de atributo (Teorey, 1999). Por exemplo, os

atributos de uma entidade como aluno podem incluir o número de matrícula, a data de matrícula, o endereço eletrônico, o número de telefone etc.

É importante ressaltar que os atributos podem ser tanto de entidades quanto de relacionamentos. Um atributo de uma relação muitos-para-muitos como alunos-disciplinas pode ser a data de matrícula do aluno nesta disciplina ou o conceito obtido pelo aluno nesta disciplina.

Um valor de atributo pode ser indefinido. Essa falta de definição pode ter diversas interpretações sendo as mais comuns “valor desconhecido” e “não aplicável”. Esse valor de atributo especial é tipicamente chamado de “valor vazio”[8].

A representação do valor nulo deve ser diferente de qualquer outro valor possível no domínio e deve ser diferente do valor zero. A utilização de valores nulos é de fundamental importância para a resolução de consultas que associam a possível pertinência de um valor a um conjunto.

2.5.4 Exemplo simplificado de Banco de Dados

Para demonstrar os conceitos necessários à construção de um banco de dados relacional consistente é conveniente utilizar um exemplo prático. Tomando-se o domínio de uma universidade de forma simplificada como exemplo, as entidades a serem modeladas são os professores, departamentos, alunos, disciplinas etc. Deseja-se manter informações relevantes para cada uma dessas entidades e para tal são definidos os esquemas relacionais para quatro tabelas:

- PROFESSORES(PROF_COD, PROF_NOME, PROF_DATA_ADMISSAO)
- DEPARTAMENTOS(DEPTO_COD, DEPTO_SIGLA, DEPTO_NOME, DEPTO_CHEFE)
- ALUNOS(ALUNO_COD, ALUNO_NOME, ALUNO_DATA_NASC)
- DISCIPLINAS(DISC_COD, DISC_NOME)

---

[8] Em inglês utiliza-se a expressão *null values*. A tradução por “valor vazio” no lugar de “valor nulo” é usual, pois a palavra nulo implica em “zero” para valores numéricos e “branco” para valores alfanuméricos e portanto, difere da conotação de valor indefinido que é expressada por “valor vazio”.

Na tabela PROFESSORES mantém-se o código do professor, seu nome e data de admissão enquanto. Na tabela de DEPARTAMENTOS mantém-se o código, a sigla, o nome do departamento bem como o código do professor que é chefe desse departamento. Na tabela alunos armazena-se o código do aluno (tipicamente corresponde ao número de matrícula), seu nome e data de nascimento e finalmente na tabela DISCIPLINAS mantém-se o código e o nome da disciplina.

PROF_COD	**PROF_NOME**	**PROF_DATA_ADMISSAO**
PROF1	“Juliana T.”	10/05/1987
PROF2	“Adriano S.”	22/10/1979
...	...	...
PROF33	“Getúlio F.”	11/01/1985

Tabela I: Tabela relacional PROFESSORES

DEPTO_COD	**DEPTO_SIGLA**	**DEPTO_NOME**	**DEPTO_CHEFE**
DEPTO1	“EGR”	“Departamento de Expressão Gráfica”	PROF1
DEPTO2	“INE”	“Departamento de Informática e Estatística”	PROF20
...	...	...	...
DEPTO20	“EPS”	“Departamento de Engenharia de Produção e Sistemas”	PROF33

Tabela II: Tabela relacional DEPARTAMENTOS

ALUNO_COD	**ALUNO_NOME**	**ALUNO_DATA_NASC**
A1	“Cassandra S.”	01/03/1982
A2	“Eduardo S. S.”	27/08/1980
...	...	...
A200	“Ana Paula C.”	15/11/1983

Tabela III: Tabela relacional ALUNOS

DISC_COD	**DISC_NOME**
DISC1	“Banco de dados”
DISC2	“Data Warehouse”
...	...
DISC20	“Ergonomia Cognitiva”

Tabela IV: Tabela relacional DISCIPLINAS

As relações desse banco de dados podem ser vistas nas tabelas I, II, III e IV. As colunas de cada tabela correspondem aos atributos da relação e as linhas correspondem as tuplas. A primeira linha de cada tabela corresponde ao esquema da relação. Uma vez que a informação armazenada nas tabelas varia com o tempo, diversas instâncias podem ser geradas a partir de um esquema da relação. As tabelas I, II, III e IV são instâncias das relações PROFESSORES, DEPARTAMENTOS, ALUNOS e DISCIPLINAS respectivamente. O número de atributos de uma relação define o seu grau enquanto o número de tuplas define sua cardinalidade.

2.5.5 Chaves

> “Uma superchave é um conjunto de um ou mais atributos que, em conjunto, permitem identificar unicamente uma entidade. Uma chave candidata é qualquer subconjunto de atributos da superchave que também permitem identificar unicamente uma tabela e que não pode ser reduzido à outra superchave. A chave primária é selecionada dentre o conjunto das chaves candidatas de uma tabela para ser utilizada como índice para essa tabela.”
>
> (TEOREY, 1999, p. 99)

O conceito de chave é fundamental em bancos de dados relacionais. No exemplo do item 2.5.4, as chaves primárias das relações PROFESSORES, DEPARTAMENTOS, ALUNOS e DISCIPLINAS são respectivamente (PROF_COD), (DEPTO_COD), (ALUNO_COD) e (DISC_COD).

Uma chave estrangeira é um atributo de uma tabela que se refere a uma chave primária em outra tabela. No exemplo do item 2.5.4, o atributo DEPTO_CHEFE na tabela DEPARTAMENTOS que armazena um código de professor é uma chave estrangeira que referencia a tabela PROFESSORES. Esta chave estrangeira configura o relacionamento entre as duas tabelas que representa a chefia de departamento.

2.5.6 Restrições

Para Korth & Silberschatz (1995) um esquema ER pode definir certas restrições com as quais o conteúdo do banco de dados tem de estar de acordo. Uma restrição importante é quanto à conectividade dos relacionamentos, que expressa o número de entidades às quais outra entidade pode ser associada (um-para-um, um-para-muitos, muitos-para-muitos).

Outra forma importante de restrições é a dependência de existência. Teorey (1999) afirma que a existência de uma entidade dentro de um relacionamento pode ser obrigatória ou opcional. Se a ocorrência de uma entidade em qualquer lado do relacionamento – seja ele “um” ou “muitos” – tiver que existir sempre para que a entidade seja incluída no relacionamento, diz-se que a entidade é obrigatória ou dominante. Quando esta ocorrência é dispensável, diz-se que a entidade é opcional ou subordinada. Em termos práticos isto significa que se uma entidade dominante for eliminada, conseqüentemente a sua entidade subordinada também o será.

Normalmente quando um relacionamento apresenta entidades opcionais, opta-se por redefini-lo como uma entidade fraca com uma relação do tipo “muitos” para todas as “pontas” que o relacionamento liga. As entidades fracas são dependentes de existência em todas as suas ocorrências, de uma entidade forte – no caso, as entidades que eram as pontas unidas pelo relacionamento são ditas fortes.

As restrições de integridade são fundamentais para o bom funcionamento de um banco de dados relacional. A integridade de entidades, tal como definida no modelo relacional, afirma que quando uma entidade existe e possui uma chave primária então sua chave primária também deve existir.

Uma das restrições que surgem com freqüência em bancos de dados relacionais são as restrições de integridade referencial. Teorey (1999) explica que a integridade referencial exige que para cada instância de uma chave estrangeira em uma tabela, a linha (o que inclui a instância da chave primária) da tabela mãe associada a esta chave estrangeira também deve existir.

Além dos conceitos da modelagem ER descritos até este ponto, existem outros decorrentes das sucessivas extensões que foram feitas ao modelo original proposto por Chen em 1976. As construções avançadas do modelo ER estendido são abstrações importantes para a modelagem conceitual integrada de projetos complexos. Os conceitos de generalização e agregação bem como de relacionamentos ternários e n-ários[9] são algumas dessas construções e sua descrição detalhada foge ao escopo deste trabalho.

2.5.7 Representação gráfica do modelo ER

Há diversas notações para apresentação de diagramas ER. A notação original ainda é utilizada mas algumas ferramentas CASE popularizaram outras notações como a de sombreamento, a *crow´s foot* e o IDEF1X.

Para demonstrar as notações gráficas serão tomados como exemplo três relacionamentos no domínio de um banco de dados de uma universidade, do mesmo modo como feito no item 2.5.4. O primeiro relacionamento, do tipo um-para-um, representa os chefes de departamento (um chefe para cada departamento). O segundo relacionamento representa o vínculo que cada professor tem com apenas um departamento. Para esse relacionamento um-para-muitos, vários professores são vinculados a um mesmo departamento, mas um mesmo professor só pode ser vinculado a um departamento. No terceiro relacionamento, representa-se as matrículas que um aluno faz em várias disciplinas. Nesse relacionamento do tipo muitos-para-muitos, um mesmo aluno matricula-se em várias disciplinas e vários alunos podem matricular-se na mesma disciplina.

Na Figura 2 pode-se observar esses relacionamentos representados na notação ER original de Chen. As entidades são representadas por retângulos e os relacionamentos são losangos que unem as entidades. No caso do relacionamento

[9] Um relacionamento binário liga duas entidades. Um relacionamento ternário liga três entidades e não pode ser representado por vários relacionamentos binários entre essas entidades. De forma análoga, um relacionamento n-ário liga n entidades e não pode ser representado por vários outros relacionamentos ternários ou binários.

um-para-um escreve-se em ambas as “pontas” o número um. Já nos relacionamentos um-para-muitos e muitos-para-muitos utiliza-se 1,N e M como pode ser visto na figura. Para representar uma entidade opcional utiliza-se um círculo sobre a linha de conexão junto ao retângulo da entidade opcional como na relação um-para-um desse exemplo.

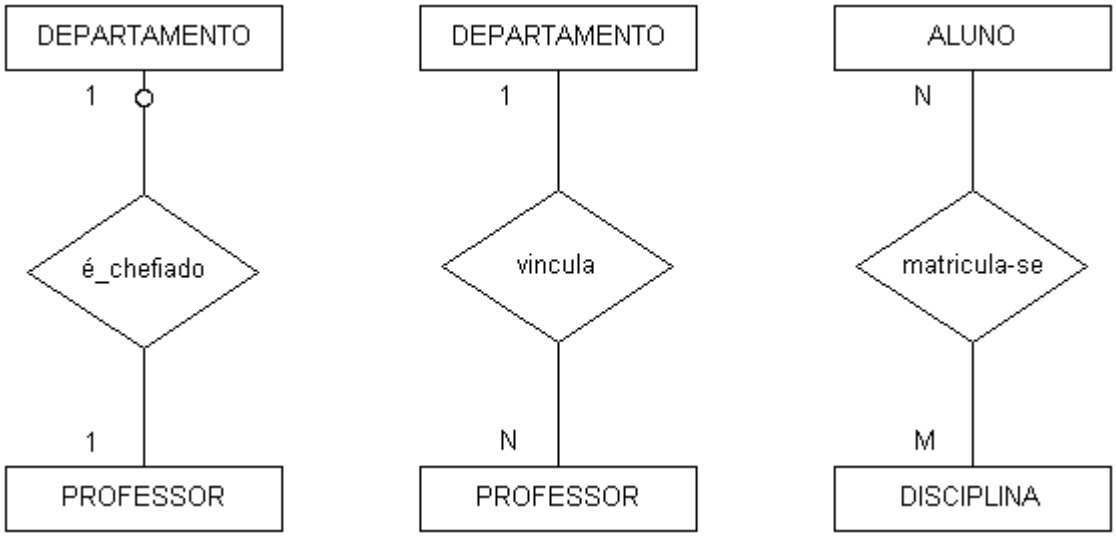


Figura 2: Relacionamentos um-para-um, um-para-muitos e muitos-para-muitos representados na notação original proposta por Chen.

Outra notação bastante utilizada é a *crow´s foot* que é uma variação da primeira proposta. Como pode ser visto na Figura 3, as entidades também são representadas por retângulos mas os relacionamentos são apenas as linhas de conexão com o nome do relacionamento escrito no centro. Nas entidades que são o lado “um” dos relacionamentos, utiliza-se duas barras perpendiculares e nas pontas do lado “muitos” a representação é feita com uma barra perpendicular e duas barras anguladas que dão origem ao nome *crow´s foot* (pata de corvo). Para representar uma entidade opcional utiliza-se um círculo da mesma forma que na notação de Chen.

Uma terceira notação bastante popular é o IDEF1X que também é bastante similar à proposta original. O IDEF1X é a notação adotada em diversas ferramentas CASE tal como a Platinum ERwin que foi escolhida como ferramenta de apoio ao projeto lógico neste trabalho.

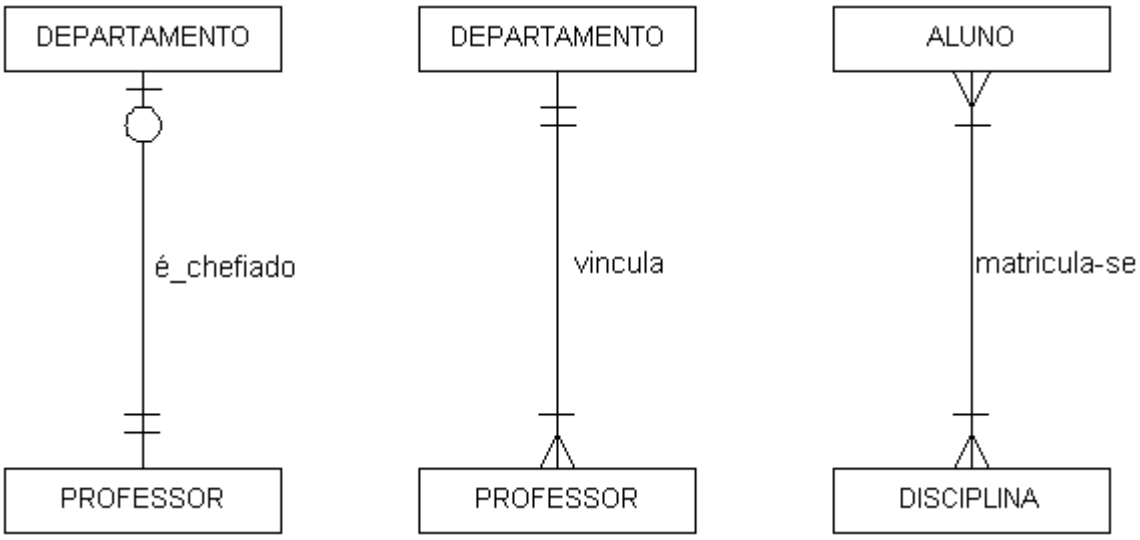


Figura 3: Relacionamentos um-para-um, um-para-muitos e muitos-para-muitos representados na notação *crow´s foot*.

Tal como nas outras abordagens, em IDEF1X as entidades são representadas por retângulos. Os relacionamentos são representados por linhas que conectam as entidades com o nome do relacionamento escrito no centro. Como pode ser visto na Figura 4, para o lado “muitos” dos relacionamentos, utiliza-se uma circunferência preenchida e para o lado “um” não é utilizado nenhum símbolo especial. Para a representação de uma entidade opcional, utiliza-se um losango junto ao retângulo da entidade opcional como na relação um-para-um desse exemplo.

Neste trabalho adotou-se a notação *crow´s foot* para a representação gráfica do modelo ER utilizado. Isto se deve às fortes semelhanças desta notação com a proposta original de Chen, por tratar-se de uma abordagem bastante popular e também por motivo de identificação pessoal do autor.

Após a modelagem através de diagramas ER vem a etapa de transformação desse modelo em tabelas candidatas[10]. De uma maneira simplificada, cada entidade transforma-se em tabela candidata diretamente. Uma transformação não direta, mas bastante comum é a criação de uma tabela para cada relacionamento do tipo

[10] Tabelas candidatas são as tabelas que na fase de implementação são passíveis de tornarem-se tabelas propriamente ditas no SGBD. Os fatores a serem avaliados para a decisão de criar ou não criar uma tabela candidata estão associadas principalmente a parâmetros físicos do BD.

muitos-para-muitos. No exemplo seriam criadas diretamente as tabelas departamentos, professores, alunos e disciplinas. O terceiro relacionamento do exemplo, que representa as diversas matrículas dos alunos nas disciplinas e é do tipo muitos-para-muitos, também torna-se uma tabela candidata.

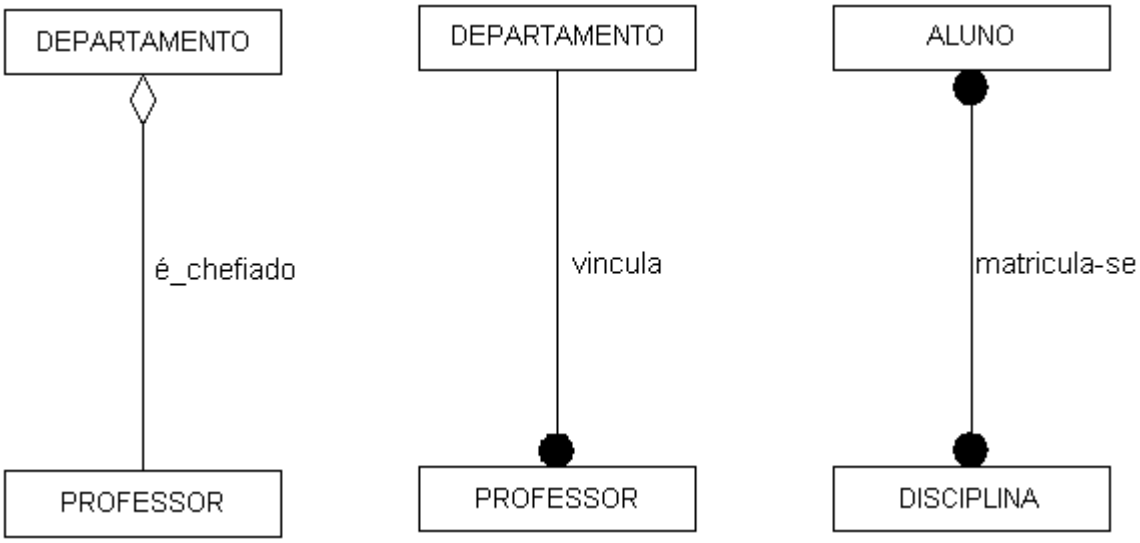


Figura 4: Relacionamentos um-para-um, um-para-muitos e muitos-para-muitos representados na notação IDEF1X.

2.5.8 Transformação do modelo ER para SQL

Logo após a transformação do modelo ER em tabelas candidatas vem uma das etapas mais importantes no projeto de banco de dados relacionais: a definição das tabelas candidatas em SQL. Para Teorey (1999), tal transformação é uma evolução natural do modelo ER para um esquema relacional. Devido ao fato de essa transição ser bastante natural, há diversas implementações de ferramentas CASE que não apenas viabilizam a geração de modelos ER, mas também fazem a conversão do modelo ER para SQL gerando automaticamente as tabelas e restrições de integridade. Na ferramenta ERwin pode-se gerar, a partir de um modelo ER, código de definição de dados SQL específico para diversos fabricantes de SGBD.

O *Structured Query Language* (SQL) – Linguagem de Consultas Estruturadas, apesar do nome, não é apenas uma linguagem para consultas. Segundo Korth & Silberschatz (1995), a linguagem também inclui recursos para definição da estrutura do banco de dados (esquemas) bem como para especificação de restrições de

autorização e integridade, permitindo também a definição de visões de dados. Sua versão original foi desenvolvida no Laboratório de Pesquisa da IBM em San Jose como parte do projeto *System/R* e chamava-se Sequel - *Structured English Query Language* ou Linguagem de Consultas Estruturadas em Inglês.

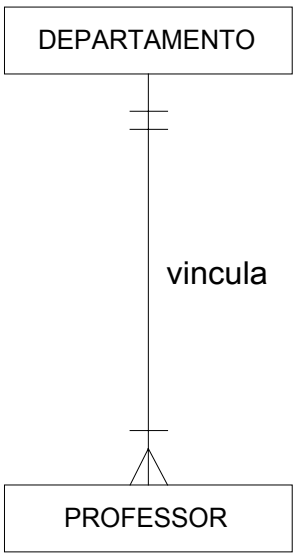
Um fator importante a respeito do SQL é seu caráter não-procedural. As linguagens procedurais descrevem detalhadamente como uma tarefa deve ser realizada e operam sobre apenas uma unidade de dados por vez. Diferentemente, as linguagens não-procedurais são uma descrição do que é desejado e cabe ao sistema resolver como conseguí-lo. Para melhor usar uma linguagem não-procedural como o SQL é necessário ter em mente a teoria de conjuntos e os conceitos da lógica formal.

Há uma grande quantidade de produtos que suportam a linguagem SQL mas cada fabricante tem sua própria versão. Apesar disso, as diferenças de implementação são secundárias. Celko (2000) afirma que atualmente o código SQL é bastante portável visto que os fabricantes convergiram seus produtos para o padrão ANSI / ISO SQL-92.

As regras para transformações das entidades no modelo ER para tabelas SQL dependem basicamente do tipo de tabela que é gerada. Há que se dar atenção especial às entidades do lado “muitos” de um relacionamento bem como às entidades fracas e aos relacionamentos ternários.

Tomando os exemplos do item anterior, a transformação do relacionamento um-para-muitos entre as tabelas professor e departamento (representação do vínculo que um professor tem com exatamente um departamento) em tabelas SQL pode ser vista na Figura 5.

É importante destacar neste exemplo que o relacionamento um-para-muitos na tabela professores é caracterizado e garantido pela presença da chave estrangeira (*foreign key*) que referencia a chave primária da tabela departamentos. No capítulo de desenvolvimento do protótipo discute-se em maior nível de detalhe essa fase do projeto lógico.




```
create table departamentos
  (DEPTO_COD      INTEGER
   DEPTO_NOME     CHAR(30),
   PRIMARY KEY    (DEPTO_COD));

create table professores
  (PROF_COD       INTEGER
   PROF_NOME      CHAR(25),
   DEPTO_COD      INTEGER NOT NULL,
   PRIMARY KEY    (PROF_COD),
   FOREIGN KEY    (DEPTO_COD) references departamentos
     on delete set default on update cascade);
```

Figura 5: Diagrama ER e representação SQL do relacionamento um-para-muitos entre as entidades departamento e professor.

2.6 Normalização

Teorey (1999) afirma que muitas vezes as tabelas de banco de dados relacionais sofrem de alguns problemas sérios quanto à eficiência, integridade e manutenibilidade. Por exemplo, quando se define um banco de dados inteiro como uma tabela grande e única, pode-se obter uma considerável quantidade de dados redundantes. Conseqüentemente as buscas feitas sobre essa tabela, mesmo com uma quantidade reduzida de registros, tendem a ser lentas. A execução de atualizações também pode demorar muito, tal como as exclusões que, além disso, ainda podem acarretar na eliminação de dados relevantes. Estes problemas também ocorrem em bancos de dados relacionais com várias tabelas, quando algumas delas possuem muitos atributos.

Estes aspectos indesejáveis num banco de dados relacional são chamados de anomalias. O processo de eliminação dessas anomalias de modo a obter relações mais adequadas é chamado de normalização. Para Tsichritzis & Lochovsky (*apud* Özsu & Valduriez, 1999) a normalização é um processo reversível no qual, dada uma coleção de relações, efetua-se sucessivas substituições nas quais as relações vão adquirindo progressivamente uma estrutura mais simples e regular.

De uma maneira geral, há quatro tipos de aspectos indesejáveis numa tabela relacional (Özsu & Valduriez, 1999):

1) **Anomalia de Repetição**: algumas informações podem ser repetidas sem necessidade. Dependendo da estrutura da tabela pode-se repetir uma mesma informação em tuplas (linhas) diferentes. Certamente trata-se de um desperdício de espaço de armazenamento além de ser uma deturpação do propósito de banco de dados devido à redundância gerada;
2) **Anomalia de Atualização**: como conseqüência da repetição de dados, a execução de atualizações pode ser problemática. Quando há redundância, é necessário efetuar atualizações em todos os lugares onde os dados são armazenados;
3) **Anomalia de Inserção**: pode não ser possível adicionar novas informações ao banco de dados. Por exemplo, se fossem armazenadas numa mesma tabela as informações do aluno e suas matrículas, não seria possível armazenar informações sobre o aluno antes que ele fizesse matrícula em alguma disciplina;
4) **Anomalia de Exclusão**: é o oposto da anomalia de inserção. Por exemplo, não é possível excluir todas as matrículas de um aluno uma vez que esse procedimento também excluirá as informações sobre o próprio aluno.

Para Özsu & Valduriez (1999), o objetivo do processo de normalização é transformar esquemas relacionais arbitrários em esquemas sem tais anomalias. A normalização é feita através da análise de interdependências existentes entre os vários atributos das tabelas. Executam-se operações de projeção que transformam as tabelas maiores em tabelas com menor quantidade de atributos.

O método mais popular de normalização de esquemas de bancos de dados relacionais é o da decomposição. Inicia-se com apenas uma relação, chamada relação universal, que contém todos os atributos (provavelmente com anomalias) que vai sendo iterativamente reduzida. A cada iteração, uma relação é dividida através de operações de projeção em duas ou mais relações de maior forma normal. Uma relação está numa determinada forma normal se ela satisfaz as condições

associadas a essa forma normal. Inicialmente, Codd definiu a primeira, segunda e terceira formas normais (1FN, 2FN e 3FN respectivamente).

> “As formas normais são uma tentativa de garantir que você não destrua dados ou crie dados falsos no seu banco de dados. Uma das maneiras de evitar erros é representar um fato apenas uma vez no banco de dados, visto que se um fato aparecer mais de uma vez, é provável que uma de suas instâncias esteja incorreta – um homem com dois relógios nunca pode estar certo de que horas são.”
>
> (CELKO, 2000, p.25)

Özsu e Valduriez (1999) demonstram que existe uma relação hierárquica entre essas formas normais. Toda relação normalizada está na 1FN; algumas das relações na 1FN também estão na 2FN, algumas das quais estão na 3FN e assim por diante. As formas normais maiores possuem propriedades melhores que as outras com relação às quatro anomalias discutidas anteriormente.

Segundo Korth & Silberschatz (1995), há várias extensões das três primeiras formas normais sendo os trabalhos mais significativos os de Boyce e Codd que propuseram uma versão modificada da 3FN conhecida como a forma normal de Boyce-Codd (FNBC) bem como os de Fagin que posteriormente definiu a quarta e quinta formas normais. A descrição detalhada destas extensões foge ao escopo desse trabalho.

Um dos requisitos fundamentais de um processo de normalização é que o mesmo seja “sem perda”, ou seja, a divisão de uma relação em várias outras não pode acarretar em perda de informação. Se for possível juntar as relações decompostas de modo a obter a relação original, diz-se que este processo é uma decomposição sem perda.

Uma operação fundamental para manipular tabelas normalizadas é a operação de *join* (junção). Intuitivamente trata-se de uma operação na qual dadas duas relações, concatena-se todas as tuplas da segunda relação com as tuplas da primeira de modo a satisfazer uma condição específica. A condição de junção é definida sobre os atributos das duas relações. O exemplo mais comum de condição de junção é

estipular que o valor de um atributo da primeira relação deve ser igual ao valor de um atributo da segunda relação. Em nível de aplicação, é possível tratar os dados das tabelas que sofreram a operação de junção, como se fossem de uma mesma tabela de maneira transparente para o usuário.

Devido à natureza dos bancos de dados relacionais a operação de junção é bastante comum e é importante que sua execução seja eficiente. Uma das modificações em nível físico que pode aumentar significativamente a eficiência do banco de dados é a definição da estratégia para as operações de junção. Esta otimização é conseguida através da definição da ordem de execução dos predicados e de mecanismos de indexação e *hashing*[11].

2.7 Tecnologias recentes

Na era da tecnologia da informação, a crescente complexidade dos bancos de dados aliada à necessidade de integrar os sistemas de diversas organizações e extrair conhecimento de seus bancos de dados, tornou-se necessário o desenvolvimento de diversas tecnologias e metodologias para suprir os requisitos avançados desses sistemas.

Neste contexto, é importante distinguir os termos dado, informação e conhecimento. Os dados são conjuntos de ações ou medidas não processadas, sem qualificação. A informação vem a ser o enriquecimento dos dados quando dotados de contexto. Já o conhecimento é a informação quando relacionada com um *know-how* ou um *know-why* significativo. O conhecimento é o recurso que permite converter informação em decisões e ações (Souza, 2000).

---

[11] *Hashing* pode ser traduzido como espalhamento, mas o termo é empregado em inglês a nível mundial. Trata-se de uma técnica amplamente utilizada em computação e tem o mesmo princípio dos índices com a diferença que não há a necessidade de acessar uma estrutura de dados para descobrir o ponteiro para o item de dado. Os ponteiros são descobertos através de uma lei matemática de formação nas chamadas funções de *hash*.

Uma tecnologia básica para a aplicação de técnicas avançadas de Descoberta de Conhecimento em Banco de Dados (KDD – *Knowledge Discovery in Databases*) é o *data warehouse*[12] (DW).

> "Um *data warehouse* é um grande repositório de dados históricos que podem ser integrados para apoio à decisão. O tamanho dos dados em *data warehouses* pode chegar a centenas de *gigabytes* ou até *terabytes.* Considera-se elementos essenciais para o apoio à decisão o *data warehouse* e o *on-line analytical processing* (OLAP). Diversos produtos comerciais estão disponíveis atualmente para contemplar esses elementos e a maioria dos fabricantes de SGBD fornece suas próprias ferramentas para essas tecnologias."
>
> (TEOREY, 1999, p.211)

Barquin & Eldestein *(apud* Teorey, 1999) consideram que os três principais usos de *data warehouses* são a geração de relatórios e gráficos padrão similares àqueles disponíveis em SGBD atualmente; as análises dimensionais tais como no OLAP e a mineração de dados. OLAP é uma forma sofisticada de metodologia de consulta usada para agregar e resumir dados em um DW. Já o *data mining* ou mineração de dados é uma forma ainda mais complexa de metodologia de consulta utilizada para descobrir tendências e relações não óbvias nos dados.

A mineração de dados combina métodos e ferramentas das seguintes áreas: aprendizagem de máquina, estatística, banco de dados, sistemas especialistas e visualização de dados (Cratochvil *apud* Dias, 2001).

As aplicações de DW beneficiam-se da organização lógica e física dos dados no DW e a sua eficiência depende fortemente das técnicas de projeto utilizadas. As pesquisas atuais procuram determinar um modo eficiente de integrar ferramentas OLAP e de mineração de dados.

---

[12] *Warehouse* é um termo inglês que significa armazém, depósito. *Data warehouse* pode ser traduzido como repositório de dados. O termo é amplamente empregado na língua original.

2.8 Considerações finais

O sucesso dos sistemas de banco de dados relacionais pode ser atribuído a vários fatores. O modelo relacional é de fácil compreensão e possui uma fundamentação teórica forte. Além disso, foi adotada a linguagem de consultas padronizada ANSI-SQL que é completamente integrada aos sistemas de banco de dados. A arquitetura dos SGBDR abrange a gama completa de funcionalidades: definição de dados, manipulação de dados, consulta, controle dos dados, gerenciamento de transações etc. Atualmente, os sistemas de gerenciamento de banco de dados relacionais estão consolidados no mercado e há diversos fabricantes disponibilizando seus produtos.

Neste capítulo foram apresentados diversos conceitos de banco de dados, dando ênfase aos bancos de dados relacionais e a abordagem entidade-relacionamento. A utilização de bancos de dados em sistemas de educação auxiliada por computador (CAE – *Computer Aided Education*) vem de longa data e atualmente é amplamente empregada em sistemas de Educação à Distância (EaD). No próximo capítulo serão apresentados conceitos e tecnologias relacionados à manipulação de informações em sistemas baseados em WWW - *World Wide Web*.

3. TECNOLOGIAS DE REDE

"*Se algo inerte é colocado em movimento, ele gradualmente ganhará vida*."
Lao Tzu

3.1 Introdução

> "Cada um dos três últimos séculos foi dominado por uma única tecnologia. O Século XVIII foi a época dos grandes sistemas mecânicos, característica da Revolução Industrial. O Século XIX foi a era das máquinas a vapor. As principais conquistas tecnológicas do século XX se deram no campo da informação. Entre outros desenvolvimentos, vimos a instalação das redes de telefonia em escala mundial, a invenção do rádio e da televisão, o nascimento e crescimento sem precedentes da indústria de computadores e o lançamento dos satélites de comunicação."
>
> (TANENBAUM, 1997, p.1)

Para Soares et al. (1995, p.3) a conjunção das tecnologias de comunicação e processamento de informações "veio revolucionar o mundo em que vivemos, abrindo as fronteiras com novas formas de comunicação, e permitindo maior eficácia dos sistemas computacionais".

Inicialmente a tecnologia de redes se desenvolveu devido ao interesse das grandes empresas em compartilhar recursos como equipamentos, programas e principalmente dados, independentemente da localização física desses recursos e dos usuários. A rede também aumenta a confiabilidade dos sistemas devido às fontes alternativas de recursos que, em caso de falha de uma máquina, podem manter o sistema funcionando e garantir a consistência dos dados. Uma outra vantagem das redes é a escalabilidade que é a possibilidade de aumentar gradualmente a capacidade do sistema de acordo com o volume da carga (Tanenbaum, 1997).

Atualmente as redes não são utilizadas somente nas grandes empresas e há um número cada vez maior de usuários domésticos que utilizam tecnologias de rede para acessar informações remotas dos mais variados assuntos e para comunicação. Esse crescimento deve-se principalmente ao advento da *World Wide Web* (WWW).

3.2 Hipertexto, hipermídia e WWW

3.2.1 Histórico

Nielsen (1995) afirma que Vannevar Bush em seu tratado futurista de 1945 intitulado "*As We May Think*" propôs o sistema Memex dando origem à idéia de tornar acessível ao homem comum o "conhecimento herdado dos tempos". Apesar de nunca ter sido implementado de fato, este sistema foi descrito em detalhes por Bush utilizando a tecnologia da época: o microfilme.

Embora este trabalho seja considerado atualmente como o precursor do hipertexto, naquela época as pesquisas na área não receberam muita atenção uma vez que os esforços concentravam-se em desenvolver os computadores para tornar o seu uso interativo. Com a evolução tecnológica atingida na década de 1960, muitos especialistas começaram a trabalhar em novos sistemas hipertexto tais como o Augment de Douglas Engelbart e o Xanadu de Ted Nelson (Nielsen, 1995). Da evolução desses trabalhos surgiram os conceitos de hipertexto e hipermídia sendo que atualmente a *World Wide Web* é o maior sistema de disseminação de conhecimento que utiliza tais conceitos.

Para Bugay & Ulbricht (2000), Ted Nelson criou o termo hipertexto para referir-se a uma organização não-linear da informação. Esta organização permite situar assuntos diferentes, mas inter-relacionados em diferentes níveis de aprofundamento, proporcionando a personalização do processo de ensino-aprendizagem e permitindo ao usuário trabalhar em seu próprio ritmo, nível e estilo. Cabe ao usuário escolher a seqüência da navegação e explorar as informações do hipertexto. Já a hipermídia é uma evolução da noção de hipertexto e trata-se da apresentação computadorizada

da informação em forma hipertextual, combinada com a multimídia (uso através do computador de textos, gráficos, sons, imagem, animação, simulação, processamento de programas e vídeo).

3.2.2 WWW – World Wide Web

> "Todos esses cientistas tiveram problemas que exigiam soluções novas e inéditas. Tim Berners-Lee precisou de um ambiente colaborativo para permitir que os físicos trocassem informações vitais sobre pesquisas sem sofrerem os atrasos inerentes às comunicações, tais como correio eletrônico (*e-mail*), protocolo de transferência de arquivos (ftp), correios convencionais ou fax. Dessa forma, a WWW foi criada em um porão, entre canos fumegantes e uma válvula enferrujada, no Laboratório Europeu de Física de Partículas (CERN), em Genebra, Suíça em 1990."
>
> (TITTEL et al., 1996, p.10)

Berners-Lee et al. (1994) explicam que a *World Wide Web* (WWW ou W3) foi desenvolvida para ser um aglomerado de conhecimento humano que iria permitir a colaboradores em locais remotos o compartilhamento de idéias e todos os aspectos de um projeto em comum.

Associado ao sistema de navegação hipertextual, o usuário da *web* pode também efetuar buscas de texto simples ou com comandos complexos para acessar os documentos hipertexto ou hipermídia relacionados ao assunto de seu interesse. Berners-Lee et al. (1994) constatam que esse processo é feito repetidamente e através dessa seqüência de buscas e seleções o usuário pode encontrar qualquer coisa que esteja "lá fora".

Pode-se observar que o direcionamento dado a *web* por Berners-Lee e os pesquisadores do CERN está em conformidade com o pensamento original de Vannevar Bush: compartilhar o conhecimento da humanidade ao homem comum. Para Myers et al. (1996), o fantástico crescimento da WWW é um resultado direto da aplicação que Tim Berners-Lee fez do hipertexto como interface para características previamente existentes da Internet. Atualmente, devido a este crescimento

exponencial, a grande maioria dos usuários não sabe distinguir a Internet da WWW e usa os termos equivocadamente como sinônimos.

> "A WWW é um sistema interativo hipermídia construído sobre a Internet, que cresceu a partir de uma pequena parte das máquinas do DARPA (*Defense Advanced Research Project Agency*), chegando atualmente em seus dias de glória total. Não vamos lhe dizer o tamanho da Internet, pois ninguém o sabe realmente! Tudo o que sabemos é que existe uma massa enorme e disforme lá do outro lado das linhas telefônicas, onde os pacotes de dados são passados a velocidades espantosas entre os servidores ligados ao *backbone* da Internet. "
>
> (TITTEL et al., 1996, p.10)

O sucesso da WWW foi praticamente imediato e além da enorme quantidade de informação na forma de hipermídia surgiram aplicações dos mais variados tipos na *web*: desde comércio eletrônico até a implantação de sistemas inteiros interligando diversas organizações. Isto causou uma revolução na forma como as pessoas trabalham com os sistemas de informação.

> "As aplicações para Internet nos domínios de comércio eletrônico, bibliotecas digitais e ensino a distância são caracterizadas por uma mistura de características sem precedentes que as tornam radicalmente diferentes de aplicações anteriores de tecnologia da informação."
>
> (MYERS et al., 1996, p. 794)

Nos últimos anos, a *World Wide Web* tem sido escolhida como plataforma para desenvolvimento de aplicações da Internet. Isto se deve ao seu poderoso paradigma de comunicação baseado na navegação (*browsing*) e multimediática, bem como à sua arquitetura padronizada e aberta que facilita a integração de diferentes tipos de conteúdos e sistemas (Fraternali, 1999).

De acordo com Rowe (1997), esse crescimento fenomenal de aplicações hipermídia na *web* é devido à facilidade que os autores têm de publicar documentos eletrônicos para uma "platéia" em nível mundial e com custos baixos.

3.3 Padrões da WWW

3.3.1 HTML

O padrão da *web* para armazenamento e distribuição de documentos eletrônicos, na forma de hipermídia, é a “linguagem” HTML. De acordo com Lie & Saarela (1999), o HTML tem suas raízes no mesmo grupo de pesquisa que deu origem à WWW no CERN em 1990. Naquela época, o HTML servia para os físicos que colaboravam entre si através do compartilhamento de artigos científicos na Internet.

JONES (2000) define o HTML (*HyperText Markup Language*) como um formato de texto com marcações (*tags*) utilizado para apresentar conteúdo da *Web* num navegador (*browser*). Uma vez que é uma linguagem bastante simples, com apenas alguns “comandos”, o HTML rapidamente tornou-se popular para apresentar documentos formatados independentes de plataforma e de processador de texto.

> “O termo *marcação* vem dos tempos em que os editores realmente marcavam os documentos para informar ao impressor – naquele tempo um ser humano – quais fontes usar e assim por diante. Portanto, as linguagens de marcação contêm comandos de formatação explícitos. Por exemplo, em HTML, <B> **significa início do modo negrito, e** </B> significa fim do modo negrito.”
>
> (TANENBAUM, 1997, p. 792)

Para Marchal (2000), de acordo com alguns estudos existem cerca de 800 milhões de páginas *Web*, todas baseadas no HTML. A WWW cresceu a partir da aplicação original de colaboração científica e tornou-se uma mídia de importância vital, principalmente devido à sua natureza interativa.

Devido às limitações do HTML 1.0, muitas extensões foram criadas nos últimos anos e a versão 4.0 tem mais de 100 marcações contra os pouco mais de dez da versão criada por Tim Berners-Lee. Além das extensões do HTML em si, um conjunto considerável de tecnologias foi criado para dar apoio às aplicações *web* tal como o Javascript, Java, CGI, ASP, JSP, PHP, Shockwave Flash, MP3, CSS, XML etc.

3.3.2 Conteúdo Interativo e Dinâmico

A WWW é um sistema de informação hipertexto disponível na Internet. Ela opera baseada no modelo Cliente/Servidor[13]. O cliente *web* é o navegador[14] tal como o Netscape ou o Internet Explorer. O servidor é qualquer máquina rodando o *software* para servidor *web*. Ao acessar uma página HTML o navegador lê o documento escrito e o apresenta, interpretando os códigos de marcação do documento. Como pode ser visto na Figura 6, quando o usuário clica em um *hyperlink*, o navegador envia uma requisição de rede através do protocolo HyperText Transfer Protocol (HTTP) – Protocolo de Transferência Hipertexto – a um servidor *web* para acessar o documento ou serviço especificado pelo *hyperlink*. O servidor *web* responde a requisição com o documento solicitado. Por fim, o navegador lê, interpreta e apresenta a informação ao usuário (Ye, 1999).

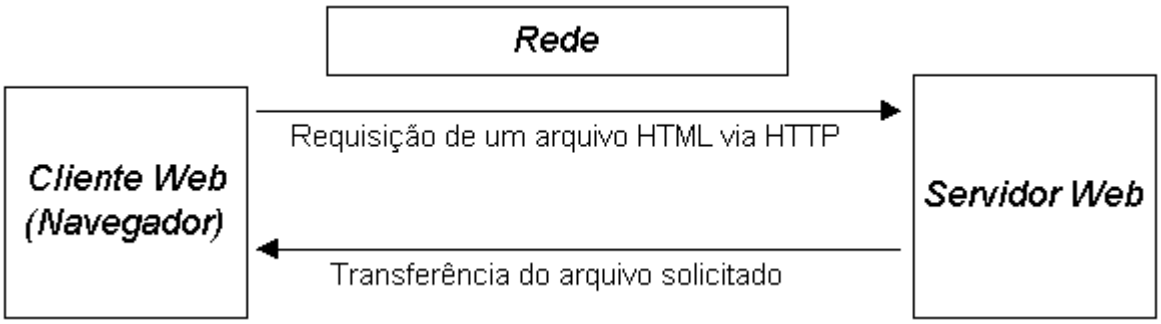


Figura 6: Iteração HTTP típica: o cliente requisita um arquivo HTML via HTTP e o servidor envia o arquivo. O cliente interpreta os *tags* HTML e apresenta o documento. Quando o usuário clica sobre um *hyperlink* o ciclo se repete.

Esta é a forma original de requisição e apresentação de conteúdo estático que popularizou a WWW. Uma forma mais recente, e com maiores possibilidades é a disponibilização de páginas dinâmicas. Jones (2000) explica que as páginas nas

---

[13] A arquitetura cliente/servidor é abordada no item 3.4.

[14] *Browser* ou navegador é o *software* cliente da WWW e permite ao usuário visualizar e navegar por entre os documentos hipermídia da *web*. A primeira versão gráfica do tipo “aponte e clique” de um navegador foi o *Mosaic*, desenvolvido por Marc Andreesen a partir dos trabalhos do CERN. Posteriormente, o *Netscape Navigator* ganhou muita popularidade e até hoje é padrão em várias plataformas excetuando-se a mais comum delas: PC/Windows onde a maioria dos usuários da WWW utiliza o *Internet Explorer* da Microsoft.

quais o conteúdo muda a cada requisição são ditas de conteúdo dinâmico. As páginas dinâmicas dominam a *web* de hoje e os *sites* nos quais o usuário interage com o conteúdo no lugar de apenas apontar e clicar para mudar a página são chamados *aplicações web*.

As aplicações *web* modernas podem ser definidas como híbridos entre hipermídia e sistemas de informação (Nielsen, 1995). As aplicações *web,* de modo semelhante as hipermídias, podem apresentar mídias discretas como texto e imagens bem como mídias contínuas tais como som e vídeo e essa informação é acessada de forma exploratória (não-linear). Também de forma semelhante aos sistemas de informação, o tamanho e a volatilidade dos dados nas aplicações *web* aliado às necessidades especiais de distribuição demanda soluções baseadas em arquiteturas consolidadas tais como os sistemas de gerenciamento de banco de dados e computação cliente/servidor (Fraternali, 1999).

Devido a esse caráter híbrido, o desenvolvimento de aplicações *web* deve lidar com uma série de requisitos tais como:

- a necessidade de tratar dados estruturados tais como registros em uma tabela de banco de dados e não-estruturados tais como diferentes itens multimídia;
- suportar o acesso não-linear através de interfaces que permitam a navegação;
- alto nível de qualidade gráfica;
- personalização e adaptação dinâmica da estrutura do conteúdo e do estilo de apresentação;
- suporte ao comportamento pró-ativo[15].

Quanto à apresentação de mídias contínuas como áudio e vídeo na rede foram desenvolvidas diversas tecnologias como a Real Audio e Real Video da RealNetworks e o Windows Media da Microsoft. Outras tecnologias como o

[15] Entende-se por comportamento pró-ativo de uma aplicação, sua capacidade de tomar decisões de modo autônomo tais como filtrar conteúdo, alterar o curso da navegação ou efetuar recomendações em benefício do usuário. Um exemplo prático é o do *site* de compras Amazon.com no qual de acordo com o perfil do usuário (determinado principalmente por compras anteriores) o agente sugere a compra de itens relacionados a esse perfil.

Shockwave Flash e Shockwave Director da Macromedia permitem a apresentação de conteúdo interativo bem como de animações e vídeos. Estas tecnologias otimizam a mídia gerada através de compactação de dados e *streamming*[16] para que ela seja rapidamente carregada pela rede diminuindo a espera do usuário.

De acordo com Jones (2000), o HTML é uma linguagem simples para fins de *layout* e, portanto não é suficiente para lidar com os requisitos necessários ao desenvolvimento de aplicações *web*. Para suprir essas limitações, diversas tecnologias foram desenvolvidas tais como linguagens de apoio que executam no lado do cliente e arquiteturas de servidor para a disponibilização de conteúdo dinâmico.

As linguagens de apoio ou s*cripts* são linguagens interpretadas (não há compilação) e relativamente limitadas. Os tipos mais comuns no contexto da WWW são o Javascript para aplicações Netscape e o JScript (similar ao Javascript) e VBScript da Microsoft.

A primeira proposta de linguagem para *web* foi o *Common Gateway Interface* (CGI). As aplicações CGI executam inteiramente no lado do servidor. Quando um *browser* contata o servidor solicitando por um arquivo CGI (um programa na forma de *script*), o servidor executa o código desse CGI e retorna para o cliente uma página HTML simples. A vantagem é que o programa CGI pode processar a informação enviada pelo *browser* e retornar HTML que varia de acordo com condições diferentes (Jones, 2000).

As primeiras aplicações CGI eram em sua maioria uma forma de receber, processar e armazenar dados oriundos do cliente. Por exemplo, um usuário da *web* deseja se cadastrar para receber notícias a respeito de uma empresa. Para tal, o usuário preenche seus dados incluindo seu endereço eletrônico num formulário no

---

[16] *Streamming* vem da palavra inglesa “stream” que significa fluxo. Ela refere-se a um fluxo de dados que é descarregado da rede no qual a apresentação do conteúdo começa antes do término do *download* desses dados. Utilizando essa tecnologia a apresentação do conteúdo começa antes diminuindo o tempo de espera do usuário.

navegador (tal como na Figura 7) e envia a requisição ao servidor. O formulário da página HTML determina o método de envio[17] e qual arquivo CGI processará a requisição. Por sua vez, o programa CGI recebe esses dados no servidor, processa-os (geralmente inserindo-os num banco de dados) e retorna ao usuário a confirmação (ou mensagem de erro) do cadastro ao usuário através de uma página HTML gerada dinamicamente.

Como primeira solução, o CGI original chegou a ser bastante utilizado para adicionar um certo grau de interatividade às páginas *web.* No entanto, os programas CGI têm deficiências importantes: Jones (2000) e Simons & Babel (2001) afirmam que para processar texto e retornar HTML para apresentação, o CGI é uma opção pouco confortável e com o inconveniente de ter que carregar o executável na memória, processar e descarregar da memória para cada requisição.

> "Uma questão de arquitetura ortogonal é o momento da ligação entre o conteúdo da base de informações e as páginas apresentadas ao cliente. Ele pode ser *estático,* quando as páginas são computadas no momento da definição da aplicação e, portanto imutáveis durante a utilização da mesma. Também pode ser *dinâmico* quando as páginas são criadas sob demanda a partir de conteúdo recente. A dinamicidade possui graus diferentes: ela pode limitar-se ao conteúdo (mantendo a navegação e apresentação estáticas) ou pode estender-se para a apresentação e navegação. "
>
> (FRATERNALI, 1999, p. 232)

Um exemplo simples de conteúdo dinâmico é o de um *site* que mantém atualizado o histórico e o valor atual de cotações de moedas. A variação dos valores precisa estar sendo atualizada com grande freqüência e além da forma numérica é necessário apresentar essa informação na forma gráfica. Uma solução para esse tipo de serviço é utilizar uma solução para disponibilização de conteúdo dinâmico

[17] No protocolo HTTP há dois métodos para envio de parâmetros e valores do cliente para um programa no servidor: o *post* e o *get*. O método *get* envia os parâmetros e valores através do próprio endereço do recurso (URL) e é limitado a 255 caracteres. No *Post*, é possível enviar uma matriz de parâmetros e valores de tamanho arbitrariamente longo (Tittel et al., 1996).

com acesso a um repositório de dados (tipicamente um banco de dados). Dessa forma, basta atualizar as informações do repositório e quando o usuário acessar o *site*, o mecanismo de conteúdo dinâmico se encarrega de ler os dados recentes do repositório e montar a página gerando dinamicamente informação textual e gráfica atualizada. Esse processo agiliza o acesso e a distribuição da informação minimizando a necessidade de atualização direta nas páginas.

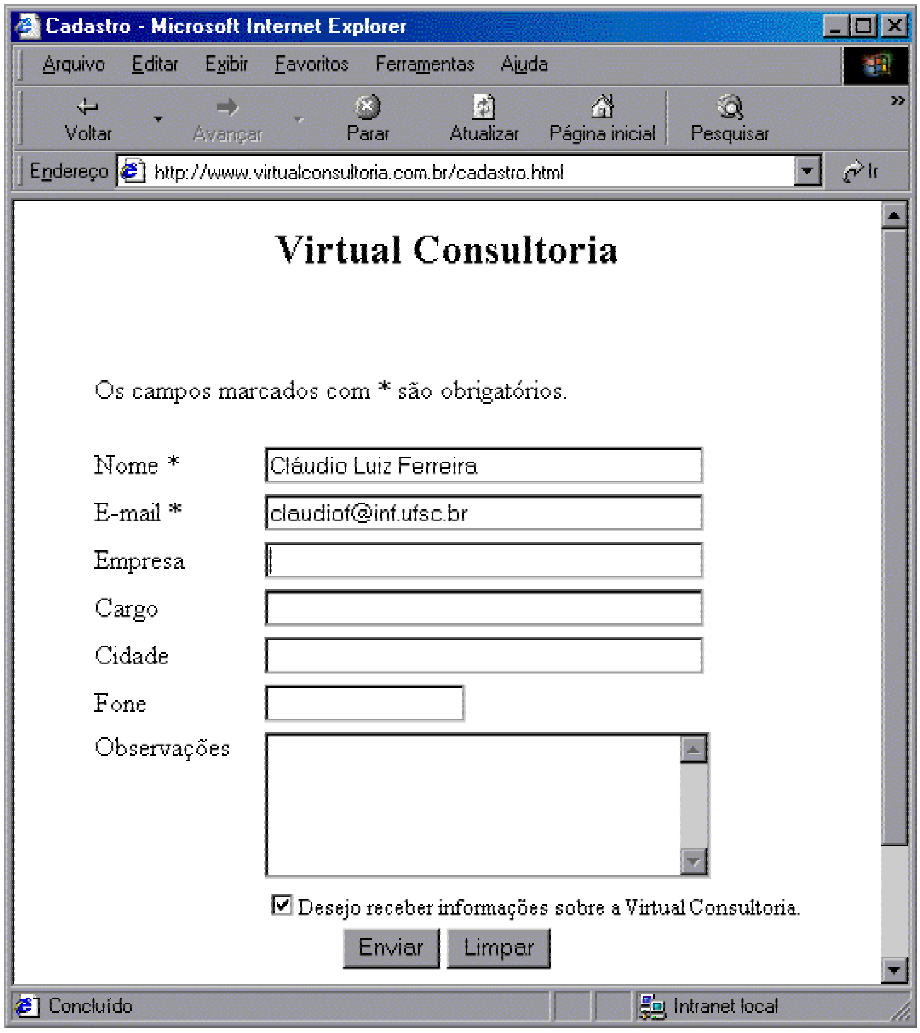


Figura 7: Tela do navegador com o formulário de cadastro

Para Berry et al. (1999), os conteúdos dinâmicos para web estão se tornando cada vez mais importantes ao passo que os serviços de Internet continuam evoluindo. Um serviço dinâmico da *web* gera conteúdo sob demanda executando código sobre um estado interno geralmente extraído de um banco de dados. A resposta gerada pode depender de argumentos da requisição ou de informação de usuário mantida no cliente e o serviço pode atualizar o estado interno em resposta à requisição.

De acordo com Jones (2000), as soluções mais recentes para a disponibilização de conteúdo dinâmico na *web* vieram com o advento da tecnologia *Internet Server Aplication Programming Interface* (ISAPI) - Interface de Programação para Aplicação em Servidores Internet. Trata-se de um padrão que permite aos programadores usar linguagens quaisquer (como o C++ e o Delphi) para processar requisições e retornar conteúdo dinâmico tal como no CGI. Entretanto, uma vez carregados esses programas residem na memória do servidor e portanto não precisam ser novamente carregados e descarregados cada vez como os programas CGI.

A partir do ISAPI surgiram linguagens para processamento no servidor tal como o Active Server Pages (ASP), JavaServer Pages (JSP) e Personal Home Page (PHP) sendo que esta última é atualmente a opção mais amplamente empregada nas aplicações *web* (Simons & Babel, 2001).

Todas essas linguagens permitem a geração de conteúdo dinâmico na *web* bem como o acesso a banco de dados e a integração com componentes tais como ActiveX, COM, Corba e Java. A principal diferença é que o ASP é dependente de plataforma. Há características específicas para cada uma dessas tecnologias e a escolha de uma ou outra para a implementação de uma aplicação *web* deve levar em consideração aspectos como a facilidade de aprendizagem, o tipo de licença de *software*, versatilidade, quantidade e qualidade de bibliotecas disponíveis, tipo de acesso a banco de dados, necessidade de interação com componentes etc (Svennes, 2001).

Considerando o exemplo anterior de um *site* para disponibilização de cotações atualizadas de moeda, pode-se desenvolver uma aplicação *web* em ASP para viabilizar as funcionalidades necessárias. Quando o usuário requisita a página com as cotações, o servidor *web* dispara a aplicação ASP que faz o acesso ao banco de dados (tal como o Microsoft Access) com as cotações atualizadas e monta a página dinamicamente retornando-a ao usuário. Essa mesma aplicação poderia ser desenvolvida utilizando PHP com o banco de dados mySQL ou a linguagem JSP com o banco de dados Oracle.

3.3.3 Documentos semiestruturados: XML

É inegável o sucesso do HTML como padrão de documento eletrônico independente de plataforma, mesmo com suas limitações enquanto linguagem de formatação. Entretanto, ao passo que a WWW se tornou uma unanimidade, começaram a surgir aplicações que forçam o HTML a lidar com dados, principalmente as aplicações com maior interação dos usuários (Jones, 2000).

O HTML é flexível o suficiente para a apresentação dos dados, mas não está preparado para lidar com a semântica do conteúdo. Para tal, precisa-se de um formato simples e legível que possa ser aplicado para diferentes domínios de informação. O formato que foi criado nesse sentido, que mais tarde viria a tornar-se padrão mundial é o XML.

Harold (1999) explica que XML vem de *Extensible Markup Language* – Linguagem de Marcação Extensível, mas é tipicamente escrito como *eXtensible Markup Language* para justificar o acrônimo. O XML é um conjunto de regras para a definição de marcações (*tags*) semânticas que dividem um documento em partes e identificam as partes desse documento. Trata-se de uma linguagem de meta-marcações que define uma sintaxe utilizada para definir outras linguagens estruturadas de domínio específico. O XML foi definido como um subconjunto do SGML[18] para facilitar a sua implementação e viabilizar o seu uso na WWW.

A codificação computadorizada de documentos sempre concentrou-se em preservar a "forma final de apresentação", tal como num documento impresso e bem formatado. Por outro lado, os formatos estruturados de documentos utilizam uma abordagem diferente na qual, no lugar de preservar a formatação, armazena-se a estrutura lógica do documento. Entre outras vantagens, essa abordagem permite a

---

[18] O SGML (*Standard Generalized Markup Language*) é um padrão internacional (ISO) usado na definição formal de texto eletrônico independente de dispositivo, sistema ou aplicativo. Em outras palavras, a SGML é uma metalinguagem (uma linguagem utilizada para descrever linguagens em geral), que define formalmente uma linguagem de marcação descritiva. O poder da SGML é sua abordagem de plataformas múltiplas e baseadas em estrutura para descrever o conteúdo estrutural dos documentos. (TITTEL et al., 1996, p.19).

independência de dispositivo, a possibilidade de efetuar buscas complexas e a reutilização da informação (Lie & Saarela, 1999).

Esta estrutura pode ser descrita formalmente através de um documento de descrição do esquema. A definição formal do esquema em documentos XML é feita por um Documento de Definição de Tipo (*Document Type Definition* – DTD) e permite verificar a validade da estrutura do documento. Gillies et al. (1999) afirmam que o formato do DTD é definido através de expressões regulares e notação BNF. A aplicação usa o *parser*[19] XML e o DTD (opcionalmente) para ler, verificar a boa formação e validar o documento XML. O DTD é o "acordo" que define quais elementos o documento pode conter proporcionando um modelo determinístico e não ambíguo.

Para Marchal (2000), a necessidade de utilizar um modelo determinístico advém do conflito entre flexibilidade e facilidade de uso. O HTML é uma solução fechada que pode ser otimizada enquanto o XML é flexível e permite a criação de estruturas específicas. A definição formal do esquema é uma tentativa de otimizar o modelo flexível do XML. Uma vez que o DTD é uma descrição formal do documento, o *software* pode conhecer a estrutura do documento e portanto adaptar-se a ela.

É importante distinguir duas classes de documentos XML: os documentos bem formados e os documentos válidos. Os primeiros simplesmente seguem a sintaxe do XML e portanto possuem as marcações de início e fim, atributos entre aspas, nomes de entidade segundo as regras e os caracteres usados são compatíveis com o conjunto de caracteres aceitáveis. Já os documentos válidos são documentos bem formados que possuem uma estrutura válida conforme uma definição DTD (Marchal, 2000).

---

[19] Marchal (2000) explica que a palavra *parser* vem da Teoria de Compiladores. Num compilador, um *parser* é o módulo responsável pela leitura e interpretação da linguagem de programação. O *parser* cria uma árvore de *parse* que é uma representação do código fonte na memória. A parte de síntese do compilador utiliza as árvores de *parse* para gerar arquivos-objeto (módulos compilados).

Ao contrário do padrão SGML para documentos estruturados, os documentos XML são semiestruturados, pois o esquema não é obrigatório, ele pode aparecer junto com o conteúdo do documento numa forma autodescritiva ou ainda conter apenas algumas restrições para partes específicas do documento (Buneman, 1997).

De acordo com Marchal (2000), há dois tipos distintos de utilização do XML:

- Documentos eletrônicos com finalidade de transferência de dados. O documento é de uso interno do *software,* ou seja, é uma forma de transferência de dados entre processos ou computadores distintos. É uma solução bastante empregada para padronizar a transferência de dados entre cliente/servidor e também entre servidores.
- meio de armazenamento de informações com finalidade de publicação. Nesta forma, deve haver uma maneira de transformar automaticamente o conteúdo XML armazenado em um outro formato tal como HTML ou RTF para ser lido pelo usuário final.

No escopo deste trabalho, destaca-se a utilização do XML como padrão para transferência de dados. Segundo Manolescu et al. (2001), esta é uma utilização bastante freqüente visto que o XML tornou-se o padrão para intercâmbio de informações devido à sua flexibilidade, portabilidade e simplicidade.

3.4 Arquitetura dos sistemas computacionais

Para entender o funcionamento de aplicações desenvolvidas para rede, seja esta de escopo local ou mundial como a Internet, é necessário compreender a arquitetura dos sistemas computacionais.

Segundo Orfali et al. (1999), da mesma forma que a arquitetura em sua acepção tradicional determina a estrutura das casas, edifícios, construções e cidades onde vivemos, na analogia da computação a arquitetura nos ajuda a determinar a estrutura e a forma dos sistemas e soluções computacionais.

No contexto da informática, arquitetura é o objeto do projeto e da implementação e reflete a organização espacial dos dados da aplicação bem como a distribuição espaço-temporal da computação (Fraternali, 1999).

3.4.1 Histórico

Para conhecer o modo como as arquiteturas modernas se configuram é importante entender um pouco da evolução dos sistemas computacionais. Soares et al. (1995) explicam que na década de 1950 os computadores eram máquinas grandes e complexas, operadas por pessoas altamente especializadas. Como pode ser visto na Figura 8, os usuários enfileiravam seus programas - na forma de cartões perfurados – que posteriormente eram processados em lote (*batch*).

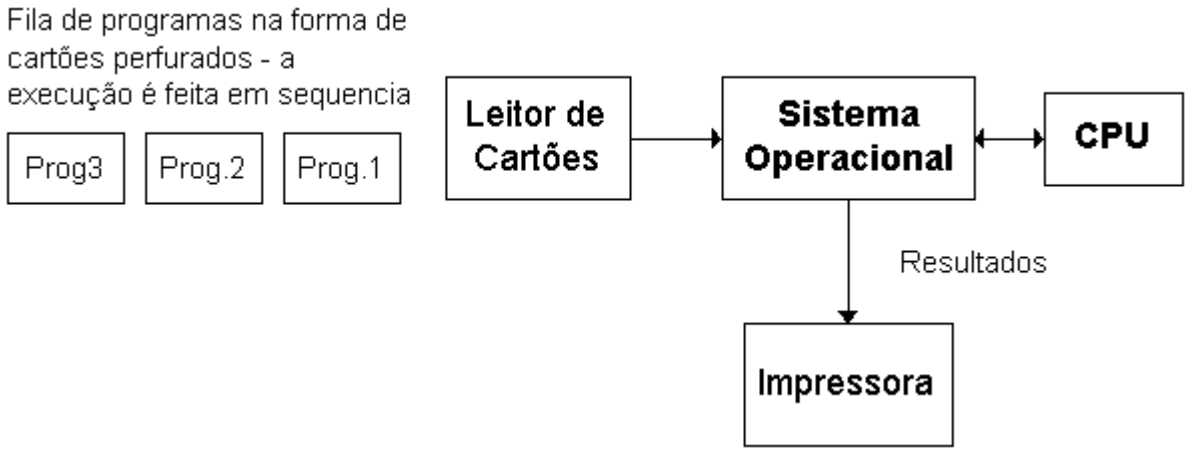


Figura 8: Processamento em lote - não há interação direta

Já na década de 1960 diversos avanços levaram ao desenvolvimento de terminais interativos que permitiam aos usuários acessar o computador central (*mainframe*) através de linhas de comunicação. Os usuários passaram a ter um mecanismo para interação direta com o computador. Na mesma época os avanços nas técnicas de processamento originaram os sistemas de tempo compartilhado (*time-sharing*) como pode ser visto na Figura 9. Nesse tipo de computação, várias tarefas de usuários diferentes ocupavam o computador central ao mesmo tempo efetuando-se um revezamento no tempo do processador. Na década de 1970 deu-se o

desenvolvimento dos mini e microcomputadores popularizando e distribuindo o poder computacional nas organizações (Soares et al., 1995).

Na década de 1980, com os extraordinários avanços de *hardware* para microcomputadores e a popularização dos mesmos, o poder de processamento antes restrito somente aos *mainframes* passou a estar disponível nas estações de trabalho. No final da década de 1980 e começo de 1990 os microcomputadores apresentavam uma relação custo/benefício bastante favorável em comparação com os minicomputadores e *mainframes* (Tanenbaum, 1997).

Segundo Bochenski (1995), para aproveitar o poder de processamento dos microcomputadores que já estavam disponíveis, diversas empresas efetuaram o *downsizing*[20] de seus sistemas que costumavam rodar em *mainframes* para uma arquitetura cliente/servidor utilizando microcomputadores em uma rede local.

3.4.2 Arquitetura cliente/servidor

> "Os componentes básicos de sistemas cliente/servidor consistem em uma parte cliente, uma parte servidor e uma parte rede. O *software* localizado no cliente de um sistema cliente/servidor é normalmente chamado de *software front-end*, e o *software* localizado no servidor é chamado *software back-end*."
>
> (BOCHENSKI, 1995, p. 25)

Tanenbaum (1997) explica que no modelo cliente/servidor, a comunicação se dá através de uma mensagem de solicitação do cliente enviada para o servidor, pedindo para que alguma tarefa seja executada. Em seguida, o servidor executa essa tarefa e envia a resposta para o cliente. Em geral há muitos clientes acessando serviços de um pequeno número de servidores. Um esquema da arquitetura cliente/servidor pode ser visto na Figura 10.

---

[20] Bochenski (1995) explica que em informática o termo *downsizing* refere-se ao processo de transferência de um sistema aplicativo que está rodando em uma máquina maior para outra menor e mais barata.

Um dos aspectos mais importantes da computação cliente/servidor é que ela tira o máximo de vantagem de vários recursos da informática tais como o aumento do poder de processamento das estações de trabalho, as interfaces gráficas de usuário e as redes. Outro fator fundamental é a viabilização do compartilhamento de recursos entre sistemas e plataformas diferentes diminuindo as restrições geográficas e os custos (Bochenski, 1995).

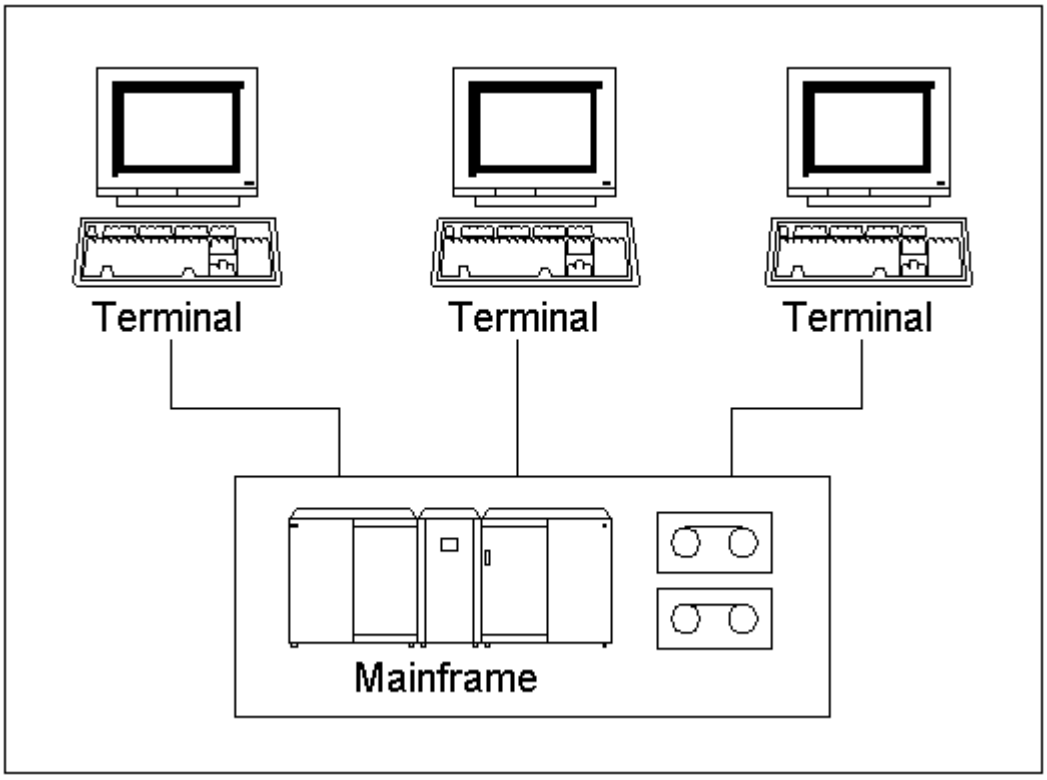


Figura 9: Sistema de tempo compartilhado (*time-sharing*)
Fonte: Adaptado de Soares et al. (1995)

3.4.3 Cliente ou *front-end*

O usuário final do sistema cliente/servidor interage diretamente no lado do cliente (*front-end*) com recursos de interface gráfica, acessando os diversos serviços oferecidos pela aplicação. É importante aplicar um guia de estilos mantendo a homogeneidade da aparência da interface de usuário ao longo das diversas partes da aplicação.

Em geral o *front-end* de uma aplicação cliente/servidor é apresentado na forma de interfaces gráficas de usuário (GUI – *Graphics User Interface*). As GUI´s

contribuíram para os sistemas cliente/servidor porque os tornaram mais fáceis de serem usados e aumentaram a produtividade. Os recursos de GUI, como multitarefa, computação de tarefas e intercâmbio de dados entre aplicativos, também contribuíram para que os sistemas ficassem mais atraentes (Bochenski, 1995).

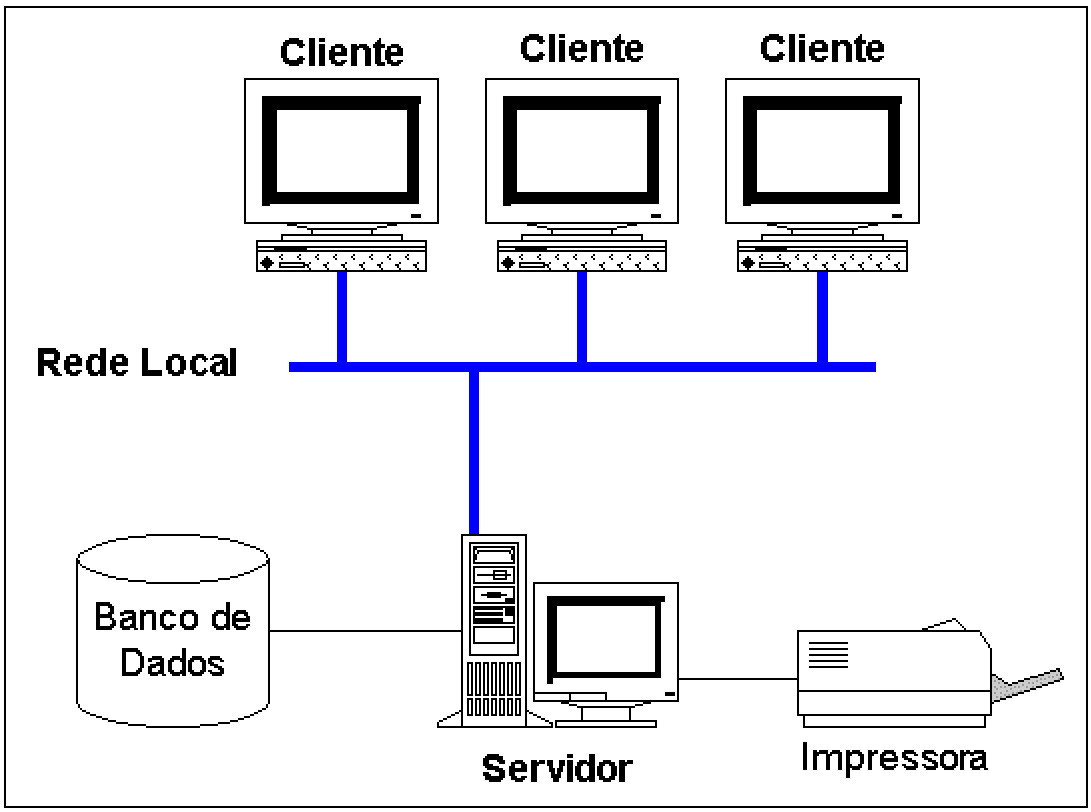


Figura 10: A rede é parte integral da arquitetura Cliente/Servidor.
Fonte: adaptado de Bochenski (1995)

Uma forma especial de *front-end* é a das aplicações *web*. Na sua forma mais tradicional, o lado cliente da aplicação *web* não efetua processamentos pesados sendo responsável apenas pela apresentação dos documentos que são baixados dos servidores *web*. Por outro lado, nas aplicações *web* modernas observa-se a utilização de objetos distribuídos como Java *applets* e componentes ActiveX para viabilizar uma forma mais interativa e com maior poder de processamento no cliente (Orfali et al., 1999).

3.4.4 Servidor ou *back-end*

Para Bochenski (1995, p.27), um “servidor é um dispositivo que oferece um serviço”. Os servidores mais comuns são os de arquivo, aplicativos, correio, de rede e

principalmente os servidores de banco de dados que são de interesse direto para este trabalho. Bochenski define servidores de banco de dados *back-end* como sistemas de gerenciamento de banco de dados (SGBDs) projetados especificamente para operar em servidores ou adaptados para servidores.

Soares et. al (1995) afirmam que as aplicações que precisam efetuar acesso a banco de dados podem executar um SGBD no cliente utilizando um servidor de arquivos como repositório dos arquivos do banco de dados. Esta abordagem dificulta a manutenção da integridade do banco de dados e diminui bastante o desempenho da rede. Como ilustrado na Figura 11, este baixo desempenho decorre do fato de que o cliente requisita ao servidor os arquivos do banco de dados na íntegra e portanto é necessário passar todos os registros pela rede para processamento no cliente.

Na prática, utiliza-se uma abordagem alternativa que consiste na execução de funções como processamento de consultas, controle de concorrência e manutenção de integridade num servidor de banco de dados que viabiliza essas tarefas através do processamento orientado a transações. Com esta solução há um aumento considerável no desempenho da rede uma vez que o processamento de consultas se dá no servidor e somente os registros que serão utilizados pela aplicação são transferidos pela rede. Nesta segunda abordagem o cliente passa uma consulta SQL que é executada no servidor retornando somente os resultados tal como ilustrado na Figura 12.

Soares et al. (1995) afirmam que além do aumento de desempenho da rede e consequentemente de todas as aplicações que utilizam a rede, essa abordagem permite concentrar os investimentos de *hardware* no servidor (processador, memória, *cache* etc). Como resultado, tem-se uma velocidade de processamento maior do que no caso onde o processamento das consultas se dá no cliente, que em geral é uma máquina com menor capacidade de processamento que o servidor.

Bochenski (1995) afirma que em sistemas cliente/servidor típicos o SQL é um denominador comum. Devido à natureza declarativa das instruções SQL, o cliente

não precisa se preocupar com a forma pela qual o servidor acessa os dados bastando apenas requisitar “quais” dados ele deseja.

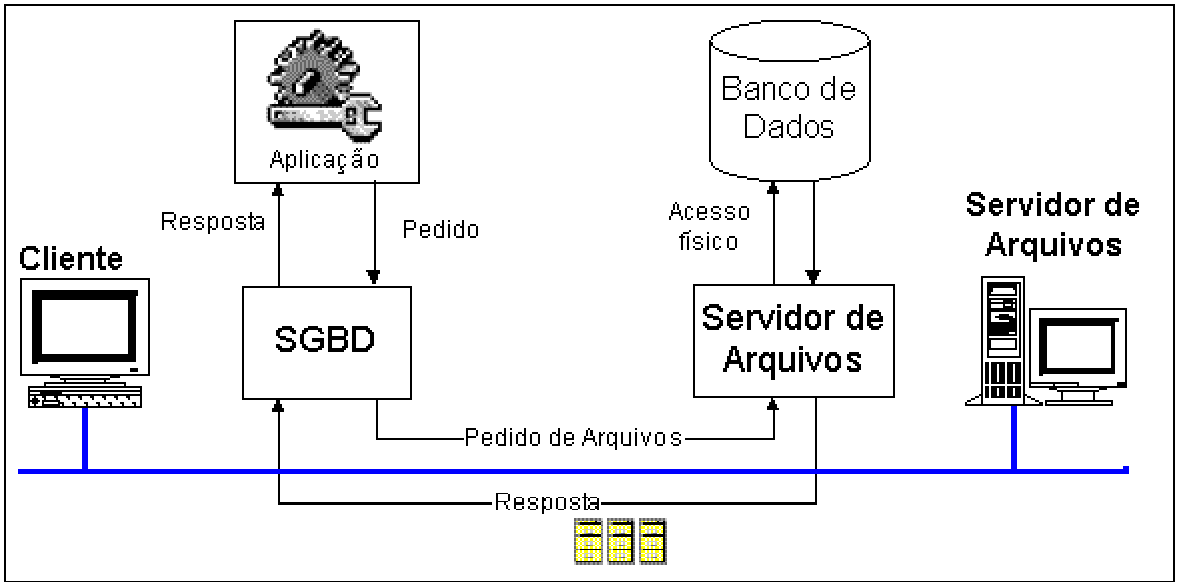


Figura 11: Gerenciamento de banco de dados com servidor de arquivos

Fonte: adaptado de Soares et al. (1995).

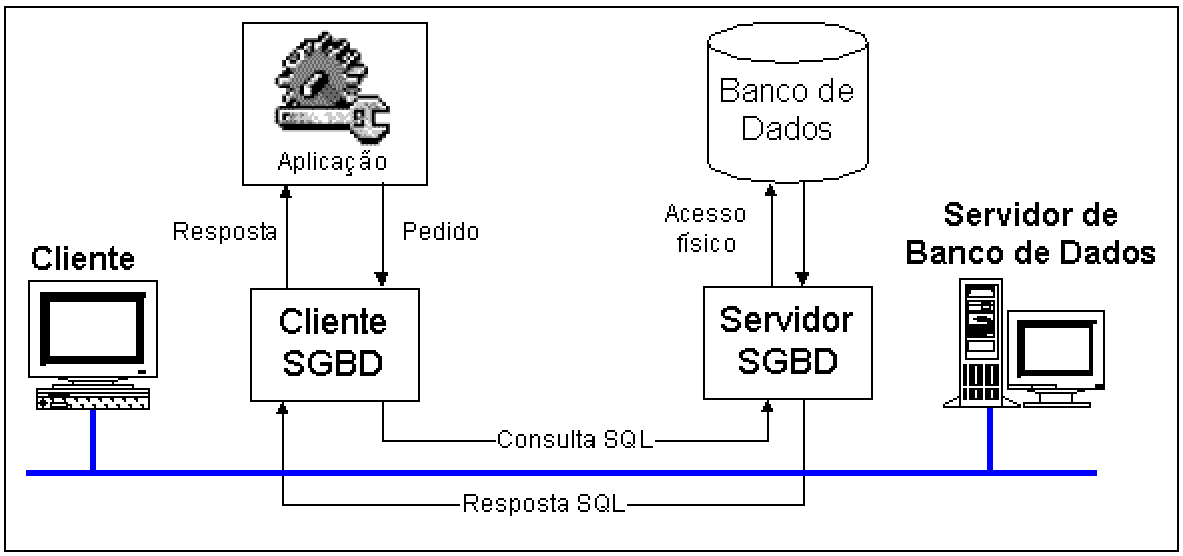


Figura 12: Servidor de Banco de dados

Fonte: adaptado de Soares et al. (1995).

3.4.5 Middleware

Orfali et al. (1999) apresentam uma analogia interessante e afirmam que utilizando os três “blocos fundamentais” da arquitetura cliente/servidor (o cliente, o servidor e o

middleware) pode-se construir desde arquiteturas simples até os sistemas mais complexos. Nesta analogia, o *middleware* é o *software* que liga o cliente ao servidor e é representado pela barra "/" na arquitetura cliente/servidor. O *middleware* é o sistema nervoso dos sistemas cliente/servidor e sua execução é feita em parte pelo cliente e em parte no servidor.

Segundo Bochenski (1995), "*middleware*" é um termo usado de forma diferente em diferentes contextos. De uma maneira geral os autores usam o termo para referir-se ao *software* que conecta dois módulos, permitindo que esses módulos comuniquem-se entre si. Por exemplo, um cliente pode acessar um servidor de banco de dados utilizando instruções SQL que, neste contexto, funciona como *middleware*.

Orfali et al.(1999) confirmam que *middleware* é um termo vago que cobre todo o *software* distribuído necessário para viabilizar interações entre clientes e servidores. Ele começa com o conjunto da API[21] no lado do cliente que é utilizada para chamar um serviço e se estende como a transmissão e a resposta dessa requisição na rede. O *middleware* não inclui o *software* do servidor que provê o serviço propriamente dito nem tampouco o banco de dados. No *front-end* o *middleware* não inclui a interface do usuário que faz parte do domínio da aplicação no cliente. Em arquiteturas mais complexas, com vários servidores, o *middleware* inclui o *software* utilizado para controlar as interações de servidor para servidor.

3.4.6 Camadas

As primeiras aplicações computacionais nas décadas de 1960 e 1970, executavam em sistemas do tipo *timesharing* monolíticos, ou seja, havia somente um processador do computador central (*mainframe*) que executava todas as tarefas sob a ordem do compartilhamento de tempo.

---

[21] Application Programmer Interface (API) – Interface de Programação para Aplicação – é o termo utilizado para referir-se às interfaces das bibliotecas de código reutilizável que disponibiizam diferentes funcionalidades aos programadores. No caso específico, refere-se à API utilizada no lado do cliente para acessar serviços remotos do servidor.

A popularização a nível mundial dos computadores ocorreu com o advento dos computadores pessoais e o aumento de desempenho das *workstations* no final da década de 1970 e durante a década de 1980. Nesta época, o uso dos computadores era tipicamente para a execução de aplicativos *stand-alone* como processadores de texto, planilhas etc.

Já no contexto das redes de computadores, a configuração espacial mínima das aplicações é a chamada arquitetura de duas camadas, conforme a Figura 13, que é uma representação fiel da arquitetura cliente/servidor típica. Nas aplicações clássicas de sistemas de informação cliente/servidor, o *front-end* é responsável por uma parte significativa do processamento ao passo que nas aplicações *web* típicas o cliente é responsável apenas pela tarefa de apresentação que é considerada “leve”. Dessa forma, a lógica e os dados da aplicação residem no lado do servidor (Fraternali, 1999).

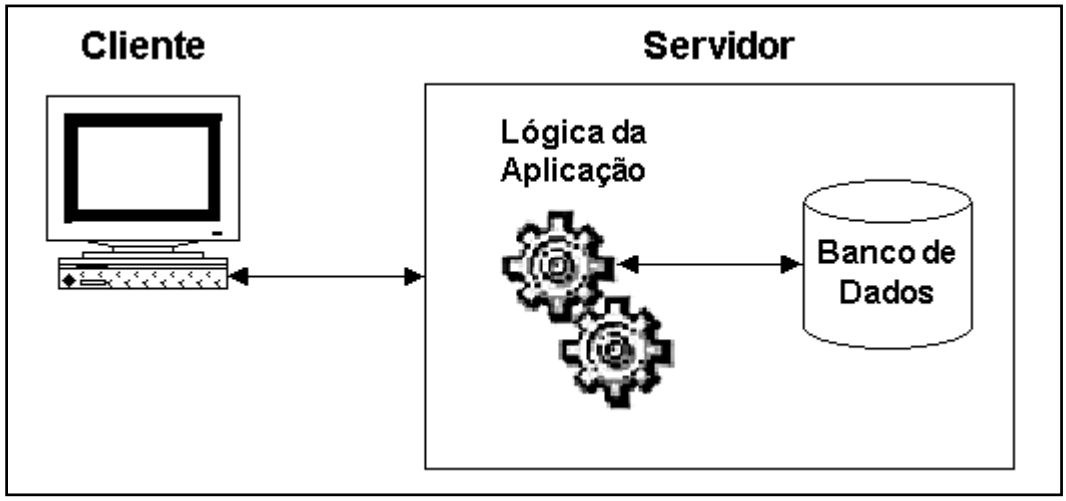


Figura 13: Arquitetura de duas camadas
Fonte: adaptado de Fraternali (1999)

Apesar desse ser o caso mais comum, atualmente cada vez mais aplicações *web* estão efetuando parte importante do processamento no lado do cliente. No capítulo 4 descreve-se o caso particular deste trabalho onde uma parte significativa do processamento é realizada no cliente.

A simplicidade de implementação de sistemas com arquitetura de duas camadas impulsionou a sua popularidade, que culminou com o desenvolvimento de diversos sistemas cliente/servidor no começo da década de 1990. Apesar dessa facilidade de desenvolvimento, a arquitetura de duas camadas é insuficiente para sistemas maiores e mais complexos (Orfali et al., 1999).

Uma configuração mais avançada, chamada de arquitetura de três camadas ou multi-camadas conforme a Figura 14, introduz uma distinção de funções no lado do servidor. A presença de uma ou mais camadas de aplicação possibilita a implementação de uma arquitetura avançada que integra o protocolo HTTP e os protocolos de distribuição de aplicações cliente/servidor resultando em maior desempenho, escalabilidade, confiabilidade e segurança (Fraternali, 1999).

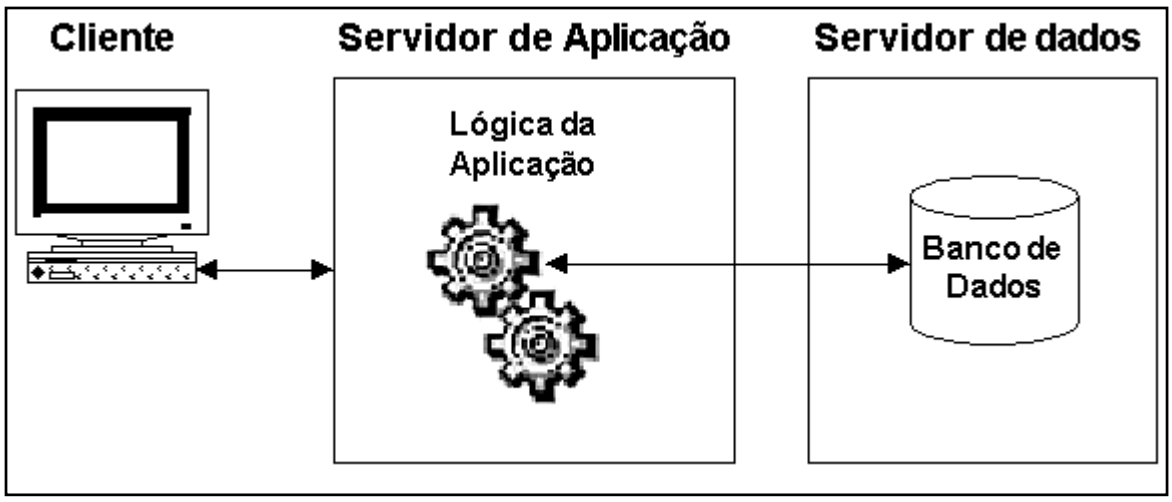


Figura 14: Arquitetura de três camadas
Fonte: adaptado de Bochenski (1999)

Para Orfali et al. (1999), uma das maiores vantagens da arquitetura de três ou mais camadas é que as aplicações minimizam as transferências pela rede através da criação de níveis abstratos de serviço. No lugar de interagir com o banco de dados diretamente, o cliente chama a lógica da aplicação no servidor que por sua vez acessa o banco de dados em nome do cliente. Consequentemente há ganho de eficiência na arquitetura de três camadas visto que para efetuar uma tarefa há apenas uma comunicação cliente/servidor na qual efetuam-se diversas consultas SQL no *back-end* para posteriormente retornar os resultados para o *front-end*.

Outra forma que vem se desenvolvendo muito nos últimos anos é a computação distribuída. Bochenski (1995) afirma que há consenso a respeito da arquitetura cliente/servidor ser uma forma de processamento distribuído pois os vários componentes desta arquitetura estão em locais separados e há interação entre eles. Entretanto, outros autores afirmam que para caracterizar-se como processamento distribuído um sistema cliente/servidor deve ter vários clientes interagindo com vários servidores e deve haver transparência quanto à localização dos diversos serviços.

Quando os pacotes de bancos de dados distribuídos estiverem mais desenvolvidos, eles permitirão o acesso transparente aos dados na rede. O gerenciador de banco de dados distribuído mantém informação a respeito da localização dos dados na rede e faz o roteamento das requisições para os nós de banco de dados específicos tornando a sua localização transparente (Orfali et al.,1999).

Orfali et al. (1999) afirmam que está havendo uma nova revolução na qual a arquitetura cliente/servidor departamental com apenas um servidor baseado em rede local está sendo substituída por uma forma cliente/servidor multi-camadas onde cada máquina na "*information highway*" pode ser tanto um cliente quanto um servidor.

3.5 Considerações finais

A gama de tecnologias para rede é muito variada e não há um consenso a respeito de qual solução é mais adequada para um determinado tipo de aplicação. Devido a aspectos financeiros, necessidade de manter compatibilidade com sistemas legados e a restrições de plataforma de *hardware* e *software, h*á inúmeras soluções adotadas por diferentes provedores de serviço e informação, mesmo quando as aplicações são bastante similares.

É inviável adotar todas as novas tecnologias e padrões que surgem muito rapidamente - o estado da arte - como parte da solução final de uma aplicação *web*.

Entretanto, é importante fixar a arquitetura do sistema em algum momento do projeto da aplicação, integrando as diversas tecnologias para viabilizar a implementação sem perder de vista as características de extensibilidade e escalabilidade da aplicação.

Neste capítulo apresentou-se um histórico e uma revisão das redes de computadores e das tecnologias recentes da Internet, em especial da *World Wide Web*. A utilização da WWW como meio para diversos tipos de sistemas de Educação à Distância é bastante conhecida. Atualmente, a WWW abriga democraticamente desde simples versões eletrônicas de documentos e tutoriais até bibliotecas digitais com grande volume de informação, ambientes colaborativos de ensino-aprendizagem e sistemas hipermídia com alto grau de interação. No próximo capítulo será apresentado o *framework* para gerenciar dados de interação do usuário em ambientes hipermídia de aprendizagem.

4. FRAMEWORK

4.1 Considerações iniciais

> "Enquanto a infra-estrutura global de informação se expande a taxas sem precedentes, estão ocorrendo mudanças dramáticas no tipo de pessoa que acessa os sistemas de informação. Desde as grandes corporações até os indivíduos estão atrelados a atividades que envolvem o acesso a banco de dados, e o sustento e/ou competitividade dependem fortemente da eficiência deste acesso."
>
> (SILBERSCHATZ & STONEBRAKER apud MYERS et al., 1996, p.800)

Tal universalização do acesso aos sistemas de informação que ocorreu nas últimas décadas e obteve taxas de crescimento sem precedentes com o advento da Internet gerou uma demanda por parte de usuários leigos. Conseqüentemente, tornou-se necessário o desenvolvimento de interfaces voltadas ao usuário que não exijam maiores conhecimentos técnicos e ao mesmo tempo permitam o acesso a uma grande variedade de informações e operações.

De acordo com Fraternali (1999), do ponto de vista do cliente a usabilidade é a qualidade mais importante de uma aplicação *web*. É possível identificar um conjunto de critérios para usabilidade em aplicações *web*:

- o grau da qualidade visual que indica a coerência global da apresentação e da metáfora de navegação bem como a qualidade individual dos recursos gráficos;
- o grau de customização que mede a facilidade de criar a interface da aplicação para usuários individuais ou para um grupo de usuários. As aplicações podem ter uma interface fixa, mas por outro lado, o conteúdo, a apresentação e a navegação podem ser individualmente personalizados;
- o grau de adaptação que é proporcional à flexibilidade da interface em tempo de execução. A aplicação pode ser imutável, mas também pode manter registro das atividades do usuário e adaptar-se conforme necessário;
- o grau de pró-atividade que indica a capacidade que uma aplicação tem de interagir com o usuário por iniciativa própria (da aplicação). As aplicações podem

ser passivas, i.e., fornecem informações somente em resposta a um pedido do usuário, ou pró-ativas quando são capazes de enviar informações para o usuário assim que um conjunto de eventos relevantes ocorrer.

Essa área de pesquisas relativa à usabilidade pertence ao ramo da ergonomia de interfaces homem-computador (IHC), que não é o foco desse trabalho. Recomenda-se como referência a leitura de Nielsen (1993) e Cybis (2003). Apesar de não ser a meta principal desse trabalho, o modelo aqui proposto tem entre outras finalidades, o objetivo de funcionar como base para a pesquisa na área de IHC. É possível efetuar uma análise detalhada das variáveis de interação do usuário com o ambiente que pode indicar, por exemplo, a necessidade de refinamentos na interface ou na proposição do conteúdo.

Para Ulbricht (1997), a modelagem de um ambiente hipermídia para a construção do conhecimento deve levar em consideração as seguintes variáveis:

- número de acertos e erros;
- número de retornos impostos pelo sistema e de retornos voluntários do aluno;
- tempo de permanência nas páginas, tempo total que separa duas respostas diferentes;
- comprometimento do aluno com o processo de aprendizagem;
- construção do conhecimento através do número crescente de acertos.

Tal sistema de EIAC (Ensino Inteligente Assistido por Computador) deve tratar as diferenças individuais dos aprendizes, fazendo uma análise detalhada dessas variáveis (Ulbricht, 1997).

Este trabalho está associado à pesquisa e desenvolvimento de aplicações de ensino-aprendizagem no laboratório HIPERLAB/EGR/UFSC. Neste contexto, procura-se dotar essas aplicações de EIAC de um certo grau de inteligência artificial. Não basta simplesmente apresentar os conteúdos e deixar o aprendiz navegando sem um objetivo definido.

A proposta desse trabalho é criar um f*ramework* para armazenagem e recuperação de variáveis de interação do usuário com ambientes hipermídia na plataforma WWW. O usuário deverá se cadastrar no sistema que por sua vez manterá o registro de todas as ações relevantes do usuário tais como:

- as visitas realizadas a cada página;
- o momento de início e fim de cada visita;
- os eventos disparados pelo usuário (tais como visitas ao glossário);
- as respostas confirmadas a cada pergunta bem como o momento em que ocorreram;
- o número de vezes em que o usuário clicou em cada uma das opções de resposta;
- as anotações do usuário.

A partir desses dados pode-se chegar a informações para um usuário ou grupo de usuários tais como:

- a ordem de visita às páginas, um histórico de visitas;
- a ocorrência de retorno às páginas;
- a duração das visitas;
- a última página visitada;
- as últimas respostas confirmadas a cada questão (o mesmo usuário pode voltar a uma página com pergunta e alterar sua resposta prévia);
- o número de sessões (retornos ao sistema);
- tempo total de interação com o sistema.

No contexto desse trabalho, persistência é a capacidade do *framework* de manter esses dados e informações ao longo do tempo. Por exemplo, após cadastrar-se no sistema e acessar algumas páginas de conteúdo, um usuário decide sair do sistema. Mais tarde, ao acessar o ambiente novamente em uma seção diferente, é necessário que o sistema “saiba” qual a última página que aquele usuário visitou para poder redirecionar a navegação para esta página. Considerando que o usuário já havia respondido a uma determinada pergunta na primeira seção, também é necessário que ao entrar na página com essa pergunta, a opção que ele escolheu

anteriormente já esteja preenchida. Dessa forma garante-se a persistência, ou seja, mantém-se um estado consistente das interações do usuário com o sistema.

Para viabilizar as diversas funcionalidades desse *framework* e prevendo futuros refinamentos, é necessário utilizar uma arquitetura de *software* capaz de cumprir esses requisitos.

4.2 Arquitetura do sistema

A arquitetura básica da WWW é a de duas camadas, como pode ser visto na Figura 6 da página 35. A navegação do usuário cliente no navegador (clicando em *links*, acessando endereços específicos) gera uma demanda por conteúdo que é transformada em uma requisição HTTP. O servidor acessa o conteúdo na forma de documentos HTML e mídias que por sua vez é transferido pela rede e apresentado no navegador. As novas interações de navegação do usuário geram novas demandas e o ciclo repete-se de forma iterativa.

A arquitetura do *framework* aqui proposto é de três camadas que são representadas dentro dos retângulos principais como pode ser visto na Figura 15: camada cliente, camada de servidor de aplicação e a camada do servidor de banco de dados. Todos os elementos que ficam entre as camadas e portanto fora dos retângulos, constituem o *middleware*: HTTP, XML, COM, ActiveX, OLE DB. As principais funcionalidades do *middleware* utilizado neste modelo são abordadas no item 4.3.2.

As camadas cliente e servidor de aplicação seguem a arquitetura básica da WWW para as requisições HTTP de conteúdo estático. No *framework*, assume-se que a mesma arquitetura é utilizada para a requisição e apresentação de conteúdo dinâmico, conforme detalhado no item 3.3.2.

Além dessa forma de comunicação cliente/servidor baseada na arquitetura WWW clássica, utilizou-se um tipo diferente de requisição HTTP entre as camadas cliente e servidor de aplicação: a transferência de dados intermediários em formato XML. Essa comunicação será vista em detalhes no item 4.3.2.

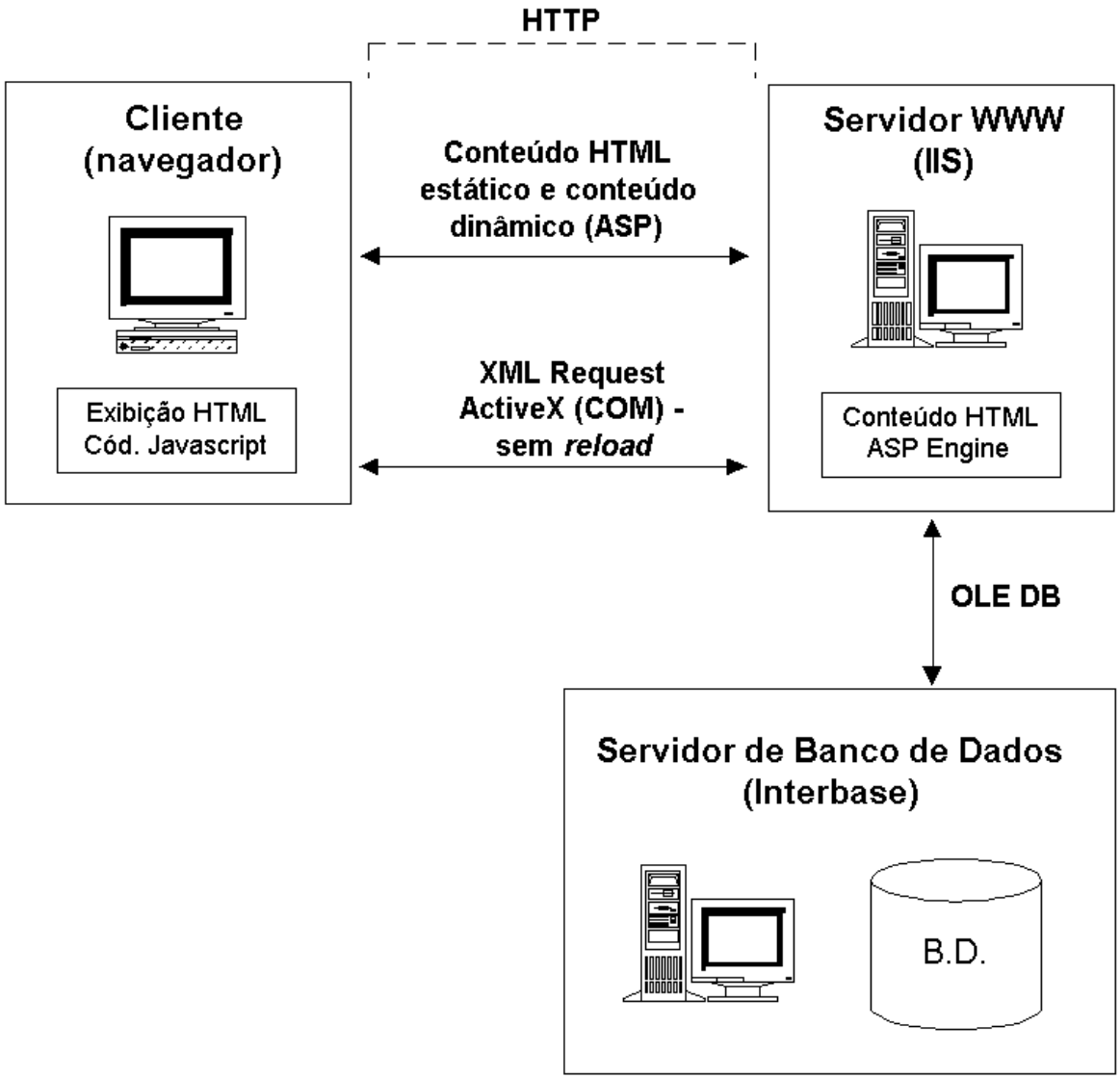


Figura 15: Arquitetura de três camadas do framework: Cliente, Servidor de Aplicação WWW e Servidor de Banco de Dados.

O acesso ao banco de dados - comunicação entre a camada de aplicação e o servidor de banco de dados – é feito através de OLE DB e é descrito no item 4.3.4.

4.3 Software

A implementação dessa arquitetura envolve diferentes tecnologias e linguagens de acordo com a camada em que se encontra o *software*. É importante distinguir, portanto, o escopo de aplicação de cada tecnologia.

4.3.1 Código no Cliente

As camadas cliente e servidor de aplicação são baseadas na arquitetura WWW e portanto utilizam diversas tecnologias associadas a *web*. No caso do cliente, para a apresentação de conteúdo emprega-se o HTML e suas extensões tais como o Shockwave, CSS etc.

Uma característica importante dessa arquitetura é que a cada requisição por um documento HTML há um *roundtrip*[22] completo e o *reload* (recarregamento) da página no cliente. Este recarregamento implica em substituir o documento HTML que estava sendo apresentado no cliente no momento da requisição pelo documento que foi requisitado.

Por esse motivo, é necessário utilizar a técnica de *frames* escondidos para manter o código (e a memória) ativos, ou seja, muda o conteúdo mas o código JScript, seus objetos e variáveis permanecem na memória. Isto é fundamental para garantir a funcionalidade da persistência tanto no monitoramento das interações da sessão atual quanto para o controle das respostas que o usuário deu em sessões anteriores.

Para efetuar as tarefas que exigem processamento, utilizou-se a linguagem JScript. O código JScript do *framework* está distribuído em diferentes arquivos .JS e provê uma biblioteca de funções para lidar com diversas tarefas tais como:

- navegação linear e não-linear;
- monitoramento das variáveis de usuário;
- controle da janela de Glossário e seus eventos;
- tratamento das variáveis de usuário em estruturas de dados apropriadas;
- eventos de inicialização e saída das páginas;
- leitura de dados anteriores do banco de dados;
- apresentação consistente de respostas de seções prévias;

---

[22] O *roundtrip* é uma volta completa no ciclo de comunicação cliente/servidor: o cliente requisita um recurso ao servidor que por sua vez envia-o para o cliente.

- direcionamento do usuário para a tela de destino de acordo com a resposta dada;
- tratamento de erros;
- conversão de variáveis em documentos XML e vice-versa;
- tratamento de documentos XML através do XML DOM[23];
- lidar com o objeto ActiveX XMLHTTPRequest (ver item 4.3.2).

Essa ampla gama de funcionalidades sendo executadas no *front-end* não é comum na maioria dos sistemas de três camadas. No entanto, este fato justifica-se visto que a natureza do sistema aqui proposto envolve o monitoramento de interações do usuário com o ambiente, que ocorrem exatamente na camada cliente do sistema.

4.3.2 Middleware

Conforme abordado anteriormente, para Bochenski (1995) "*middleware*" é o *software* que conecta dois módulos, permitindo que eles se comuniquem entre si. Este *software* é executado tanto pelas camadas que solicitam os serviços quanto pelos servidores.

Dentro da arquitetura aqui proposta, as comunicações HTTP entre cliente e servidor, os objetos ActiveX[24] para requisições HTTP diferenciadas e a tecnologia de acesso a dados OLE DB[25] funcionam como *middleware*.

Nas aplicações *web* modernas observa-se a utilização de objetos distribuídos como Java *applets* e componentes ActiveX para viabilizar uma forma mais interativa e com maior poder de processamento no cliente (Orfali et al., 1999). No caso do *framework*

---

[23] Document Object Model (DOM) – Modelo para objetos de documentos – é um padrão da W3C (consórcio de padronização da WWW) que especifica os objetos relacionados aos padrões HTML e XML. Para o XML, o DOM especifica diversos objetos para tratamento dos nós, documentos, elementos, atributos, entidades, parser etc. (Marchal, 2000).

[24] O padrão ActiveX é abordado no item 4.3.4.

[25] O OLE DB e outras tecnologias Microsoft serão abordadas no item 4.3.4.

aqui proposto, empregou-se a tecnologia ActiveX da Microsoft para resolver duas tarefas específicas.

A primeira é garantir a funcionalidade XML em documentos inseridos no HTML. O texto XML com uma identificação é reconhecido como um objeto ActiveX (um *XML data island* - ilha de dados XML) que possui as propriedades e métodos especificados no padrão DOM. Os dados contidos nesses documentos XML mantêm informações sobre as interações do usuário tanto da sessão atual quanto em sessões anteriores. O código JScript no cliente é encarregado de tratar esses dados fazendo acesso aos atributos e métodos dos objetos ActiveX que implementam o DOM para XML.

Outra tarefa importante viabilizada por um componente ActiveX é a de agilizar a comunicação com o servidor de aplicação através do objeto XMLHTTPRequest. Este componente provê um tipo diferente de requisição HTTP que não segue o processo típico da WWW. A finalidade dessa comunicação é a transferência de dados na forma de XML entre o cliente e o servidor de aplicação.

O fluxo de dados funciona da seguinte maneira: o *software* do cliente monitora as interações do usuário com o sistema e mantém os dados internamente através do JScript. No momento em que o usuário sai do sistema, é necessário gravar esses dados de histórico. Para tal, o cliente cria via JScript o objeto ActiveX "Msxml2.XMLHTTP" no navegador. Esse componente permite fazer uma requisição HTTP, enviando um documento XML para uma página ASP no servidor.

Por sua vez, a página ASP no servidor WWW recebe o documento XML num objeto idêntico ao do cliente garantindo um tratamento padronizado dos documentos XML no cliente e no servidor. Na seqüência, o servidor efetua a tradução do documento XML para uma série de instruções SQL que formarão um *script* SQL. Por fim, o código SQL é executado no servidor de banco de dados através do OLE DB.

Ao entrar novamente no sistema, utiliza-se a mesma forma de comunicação para recuperar as informações do banco de dados. Para garantir a persistência é

necessário recuperar as últimas respostas que o usuário atribuiu em sessões anteriores para cada pergunta.

O servidor WWW começa o fluxo inverso solicitando ao servidor de banco de dados a execução da consulta SQL detalhada no item 4.4.4. Os dados com as últimas respostas do usuário são retornados ao servidor WWW que por sua vez codifica-os na forma de um objeto XML *data island* e encaminha para o navegador no cliente.

A vantagem dessa abordagem é que através do objeto ActiveX "Msxml2.XMLHTTP" é possível fazer um *roundtrip* completo sem a necessidade de um recarregamento no navegador.

Para exemplificar, segue um trecho de um documento XML com diversas interações de usuário efetuadas sobre o sistema Geometrando. Esse documento XML funciona como uma forma de intercâmbio de dados entre as camadas do sistema. Pode-se extrair informações a respeito do usuário, das páginas que ele visitou, das respostas que deu, dos eventos que gerou etc.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE geometrando SYSTEM "geometra.dtd">
<geometrando>
   <usuario cod="1">
      <login>claudiof</login>
      <nome>Claudio Luiz Ferreira</nome>
      <ultima_pagina>../cubo/cubo_d070.html</ultima_pagina>
   </usuario>
   <paginas>
      <pagina>
         <strNome>cubo_d070</strNome>
         <tsEntrada>11/25/2002 17:29:37.882</tsEntrada>
         <strTempo>00:00:03.505</strTempo>
         <pergunta>
            <nome>cubo_d070</nome>
            <tipo>radio</tipo>
            <respCorreta>2</respCorreta>
            <resposta num="1">
```

```
        <strRespUser>1</strRespUser>
        <strClicadas>1,0</strClicadas>
        <timestamp>11/25/2002 17:29:41.387</timestamp>
      </resposta>
    </pergunta>
  </pagina>
  <pagina>
    <strNome>cubo_d071</strNome>
    <tsEntrada>11/25/2002 17:29:41.797</tsEntrada>
    <strTempo>00:00:23.514</strTempo>
    <pergunta>
      <nome>cubo_d071</nome>
      <tipo>radio</tipo>
      <respCorreta>2</respCorreta>
      <resposta num="1">
        <strRespUser>2</strRespUser>
        <strClicadas>1,1</strClicadas>
        <timestamp>11/25/2002 17:30:05.311</timestamp>
      </resposta>
    </pergunta>
  </pagina>
  <pagina>
    <strNome>cubo_d072</strNome>
    <tsEntrada>11/25/2002 17:30:05.712</tsEntrada>
    <strTempo>00:00:12.167</strTempo>
    <evento>
      <tipo>ABRE_GLOSSARIO_NO_ITEM</tipo>
      <alvo>poligono_plano</alvo>
      <timestamp>11/25/2002 17:30:07.654</timestamp>
    </evento>
  </pagina>
  <pagina>
    <strNome>cubo_d070</strNome>
    <tsEntrada>11/25/2002 17:30:18.210</tsEntrada>
    <strTempo>00:00:02.443</strTempo>
    <pergunta>
      <nome>cubo_d070</nome>
      <tipo>radio</tipo>
      <respCorreta>2</respCorreta>
      <resposta num="1">
        <strRespUser>1</strRespUser>
```

```
            <strClicadas>1,0</strClicadas>
            <timestamp>11/25/2002 17:30:20.653</timestamp>
          </resposta>
        </pergunta>
      </pagina>
   </paginas>
</geometrando>
```

É importante ressaltar que o distanciamento das margens - tabulação - ajuda a visualizar a relação de pertinência dos elementos. Dessa forma, a marcação <ultima_pagina> pertence ao elemento <usuario> e representa a última página visitada pelo usuário. Essa informação é importante para o redirecionamento do usuário numa nova sessão de interação com o ambiente.

Pode-se observar também que cada visita, resposta e evento mantém um atributo *timestamp,* ou seja, a data (dia, mês e ano) e momento exato (hora, minuto, segundo e milisegundo) em que ocorreram. Essa informação é de vital importância para diversas operações internas do sistema (e.g. definir qual foi a última resposta dada para uma determinada pergunta). A partir do *timestamp*, pode-se definir a ordem cronológica dos acontecimentos, informação que pode ser de grande valia para os sistemas inteligentes que processam as variáveis de usuário.

Uma outra informação que pode ser extraída é o número de “clicadas” que o usuário dá sobre cada item de resposta. Uma vez que a maioria das perguntas é do tipo múltipla escolha, é possível saber se o usuário chegou a mudar sua resposta ou se sua primeira escolha foi a resposta final. Essa informação, cruzada com a duração da visita àquela página pode ser bastante útil para determinar o comprometimento do aluno com o processo de aprendizagem.

Além dos documentos XML de intercâmbio de dados, definiu-se também o DTD para validar esses documentos. A partir dessa especificação formal, pode-se definir quais elementos e atributos são necessários, de que tipo são os seus valores bem como a ordem que devem aparecer. Segue a especificação formal DTD usada para o

ambiente Geometrando (arquivo “Geometra.DTD” referenciado no documento XML anterior).

```
<!ELEMENT geometrando (user, paginas)>
<!ELEMENT usuario (login, nome, ultima_pagina)>
<!ATTLIST usuario
   cod CDATA #REQUIRED
>
<!ELEMENT login (#PCDATA)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT ultima_pagina (#PCDATA)>
<!ELEMENT paginas (pagina+)>
<!ELEMENT pagina (strNome, tsEntrada, strTempo, evento*, pergunta*)>
<!ELEMENT strNome (#PCDATA)>
<!ELEMENT tsEntrada (#PCDATA)>
<!ELEMENT strTempo (#PCDATA)>
<!ELEMENT evento (tipo, alvo, timestamp)>
<!ELEMENT tipo (#PCDATA)>
<!ELEMENT alvo (#PCDATA)>
<!ELEMENT timestamp (#PCDATA)>
<!ELEMENT pergunta (nome, tipo, respCorreta, resposta*)>
<!ELEMENT respCorreta (#PCDATA)>
<!ELEMENT resposta (strRespUser, strClicadas, timestamp)>
<!ATTLIST resposta
   num CDATA #REQUIRED
>
<!ELEMENT strRespUser (#PCDATA)>
<!ELEMENT strClicadas (#PCDATA)>
```

A descrição detalhada do modelo formal de definição de documentos foge ao escopo desse trabalho. Recomenda-se como referência a leitura de Harold (1999) e Marchal (2000).

4.3.3 Código no Servidor

O gerenciamento de todas as requisições HTTP no servidor de aplicação é realizado pelo *software* servidor de WWW, no caso o Internet Information Server (IIS) que é distribuído com o sistema operacional Microsoft Windows NT Server. A programação

no lado do servidor de aplicação é feita com o Active Server Pages (ASP) cujo funcionamento foi detalhado no item 3.3.2.

Essa tecnologia de disponibilização de conteúdo dinâmico para WWW é baseada na linguagem Visual Basic e também possui a capacidade de lidar com componentes ActiveX. Dessa forma, os objetos utilizados no lado do cliente também estão disponíveis para o servidor de aplicação via ASP, o que proporciona um tratamento padronizado e transparente dos objetos utilizados no *framework*.

A principal funcionalidade do servidor de aplicação é tratar as requisições HTTP e ligar os dados XML gerados no cliente com o banco de dados relacional ao servidor de banco de dados e vice-versa.

4.3.4 Acesso a Dados

Para descrever o *software* nas diversas camadas do *framework* é necessário definir algumas tecnologias Microsoft no sentido de diferenciá-las e evitar confusões com as siglas.

> "A abordagem evolucionária da Microsoft no desenvolvimento de *software* freqüentemente leva a confusões com os nomes dos produtos. Isso é especialmente verdadeiro para a família COM, DCOM, OLE e ActiveX. O *Component Object Model* (COM) – Modelo de Objetos Componentes é a estrutura de componentes da Microsoft. Trata-se de um padrão binário para a construção e execução de aplicações baseadas em componentes[26]. Os objetos COM encapsulam funcionalidades e fornecem uma interface padrão para a comunicação entre objetos. Por ser um padrão binário, os objetos COM são independentes de linguagem e podem ser usados e estendidos sem a necessidade de alterar o código fonte original. O COM é inerentemente

[26] Para Harris (apud Orfali et al., 1999) um componente é "uma peça de *software* pequena o suficiente para ser criada e mantida, grande o suficiente para ser distribuída e com interfaces padrão para garantir a interoperabilidade". De forma simplificada, componentes são peças reutilizáveis de software.

> distribuído e o *Distributed COM* (DCOM) – COM distribuído é apenas uma utilização distribuída do COM. O *Object Linking and Embedding* (OLE) – Ligação e Incorporação de Objetos é uma extensão construída sobre o COM para tratar documentos complexos. O ActiveX é outra extensão do COM que fornece uma grande gama de funcionalidades. Os controles ActiveX incluem suporte de baixo nível a elementos básicos de interface do Windows tais como menus, botões, arrastar-e-soltar, etc. Por outro lado, o ActiveX também é a resposta da Microsoft para o Java em termos de programas que podem ser baixados na Internet via WWW. “
>
> (HERRING et al., 1998, p.2)

Apesar das soluções de *software* da Microsoft evoluírem numa espécie de universo paralelo, as tecnologias Windows, COM e ActiveX, Internet Information Server, Microsoft Transaction Server, OleDB etc. são integradas e proporcionam uma estrutura de desenvolvimento operacional (Orfali et al., 1999). Para Dolgicer (apud Orfali et al., 1999, p. 250) “tudo o que a Microsoft faz é relacionado com o Windows e o COM. Ninguém, nem mesmo os maiores adversários da Microsoft, podem acusá-la de ser inarticulada com relação a sua estratégia.”

Outra tecnologia relevante para esse trabalho é o *Open Database Conectivity* (ODBC) – Conectividade Aberta para Banco de Dados. Trata-se de um padrão Microsoft de acesso a dados inicialmente disponível apenas para Windows mas que teve uma popularização muito grande após o lançamento de sua segunda versão em 1994. O padrão ganhou versões para outras plataformas e a maioria dos fabricantes atualmente fornecem suporte a API ODBC juntamente com os *drivers* ODBC para os servidores em questão (Orfali et al.,1999).

Com o objetivo de estender o sucesso da tecnologia ODBC, criou-se o OLE DB que é uma interface de baixo nível para dados distribuídos numa organização. Trata-se de uma especificação aberta que disponibiliza um padrão aberto para o acesso a todos os tipos de dados (Microsoft, 2001).

Orfali et al. (1999) explicam que o OLE DB é um conjunto de objetos COM e interfaces para o acesso a dados. Em conjunto com o ADO[27] formam a estratégia da Microsoft para dados, o chamado *Universal Data Access* (UDA) – Acesso Universal de dados.

O acesso ao banco de dados do *framework* é feito através do OLE DB. Uma vez instalado um *driver* (no servidor de aplicação) OLE DB para acesso ao banco de dados no servidor de banco de dados, basta solicitar o serviço ao driver OLE DB e o acesso é feito de modo transparente.

A quantidade de dados transmitidos entre o servidor de banco de dados e o servidor de aplicação não é muito grande devido a otimização das consultas. Apesar disso, para aumentar a eficiência do sistema como um todo, recomenda-se que o servidor de aplicação e o servidor de banco de dados pertençam a uma mesma sub-rede ou tenham uma linha de comunicação rápida.

4.4 Banco de dados

O banco de dados escolhido para implementar o *framework* foi o Borland Interbase 6.0. Trata-se de um Sistema Gerenciador de Banco de Dados relacional *open-source*[28] compatível com grande parte do padrão ANSI SQL92. O Interbase suporta funcionalidades como *stored procedures*, integridade referencial, *triggers*, *generators* e transações. Essas características permitem reduzir o tempo de desenvolvimento e aumentar o grau de confiabilidade da aplicação (Borland, 2001).

---

[27] Activex Data Objects (ADO) - Objetos de Dados ActiveX – é uma API de alto nível para acesso a dados construída sobre o OLE DB. O ADO está disponível para algumas linguagens de programação tais como Visual Basic, PowerBuilder, Delphi, Java, Javascript e ASP (Orfali et al., 1999).

[28] De uma maneira geral, o termo *open source* (código aberto ou livre) refere-se a qualquer programa cujo código fonte é disponibilizado para ser utilizado ou modificado por outros desenvolvedores. Historicamente, os fabricantes de software geralmente não disponibilizavam o código fonte e essa ainda é a prática mais comum. O software de código aberto é normalmente desenvolvido como uma colaboração pública e é disponibilizado livremente (Searchenterpriselinux.com, 2003).

Uma característica importante a respeito do Interbase é que o *software* servidor de banco de dados está disponível para várias plataformas tais como Linux, Solaris e Windows. Portanto, é possível executar o *software* servidor de banco de dados em servidores (hardware) com processamento RISC ou computadores PC rápidos com sistemas operacionais Linux ou Windows. Conseqüentemente é possível aproveitar melhor os recursos disponíveis numa rede de computadores, beneficiando-se da arquitetura de três camadas e aumentando a eficiência do sistema como um todo.

O banco de dados relacional do *framework* foi projetado com o objetivo de servir a qualquer ambiente de ensino-aprendizagem que tenha a necessidade de garantir a persistência das interações do usuário com o ambiente. Portanto, o banco de dados deve ser capaz de manter os dados relacionados no item 4.1.

Além de manter os dados, a estrutura relacional do banco de dados deve otimizar o tempo de resposta para as consultas mais comuns e viabilizar a extração de informações que podem ser usadas para tomada de decisão.

Para compreender a maneira que os dados são organizados nesse banco de dados relacional, é importante visualizar a representação gráfica do modelo entidade-relacionamento.

4.4.1 Representação Gráfica do Modelo Entidade-Relacionamento

Para facilitar a compreensão desse modelo gráfico (Figura 16) é conveniente observar subconjuntos de relacionamentos entre as tabelas. Tomando-se por exemplo as tabelas MODULOS, PAGINAS e PERGUNTAS. Há uma clara relação de pertinência entre essas entidades. Cada pergunta pertence a uma determinada página de conteúdo que por sua vez pertence a um determinado módulo. Um módulo pode possuir várias páginas e cada página pode possuir várias perguntas.

Considerando o modelo E-R específico aqui proposto e as características inerentes do modelo relacional é possível determinar, por exemplo, a qual módulo uma determinada pergunta pertence. A possibilidade de chegar a esse tipo de informação indireta é uma das grandes vantagens do modelo relacional.

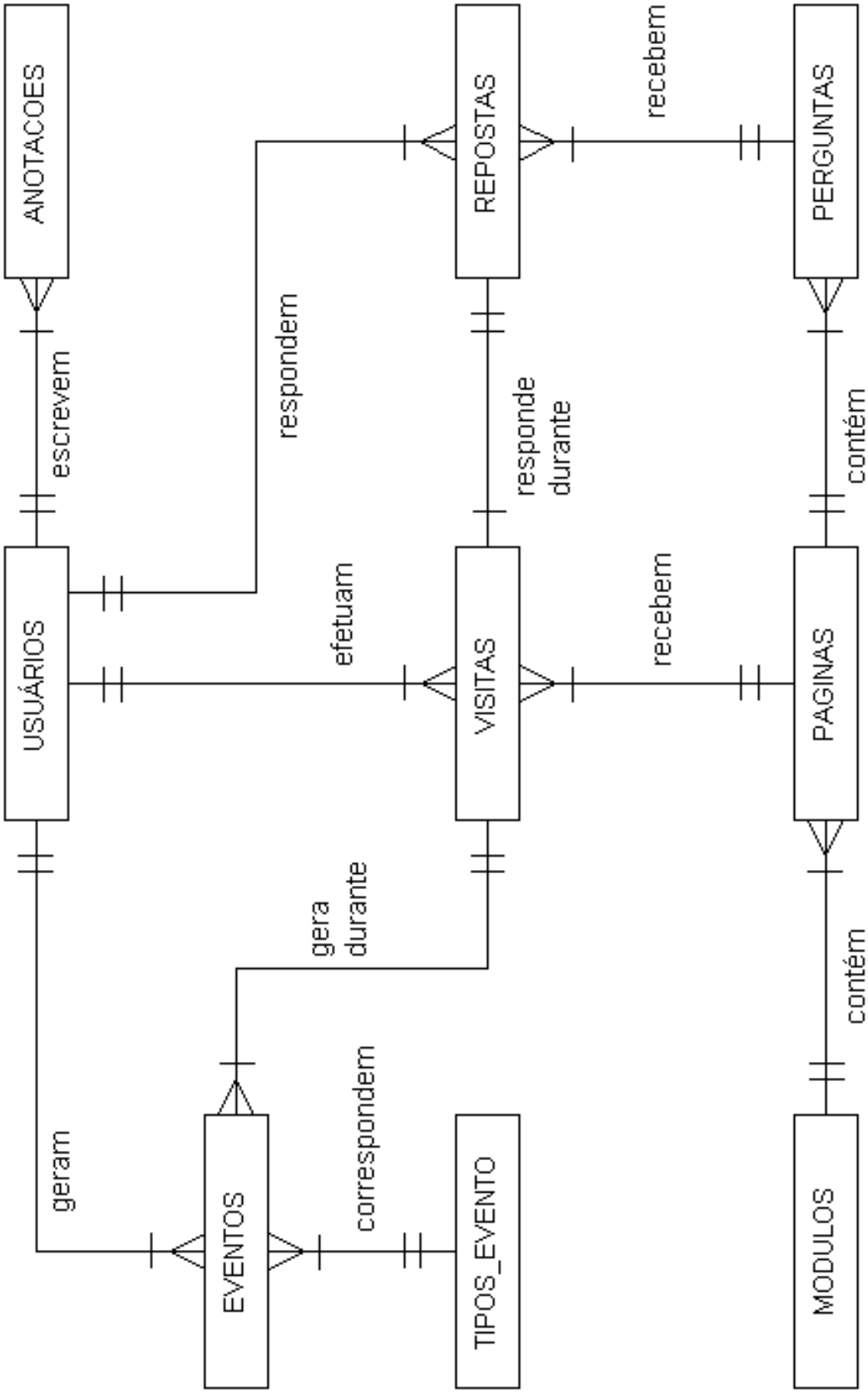


Figura 16: Modelo Entidade-Relacionamento de acordo com a notação *Crow´s foot*

Os dados pessoais de usuário ficam armazenados na tabela USUARIOS. Já os dados a respeito das interações dos usuários com as PAGINAS e PERGUNTAS são armazenados nas tabelas VISITAS e RESPOSTAS respectivamente. Uma vez que o usuário pode visitar uma mesma página várias vezes e portanto responder uma pergunta mais de uma vez, a tabela RESPOSTAS tem um relacionamento direto com a tabela VISITAS visto que uma resposta é dada sempre em uma determinada visita.

A tabela EVENTOS mantém informações a respeito dos eventos gerados pelos usuários tais como entrada e saída do sistema ou ainda as interações com as *hyperlinks*. Os eventos estão associados a tabela VISITAS pois é durante uma visita que os eventos ocorrem. Também criou-se uma tabela TIPOS_EVENTOS para evitar a redundância de dados – a descrição do evento fica nesta última tabela. Outro motivo para a criação da tabela de tipos de eventos é viabilizar a utilização do *framework* para múltiplos ambientes que podem ter interesse em monitorar diferentes eventos.

As anotações textuais dos usuários ficam na tabela ANOTACOES que é separada da tabela USUARIOS para otimizar a configuração física do banco de dados.

4.4.2 Dicionário de Dados

A descrição detalhada dos campos de dados em cada tabela constitui o dicionário de dados e serve a diversos fins. O principal deles é garantir que todos as pessoas que trabalham com o banco de dados saibam exatamente que informação cada campo e tabela armazena. O *script* SQL de geração do banco de dados pode ser visto, na íntegra, no item 7.1.

Primeiramente é necessário descrever os domínios que funcionam como tipos de variáveis em linguagens de programação (Tabela V). Para facilitar a compreensão das tabelas serão utilizadas as abreviações CP e CE referentes a chave primária e chave estrangeira respectivamente.

Domínio	CONTEÚDO	ESTRUTURA	INTEGRIDADE
DESCRICAO	A descrição de uma entidade	Caracter(30)	
EMAIL	Um endereço eletrônico	Caracter(50)	
FONE	Um número de telefone	Caracter(17)	
NOME	Um nome de pessoa	Caracter(40)	Não pode ser nulo
PAGINA_REF	Uma referência a uma página do ambiente de ensino-aprendizagem	Caracter(60)	
PERGUNTA	Uma referência a uma pergunta do ambiente de ensino-aprendizagem	Caracter(20)	
USUARIO_ID	O código de um usuário	Inteiro Curto	Não pode ser nulo
USUARIO_LOGIN	O apelido ou nome pelo qual o sistema identifica um usuário	Caracter(12)	Não pode ser nulo
VISITA_ID	O código de uma visita efetuada a uma determinada página do sistema	inteiro	Não pode ser nulo

Tabela V: Domínios do banco de dados

Tabela MODULOS

Esta tabela armazena dados referentes a diferentes módulos (ou grupos) de conteúdo dentro de uma aplicação de ensino-aprendizagem (detalhamento na Tabela VI). O comando SQL para a criação da tabela MODULOS é o seguinte:

```
CREATE TABLE MODULOS
(
          MODULO_ID     INTEGER      NOT NULL,
        MODULO_NOME     VARCHAR(30)  CHARACTER SET ISO8859_1,
   MODULO_DESCRICAO     DESCRICAO,
   MODULO_PEDAGOGIA     DESCRICAO
);
COMMIT;

ALTER TABLE MODULOS ADD CONSTRAINT PK_MODULOS_ID PRIMARY KEY (MODULO_ID);
COMMIT;
```

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
MODULO_ ID	O código de identificação única do módulo	inteiro	Não pode ser nulo (CP)
MODULO_ DESCRICAO	A descrição textual do módulo.	DESCRICAO	
MODULO_ PEDAGOGIA	A abordagem pedagógica utilizada no módulo.	DESCRICAO	

Tabela VI: Tabela relacional MODULOS

Tabela PAGINAS

Esta tabela armazena dados referentes às páginas de conteúdo de uma aplicação de ensino-aprendizagem (detalhamento na Tabela VII). O comando SQL para a criação da tabela PAGINAS é o seguinte:

```
CREATE TABLE PAGINAS
(
        PAGINA_ID     INTEGER         NOT NULL,
      PAGINA_NOME     VARCHAR(20)     CHARACTER SET ISO8859_1,
   PAGINA_ASSUNTO     VARCHAR(60)     CHARACTER SET ISO8859_1,
        MODULO_ID     INTEGER         NOT NULL
);
COMMIT;

ALTER TABLE PAGINAS ADD CONSTRAINT PK_PAGINAS_ID PRIMARY KEY (PAGINA_ID);
COMMIT;
```

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
PAGINA_ID	O código de identificação única da página	inteiro	Não pode ser nulo (CP)
PAGINA_ NOME	O título ou nome da página.	Caracter(20)	
PAGINA_ ASSUNTO	A definição do assunto da página.	Caracter(60)	
MODULO_I D	A referência ao código do módulo ao qual pertence a página: referencia a tabela MODULOS	inteiro	Não pode ser nulo (CE)

Tabela VII: Tabela relacional PAGINAS

Tabela PERGUNTAS

Armazena dados referentes às perguntas apresentadas em páginas de conteúdo de uma aplicação de ensino-aprendizagem (detalhamento na Tabela VIII). O comando SQL para a criação da tabela PERGUNTAS é o seguinte:

```
CREATE TABLE PERGUNTAS
(
         PERGUNTA_ID    INTEGER          NOT NULL,
       PERGUNTA_NOME    PERGUNTA,
       PERGUNTA_TIPO    INTEGER,
    PERGUNTA_ASSUNTO    VARCHAR(60)      CHARACTER SET ISO8859_1,
   PERGUNTA_RESPOSTA    VARCHAR(250)     CHARACTER SET ISO8859_1,
          PAGINA_ID    INTEGER          NOT NULL
);
COMMIT;

ALTER TABLE PERGUNTAS ADD CONSTRAINT PK_PERGUNTAS_ID PRIMARY KEY
(PERGUNTA_ID);

ALTER TABLE PERGUNTAS ADD CONSTRAINT FK_PERGUNTAS__PAGINA_ID
  FOREIGN KEY (PAGINA_ID) REFERENCES PAGINAS (PAGINA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;
```

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
PERGUNTA_ID	O código de identificação única da pergunta	inteiro	Não pode ser nulo (CP)
PERGUNTA_NOME	O título ou nome da pergunta.	PERGUNTA	
PERGUNTA_TIPO	O tipo de pergunta (escolha simples, múltipla escolha, discursiva)	Inteiro	Entre “0” e “3” [29]
PERGUNTA_ASSUNTO	A definição do assunto da pergunta.	inteiro	
PERGUNTA_RESPOSTA	A resposta correta para a pergunta.	Caracter(250)	
PAGINA_ID	A referência ao código da página a qual pertence a pergunta: referencia a tabela PAGINAS	inteiro	Não pode ser nulo (CE)

Tabela VIII: Tabela relacional PERGUNTAS

[29] Decidiu-se adotar um código fixo para determinar o tipo de pergunta uma vez que não há, a priori, grande variação nos tipos de perguntas existentes portanto não justifica a criação de uma tabela e mais uma chave estrangeira para tal.

Tabela VISITAS

Esta tabela armazena dados referentes às visitas que os usuários efetuam às páginas de conteúdo de uma aplicação de ensino-aprendizagem (detalhamento na Tabela IX). O comando SQL para a criação da tabela VISITAS é o seguinte:

```
CREATE TABLE VISITAS
(
           VISITA_ID              VISITA_ID,
    VISITA_DATA_HORA              TIMESTAMP       NOT NULL,
    VISITA_TEMPO_STR              VARCHAR(12)     CHARACTER SET ISO8859_1,
          USUARIO_ID              USUARIO_ID,
           PAGINA_ID              INTEGER         NOT NULL
);
COMMIT;

ALTER TABLE VISITAS ADD CONSTRAINT PK_VISITAS_ID PRIMARY KEY (VISITA_ID);
COMMIT;

ALTER TABLE VISITAS ADD CONSTRAINT FK_VISITAS__PAGINA_ID
  FOREIGN KEY (PAGINA_ID) REFERENCES PAGINAS (PAGINA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;

ALTER TABLE VISITAS ADD CONSTRAINT FK_VISITAS__USUARIO_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;

CREATE ASC INDEX IDX_USUARIO_ID__VISITAS
ON VISITAS (USUARIO_ID);
COMMIT;
```

Tabela RESPOSTAS

Mantém dados referentes às respostas dadas pelos usuários às perguntas presentes em páginas de conteúdo de uma aplicação de ensino-aprendizagem (detalhamento na Tabela X). O comando SQL para a criação da tabela RESPOSTAS é o seguinte:

```
CREATE TABLE RESPOSTAS
(
          RESPOSTA_ID             INTEGER         NOT NULL,
         RESPOSTA_STR             VARCHAR(250)    CHARACTER SET ISO8859_1,
    RESPOSTA_CLICADAS             VARCHAR(250)    CHARACTER SET ISO8859_1,
   RESPOSTA_DATA_HORA             TIMESTAMP       NOT NULL,
          PERGUNTA_ID             INTEGER         NOT NULL,
           USUARIO_ID             USUARIO_ID,
```

```
            VISITA_ID              VISITA_ID
);
COMMIT;

ALTER TABLE RESPOSTAS ADD CONSTRAINT PK_RESPOSTAS_ID PRIMARY KEY
(RESPOSTA_ID);
COMMIT;

ALTER TABLE RESPOSTAS ADD CONSTRAINT FK_RESPOSTAS__PERGUNTA_ID
  FOREIGN KEY (PERGUNTA_ID) REFERENCES PERGUNTAS (PERGUNTA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;

ALTER TABLE RESPOSTAS ADD CONSTRAINT FK_RESPOSTA__USER_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;

ALTER TABLE RESPOSTAS ADD CONSTRAINT FK_RESPOSTA__VISITA_ID
  FOREIGN KEY (VISITA_ID) REFERENCES VISITAS (VISITA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;

CREATE ASC INDEX IDX_USUARIO_E_PERG__RESPOSTAS
ON RESPOSTAS (PERGUNTA_ID, USUARIO_ID);
COMMIT;
```

O índice IDX_USUARIO_E_PERG__RESPOSTAS, composto pelos campos PERGUNTA_ID e USUARIO_ID foi criado para otimizar as consultas para formação de visões de resposta.

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
VISITA_ID	O código de identificação única da visita	inteiro	Não pode ser nulo (CP)
VISITA_ DATA_HORA	A data e hora exatas em que o usuário entra na página que está visitando.	timestamp	Não pode ser nulo
VISITA_ TEMPO_STR	A duração (intervalo de tempo) da visita	Caracter(12)	
USUARIO_ID	A referência ao usuário que efetuou a visita: referencia a tabela USUARIOS	USUARIO_ID	Não pode ser nulo (CE)
PAGINA_ID	A referência ao código da página a qual pertence a pergunta: referencia a tabela PAGINAS	inteiro	Não pode ser nulo (CE)

Tabela IX: Tabela relacional VISITAS

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
RESPOSTA_ ID	O código de identificação única da resposta	inteiro	Não pode ser nulo (CP)
RESPOSTA_ STR	A resposta dada pelo usuário	Caracter(250)	
RESPOSTA_ CLICADAS	Um vetor representando as clicadas que o usuário efetuou sobre os itens de resposta[30]	Caracter(250)	
RESPOSTA_ DATA_HORA	A data e hora exatas em que o usuário entra na página que está visitando.	timestamp	Não pode ser nulo
PERGUNTA_ ID	A referência a pergunta que foi respondida: referencia a tabela PERGUNTAS	inteiro	Não pode ser nulo (CE)
USUARIO_ID	A referência ao usuário que efetuou a visita: referencia a tabela USUARIOS	USUARIO_ID	Não pode ser nulo (CE)
VISITA_ID	A referência ao código da visita na qual a resposta foi efetuada: referencia a tabela VISITAS	inteiro	Não pode ser nulo (CE)

Tabela X: Tabela relacional RESPOSTAS

Tabela EVENTOS

Armazena dados referentes a eventos gerados pelo usuário em uma visita (detalhamento na Tabela XI). O comando SQL para a criação da tabela EVENTOS é o seguinte:

```
CREATE TABLE EVENTOS
(
          EVENTO_ID              INTEGER       NOT NULL,
   EVENTO_DATA_HORA              TIMESTAMP     NOT NULL,
        EVENTO_ALVO              VARCHAR(100)  CHARACTER SET ISO8859_1,
     TIPO_EVENTO_ID              INTEGER       NOT NULL,
         USUARIO_ID              USUARIO_ID,
          VISITA_ID              VISITA_ID
);
```

[30] Uma pergunta geralmente tem mais de uma alternativa a ser escolhida pelo usuário. Por motivos de otimização física do banco de dados decidiu-se optar pela denormalização e manter um vetor nesse campo. As clicadas ficam armazenadas separadas por vírgula tal como em “0,1,0” que quer dizer: o usuário clicou somente uma vez na Segunda alternativa.

```
COMMIT;

ALTER TABLE EVENTOS ADD CONSTRAINT PK_EVENTOS_ID PRIMARY KEY (EVENTO_ID);
COMMIT;

ALTER TABLE EVENTOS ADD CONSTRAINT FK_EVENTOS__PAGINA_ID
  FOREIGN KEY (VISITA_ID) REFERENCES VISITAS (VISITA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE EVENTOS ADD CONSTRAINT FK_EVENTOS__USUARIO_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;

ALTER TABLE EVENTOS ADD CONSTRAINT FK_EVENTOS__EVENTO_TIPO_ID
  FOREIGN KEY (TIPO_EVENTO_ID) REFERENCES TIPOS_EVENTO (TIPO_EVENTO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;
```

Tabela TIPOS_EVENTOS

Esta tabela armazena dados referentes aos diferentes tipos de eventos gerados pelo usuário enquanto interage com o sistema (detalhamento na Tabela XII). O comando SQL para a criação da tabela TIPOS_EVENTOS é o seguinte:

```
CREATE TABLE TIPOS_EVENTO
(
       TIPO_EVENTO_ID              INTEGER       NOT NULL,
  TIPO_EVENTO_APELIDO              VARCHAR(30)   CHARACTER SET ISO8859_1,
TIPO_EVENTO_DESCRICAO              VARCHAR(255)  CHARACTER SET ISO8859_1
);

ALTER TABLE TIPOS_EVENTO ADD CONSTRAINT PK_TIPO_EVENTO_ID PRIMARY KEY
(TIPO_EVENTO_ID);
COMMIT;

CREATE ASC INDEX IDX_TIPO_EVENTO_APELIDO__EVENTOS
ON TIPOS_EVENTO (TIPO_EVENTO_APELIDO);
COMMIT;
```

Tabela USUARIOS

Mantém dados referentes aos usuários do sistema (detalhamento na Tabela XIII). O comando SQL para a criação da tabela USUARIOS é o seguinte:

```
CREATE TABLE USUARIOS
(
              USUARIO_ID             USUARIO_ID,
           USUARIO_LOGIN             USUARIO_LOGIN,
           USUARIO_SENHA             USUARIO_LOGIN,
            USUARIO_NOME             NOME,
          USUARIO_DTNASC             TIMESTAMP,
       USUARIO_DTCRIACAO             TIMESTAMP,
   USUARIO_ULTIMA_PAGINA             PAGINA_REF,
    USUARIO_ULTIMO_LOGIN             TIMESTAMP,
        USUARIO_ENDERECO             VARCHAR(50)  CHARACTER SET ISO8859_1,
          USUARIO_BAIRRO             VARCHAR(30)  CHARACTER SET ISO8859_1,
          USUARIO_CIDADE             VARCHAR(30)  CHARACTER SET ISO8859_1,
          USUARIO_ESTADO             VARCHAR(30)  CHARACTER SET ISO8859_1,
             USUARIO_CEP             VARCHAR(9)   CHARACTER SET ISO8859_1,
            USUARIO_FONE             FONE,
           USUARIO_EMAIL             EMAIL
);
COMMIT;

ALTER TABLE USUARIOS ADD CONSTRAINT PK_USUARIOS_ID PRIMARY KEY
(USUARIO_ID);
COMMIT;

CREATE ASC INDEX IDX_USUARIO_LOGIN__USUARIOS
ON USUARIOS (USUARIO_LOGIN);
COMMIT;

CREATE ASC INDEX IDX_USUARIO_NOME__USUARIOS
ON USUARIOS (USUARIO_NOME);
COMMIT;
```

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
EVENTO_ID	O código de identificação única do evento	inteiro	Não pode ser nulo (CP)
EVENTO_ DATA_HORA	A data e hora exatas em que o usuário gera o evento	timestamp	Não pode ser nulo
EVENTO_ ALVO	O objeto sobre o qual o evento é gerado tal como um *hyperlink* que foi clicado.	Caracter(100)	
TIPO_ EVENTO_ID	A referência ao tipo de evento que foi gerado: referencia a tabela TIPOS_EVENTO	inteiro	Não pode ser nulo (CE)
USUARIO_ID	A referência ao usuário que efetuou a visita: referencia a tabela USUARIOS	USUARIO_ID	Não pode ser nulo (CE)
VISITA_ID	A referência ao código da visita na qual o evento foi gerado: referencia a tabela VISITAS	inteiro	Não pode ser nulo (CE)

Tabela XI: Tabela relacional EVENTOS

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
TIPO_EVENTO_ ID	O código de identificação única do tipo de evento	inteiro	Não pode ser nulo (CP)
TIPO_EVENTO_ APELIDO	Uma descrição breve do tipo de evento.	Caracter(30)	
TIPO_EVENTO_ DESCRICAO	Uma descrição detalhada do tipo de evento.	Caracter(255)	

Tabela XII: Tabela relacional TIPOS_EVENTO

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
USUARIO_ID	O código de identificação única do usuário	inteiro	Não pode ser nulo (CP)
USUARIO_LOGIN	O apelido pelo qual o sistema conhece o usuário.	USUARIO_ LOGIN	Não pode ser nulo
USUARIO_ SENHA	A senha que permite ao usuário acessar o sistema.	USUARIO_ LOGIN	Não pode ser nulo
USUARIO_NOME	O nome completo do usuário	NOME	Não pode ser nulo
USUARIO_ DTNASC	A data de nascimento do usuário.	Timestamp	
USUARIO_ DTCRIACAO	A data de criação da conta do usuário.	Timestamp	
USUARIO_ ULTIMA_PAGINA	A última página visitada pelo usuário.	PAGINA_REF	
USUARIO_ ULTIMO_LOGIN	A última vez que o usuário acessou o sistema.	timestamp	
USUARIO_ ENDERECO	O endereço do usuário.	Caracter(50)	
USUARIO_ BAIRRO	O bairro do endereço do usuário.	Caracter(30)	
USUARIO_ CIDADE	A cidade do endereço do usuário	Caracter(30)	
USUARIO_ ESTADO	O estado do endereço do usuário	Caracter(30)	
USUARIO_CEP	O CEP do endereço do usuário.	Caracter(9)	
USUARIO_FONE		FONE	
USUARIO_EMAIL		EMAIL	

Tabela XIII: Tabela relacional USUARIOS

Os campos USUARIO_LOGIN e USUARIO_NOME possuem índices ascendentes (IDX_USUARIO_LOGIN__USUARIOS e IDX_USUARIO_NOME__USUARIOS

respectivamente) com o objetivo de otimizar a busca de um usuário a partir de seu *login* ou nome, que é uma tarefa consulta freqüente.

Tabela ANOTAÇÕES

Esta tabela armazena as anotações textuais dos usuários do sistema (detalhamento na Tabela XIV). O comando SQL para a criação da tabela ANOTACOES é o seguinte:

```
CREATE TABLE ANOTACOES
(
        ANOTACAO_ID              INTEGER          NOT NULL,
     ANOTACAO_TEXTO              VARCHAR(32000)   CHARACTER SET ISO8859_1,
 ANOTACAO_DATA_HORA              TIMESTAMP,
    ANOTACAO_TITULO              VARCHAR(64)      CHARACTER SET ISO8859_1,
         USUARIO_ID              USUARIO_ID
);
COMMIT;

ALTER TABLE ANOTACOES ADD CONSTRAINT PK_ANOTACOES_ID PRIMARY KEY
(ANOTACAO_ID);
COMMIT;

ALTER TABLE ANOTACOES ADD CONSTRAINT FK_ANOTACOES__USUARIO_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;
```

CAMPO	CONTEÚDO	ESTRUTURA	INTEGRIDADE
ANOTACAO_ID	O código de identificação única do usuário	inteiro	Não pode ser nulo (CP)
ANOTACAO_ TEXTO	O texto da anotação em si.	Caracter (32000)	
ANOTACAO_ DATA_HORA	A data e hora exatas em que a anotação foi feita.	Timestamp	
ANOTACAO_ TITULO	O título dado a anotação pelo usuário.	Caracter(64)	
USUARIO_ID	A referência ao usuário que efetuou a visita: referencia a tabela USUARIOS	USUARIO_ID	Não pode ser nulo (CE)

Tabela XIV: Tabela relacional ANOTACOES

O esquema do banco de dados aqui proposto está na terceira forma normal, desconsiderando os casos citados no dicionário de dados, que foram resultado do processo de otimização física. Apesar dessas pequenas alterações pode-se afirmar que o banco de dados não sofre de anomalias de repetição, atualização, inserção e exclusão abordadas no item 2.6.

Uma representação gráfica com detalhamento dos campos pode ser vista na Figura 17. Essa representação foi gerada com o *software Computer Associates ERwin 4.0* através de engenharia reversa, ou seja, a representação foi gerada a partir do *script* SQL de criação do banco de dados.

4.4.3 Visões

As visões utilizadas nesse banco de dados foram criadas no sentido de facilitar o acesso diferenciado aos dados. Uma primeira visão, bastante simples mas de grande relevância é a visão ULTIMAS_RESPOSTAS (código SQL a seguir).

```
SELECT PERGUNTA_ID, USUARIO_ID, MAX(RESPOSTA_DATA_HORA) AS
RESPOSTA_DATA_HORA FROM RESPOSTAS
GROUP BY PERGUNTA_ID, USUARIO_ID;
```

Trata-se de um agrupamento específico da tabela RESPOSTAS. Toma-se os dados da tabela agrupados por usuário e por pergunta e retorna-se apenas as tuplas com a maior data. O resultado é a tabela RESPOSTAS apenas com as últimas respostas de cada usuário para cada pergunta.

Já a visão ESTRUTURA_PERGUNTAS é a junção das tabelas PAGINAS, PERGUNTAS e RESPOSTAS. Ela proporciona o acesso a um subconjunto maior de dados do que apenas as tabelas separadas. O código SQL para a visão ESTRUTURA_PERGUNTAS_RESPOSTAS é o seguinte:

```
SELECT PAG.PAGINA_NOME, PERG.PAGINA_ID, PERG.PERGUNTA_NOME, PERGUNTA_TIPO,
R.* FROM
(
```

```
  PAGINAS PAG LEFT JOIN PERGUNTAS PERG ON PAG.PAGINA_ID = PERG.PAGINA_ID
)  RIGHT JOIN RESPOSTAS R ON (R.PERGUNTA_ID = PERG.PERGUNTA_ID);
```

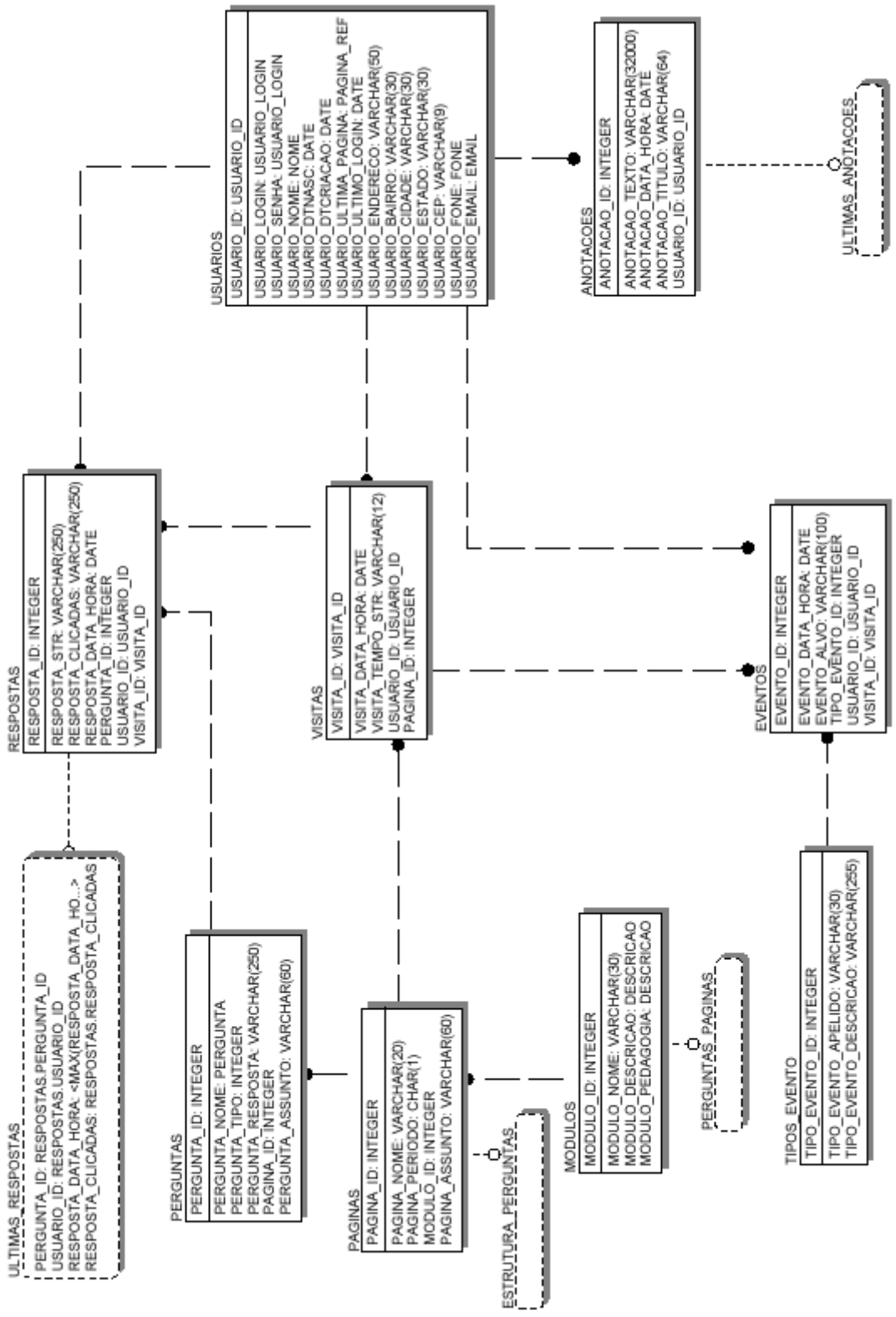


Figura 17: Modelo Entidade-Relacionamento gerado através de engenharia reversa com a ferramenta *ERwin 4.0*

É importante ressaltar que há duas junções nessa consulta uma vez que ela une campos de três tabelas diferentes. A primeira delas é a junção entre as tabelas PAGINAS e PERGUNTAS. O termo LEFT JOIN refere-se a uma junção externa que preserva as tuplas da tabela da esquerda da junção, no caso a tabela PAGINAS. A segunda junção é entre a visão que resulta da primeira junção e a tabela RESPOSTAS. O termo RIGHT JOIN refere-se a uma junção externa que preserva todas as tuplas da tabela da direita da junção, no caso a tabela RESPOSTAS.

As tuplas que resultam dessa visão são todas as respostas de todos os usuários relacionadas às suas respectivas páginas e perguntas. Com esta visão é possível determinar diretamente a qual página pertence uma determinada pergunta, quais usuários responderam determinada pergunta, qual a resposta de cada usuário etc.

4.4.4 Otimização de Consultas

Como pode ser visto através do modelo E-R (item 4.4.1), o banco de dados do *framework* tem diversos relacionamentos. Conseqüentemente, mesmo consultas simples utilizam uma ou mais junções. Consultas mais complexas, que executam muitas junções, exigem muito do processamento e portanto precisam ser otimizadas para executar rapidamente.

Por exemplo, cada vez que o usuário entra no sistema, é necessário efetuar uma consulta que serve para a recuperação de suas últimas respostas, garantindo assim a continuidade do estado gerado nas sessões anteriores. Se o usuário já respondeu uma mesma pergunta mais de uma vez, é necessário que se leve em consideração somente a última resposta dada para a questão.

Para tal, executa-se uma consulta complexa que acessa as visões ESTRUTURA_PERGUNTAS e ULTIMAS_RESPOSTAS. Uma vez que essas visões acessam as tabelas PAGINAS, PERGUNTAS e RESPOSTAS indiretamente e repetidas vezes, há uma grande quantidade de junções. Uma forma de efetuar essa consulta em SQL é a seguinte:

```
SELECT EP.PAGINA_NOME, EP.RESPOSTA_STR
 FROM ESTRUTURA_PERGUNTAS EP RIGHT JOIN ULTIMAS_RESPOSTAS UR
  ON (EP.PERGUNTA_ID = UR.PERGUNTA_ID)
      AND (EP.RESPOSTA_DATA_HORA = UR.RESPOSTA_DATA_HORA)
 WHERE (USUARIO_ID = <VALOR_LITERAL>);
```

Durante a execução, o valor literal é substituído por um valor real, por exemplo, para o usuário com o código de identificação “1”, o servidor executa essa consulta, substituindo <VALOR_LITERAL> pelo número “1”. Num teste simples executado num computador AMD Athlon 700Mhz essa consulta resultou nos seguintes parâmetros:

- Tempo de execução: 5min e 8s
- Tempo de preparação: 60 milésimos de segundo
- Leituras efetuadas: 208
- Escritas efetuadas: 182
- Registros resultado (*fetched*): 9

O plano de execução gerado pelo SGBD para essa consulta foi:

```
PLAN JOIN (UR RESPOSTAS ORDER IDX_USUARIO_E_PERG__RESPOSTAS,
           JOIN (EP R NATURAL,
                 JOIN (EP PAG NATURAL,
                       EP PERG INDEX (FK_PERGUNTAS__PAGINA_ID))
                 ))
```

Apesar de utilizar a indexação composta dos campos PERGUNTA_ID e USUARIO_ID (índice ascendente IDX_USUARIO_E_PERG__RESPOSTAS da tabela RESPOSTAS), a consulta tem uma execução lenta porque leva em consideração os registros de todos os usuários.

Um agravante é que quanto maior o número de usuários, maior o total de registros de visitas e respostas e mais tempo será necessário para a execução dessa consulta. Uma outra consulta SQL que gera os mesmos resultados é a seguinte:

```
SELECT EP.PAGINA_NOME, EP.PERGUNTA_NOME, EP.PERGUNTA_TIPO,
EP.RESPOSTA_DATA_HORA, EP.RESPOSTA_STR, UR.*
  FROM ESTRUTURA_PERGUNTAS EP, ULTIMAS_RESPOSTAS UR
  WHERE ((EP.USUARIO_ID = <VALOR_LITERAL>)
          AND (UR.USUARIO_ID = <VALOR_LITERAL>)
          AND (EP.PERGUNTA_ID = UR.PERGUNTA_ID)
          AND (EP.RESPOSTA_DATA_HORA = UR.RESPOSTA_DATA_HORA));
```

O teste dessa consulta efetuado nas mesmas condições do anterior resultou nos seguintes parâmetros:

- Tempo de execução: 380 milésimos de segundo
- Tempo de preparação: 60 milésimos de segundo
- Leituras efetuadas: 163
- Escritas efetuadas: 7
- Registros resultado (*fetched*): 9

O plano de execução gerado pelo SGBD para essa consulta foi:

```
PLAN MERGE (
SORT  (UR RESPOSTAS ORDER IDX_USUARIO_E_PERG__RESPOSTAS),
SORT  (JOIN (EP R INDEX (FK_RESPOSTA__USER_ID)
             JOIN (EP PAG NATURAL,
                   EP PERG INDEX (FK_PERGUNTAS__PAGINA_ID))
             )))
```

Esta consulta executa mais rapidamente porque a seleção das tuplas relativas ao usuário específico é feita antes e, portanto somente os registros daquele usuário são levados em consideração. Isso ocorre porque o plano de execução inclui a chave estrangeira do campo USUARIO_ID na tabela RESPOSTAS. Outro fator de suma importância para essa otimização foi a ordem de execução dos predicados que gerou um produto cartesiano muito menor que na consulta anterior. Numa consulta com diversas junções como esta, o ganho de eficiência é considerável. Nesse caso, por exemplo, o tempo de execução baixou de 5 minutos e 8 segundos para apenas 380 milésimos de segundo, ou seja, aproximadamente 800 vezes mais rápido.

As duas consultas retornam as mesmas tuplas, mas a segunda é muito mais eficiente. Esta segunda consulta SQL é mais rápida pois a velocidade de execução depende somente do número de tuplas (visitas e respostas) do usuário em questão e não é afetada pelo total de tuplas nessas tabelas. Tal otimização é bastante importante visto que é inaceitável, do ponto de vista da qualidade do sistema, uma degradação significativa da velocidade de resposta ao passo que o volume de dados vai aumentando.

Uma vez que essa consulta é efetuada sempre que o usuário acessa o sistema, este ganho de velocidade é crucial para garantir o tempo de resposta e a fluidez da navegação do usuário.

4.5 Considerações finais

A arquitetura aqui proposta, serve de base para o desenvolvimento de aplicações hipermídia de ensino-aprendizagem que precisam manter dados a respeito das interações do usuário com o ambiente. Uma característica do *framework* que merece destaque é a arquitetura de três camadas.

> "A arquitetura de três camadas é uma nova área de crescimento para a computação cliente/servidor porque ela cumpre os requisitos de aplicações de larga escala para Internet e *intranets*. Na teoria, sistemas cliente/servidor de três camadas são mais escaláveis, robustos e flexíveis. Além disso, tais sistemas permitem a integração de dados de fontes diversas. Aplicações de três camadas são mais fáceis de gerenciar e de distribuir uma vez que a maioria do código executa nos servidores."
>
> (ORFALI et al., 1999, p.24)

A adoção do modelo de três camadas para a arquitetura do *framework* proposto neste trabalho não garante as características de escalabilidade e extensibilidade do sistema. Entretanto, pode-se dizer que o mesmo herda tais características do

modelo de três camadas visto que a arquitetura utilizada no *framework* segue as diretrizes para sistemas de três camadas.

Quanto à escalabilidade do *framework,* havendo necessidade de aumentar o número de usuários atendidos pelo sistema, é possível adotar uma abordagem diferenciada no servidor de aplicação permitindo a utilização compartilhada de múltiplos servidores.

Também é possível utilizar diferentes plataformas para executar o servidor de banco de dados por exemplo. Essa flexibilidade permite que o *framework* passe por refinamentos consecutivos sem sofrer restrições de *software* e hardware significativas.

Outra característica importante é que o intercâmbio de dados entre cliente e servidor é otimizado e portanto não exige muito da banda de comunicação. Conseqüentemente o modelo aqui proposto serve tanto para aplicações em intranets como para a Internet. No entanto, as dificuldades inerentes da Internet são maiores do que as de uma intranet tais como questões associadas à qualidade de serviço. A utilização desse modelo e aplicações para a Internet demanda portanto um estudo específico, que foge ao escopo desse trabalho.

A escolha das ferramentas e soluções tecnológicas da Microsoft utilizadas na arquitetura do *framework* está diretamente relacionada ao contexto no qual este trabalho está inserido. No início dos trabalhos do grupo de pesquisas do Geometrando, os recursos de *software* disponíveis eram o Microsoft Windows NT Server bem como o pacote Microsoft Back Office que inclui o servidor HTTP Microsoft Internet Information Server. O conhecimento e experiência prévia do autor também foram decisivos para a escolha dessas ferramentas.

5. CONCLUSÕES E RECOMENDAÇÕES

5.1 Conclusões finais

A concepção e implementação do *framework* apresentado nesse trabalho, teve como sua maior contribuição, a formação de uma infra-estrutura de *software* para viabilizar a armazenagem e recuperação de variáveis de interação do usuário em ambientes hipermídia de aprendizagem.

Inicialmente, apresentou-se uma revisão bibliográfica sobre banco de dados, tecnologias de rede e arquitetura de sistemas computacionais. Os estudos realizados pelo autor sobre esses temas formam o tripé de sustentação desse trabalho.

A apresentação do *framework* propriamente dito inicia com a enumeração das variáveis de interação do usuário, que são as informações de interesse para este trabalho. Merecem destaque os elementos da arquitetura de três camadas utilizada no *framework*:

- código no *front-end* para monitoramento das variáveis;
- utilização de *middleware* para as comunicações intercamadas;
- processamento de requisições e documentos XML no servidor de aplicação;
- armazenamento e recuperação de dados no servidor de banco de dados.

A implementação atual do *framework* privilegia a execução de tarefas no lado do cliente com uma baixa carga no servidor de aplicação. As comunicações entre cliente/servidor também são minimizadas – ocorrem somente na entrada e saída do usuário - para otimizar a utilização de recursos da rede. O desenvolvimento de algumas aplicações pode exigir uma configuração diferente da arquitetura atual do *framework*. As possibilidades de refinamentos técnicos são aprofundadas no item 5.2.2.

Por final, apresentou-se o banco de dados relacional utilizado para armazenar os dados de interação dos usuários. A descrição detalhada das entidades é feita no

dicionário de dados e utilizou-se a notação *crow´s foot* para a representação gráfica do modelo entidade-relacionamento do banco de dados.

Um resultado direto desse trabalho é que através da aplicação do *framework* em um ambiente hipermídia de aprendizagem, pode-se garantir a persistência das variáveis de interação do usuário. Logo, independentemente do local e quantidade de vezes que um usuário acessar o ambiente, um estado consistente de suas interações prévias e atuais será mantido.

A validação do *framework* não foi realizada formalmente pois deverá ocorrer quando o Geometrando for aplicado para seus usuários finais. No entanto, foram efetuados uma série de testes práticos com os diferentes módulos implementados, que sugerem não haver problemas quanto a perda de dados ou inconsistência. Uma possibilidade interessante para validar o *framework* é utilizar uma técnica semelhante à de avaliação de interfaces por testagem.

> “A avaliação por testagem é uma ferramenta de pesquisa com raízes na metodologia experimental clássica. Alguns usuários são escolhidos para participar de tarefas, interagindo com a interface enquanto são observados por avaliadores em um laboratório de usabilidade. Idealmente, um laboratório desta categoria deve ser equipado com câmeras de vídeo, visando capturar as ações e reações dos usuários no processo de interação com o sistema, assim como os diferentes estados da interface. Espelhos falsos também são usados para se garantir a observação dos usuários de forma a minimizar a presença intrusiva dos avaliadores.“
>
> (RODRIGUES, 2002, p. 25)

De forma semelhante, pode-se empregar esta técnica para aferir a robustez e a confiabilidade do *framework* efetuando uma aplicação com diferentes usuários. A utilização de câmaras de vídeo que gravem as ações do usuário bem como a informação temporal permitirá conferir, a posteriori, se as visitas, respostas, eventos e ações do usuário estão sendo mantidas de acordo com o esperado.

Uma alternativa para esse método é a utilização de um *software* para captura de tela tal como o *Techsmith Camtasia*. Esta ferramenta grava em formato de vídeo digital otimizado, todas as interações de tela incluindo a movimentação e os cliques do *mouse* efetuados pelo usuário. Através desse vídeo digital também pode ser realizada uma comparação entre os valores armazenados no banco de dados e a informação capturada no vídeo.

Sugere-se a utilização desse método como coadjuvante num processo formal de validação, levando em consideração diversos aspectos relevantes ao *framework*. Em ambas alternativas devem ser levadas em consideração questões relativas ao direito de imagem dos usuários que participam do teste.

Outra questão relevante diz respeito ao rastreamento de interações do usuário com um ambiente de ensino-aprendizagem realizadas através desse *framework*. É imprescindível que o usuário tenha consciência de que suas ações estão sendo gravadas. Sugere-se que essa informação seja passada no momento da criação da conta pessoal de acesso ao sistema. Uma alternativa é que a conta só seja criada mediante a concordância com um termo de utilização que esclareça as ações realizadas pelo sistema que não são visíveis ao usuário final.

É importante ressaltar que os resultados desse trabalho servem de base para a realização de diferentes pesquisas visando o aprimoramento dos ambientes de EIAC, especificamente as hipermídias pedagógicas. Outro fator relevante é que apesar de ter sido concebido para os sistemas desenvolvidos no HIPERLAB/EGR/UFSC, particularmente o Geometrando, os resultados deste trabalho podem ser aplicados em outros ambientes de ensino-aprendizagem voltados para a *web,* sendo relevantes também para os desenvolvimentos na área de ensino à distância.

A meta em longo prazo do grupo de pesquisas é definir um modelo para a criação de hipermídias de aprendizagem, que levam em conta as necessidades e características individuais do aprendiz. Pretende-se dotar o sistema de um comportamento pró-ativo (tomando decisões por conta própria) para aprimorar o processo de ensino-aprendizagem auxiliado por computador.

5.2 Sugestões para trabalhos futuros

5.2.1 Pesquisas

A realização desse trabalho, bem como os debates periódicos do grupo de pesquisa do Geometrando, permitiram vislumbrar diferentes encaminhamentos para pesquisas em diferentes áreas, tais como:

- avaliação do comprometimento do aluno com o processo de ensino-aprendizagem e avaliação da construção de conhecimento através da análise das variáveis de interação do usuário (Ulbricht, 1997);
- aplicação de técnicas modernas de extração de conhecimento de banco de dados como o *data warehouse,* OLAP e *data mining* (ver item 2.7);
- concepção e implementação de um sistema de apresentação gráfica interativa das informações de usuário extraídas do banco de dados;
- concepção e implementação de agentes inteligentes capazes de tomar decisões de forma pró-ativa tais como o redirecionamento da navegação do usuário (hipermídia adaptativa);
- concepção e implementação de um sistema de inteligência artificial que leva em consideração aspectos pedagógicos para melhorar o processo de ensino-aprendizagem;
- avaliação da coerência e homogeneidade do conteúdo e interface, permitindo a correção de pontos críticos.

Uma linha de pesquisas que merece destaque é o desenvolvimento de agentes inteligentes para hipermídias pedagógicas. Acredita-se que através de um comportamento inteligente pró-ativo, o sistema de EIAC pode analisar as variáveis de interação para efetuar recomendações ao usuário, respeitando suas características individuais.

Esse tipo de *feedback* inteligente pode ser alcançado através de processamentos síncronos ou assíncronos utilizando técnicas de inteligência artificial e hipermídia adaptativa.

> "As tecnologias adaptativas podem contribuir para diversos rumos na pesquisa e desenvolvimento de sistemas educacionais para *web*. A apresentação adaptativa pode melhorar a usabilidade da apresentação de conteúdo. O suporte à navegação adaptativa e o sequenciamento adaptativo podem ser utilizados para o controle global do conteúdo e para ajudar o aprendiz a selecionar os testes e tarefas mais relevantes. O apoio a resolução de problemas e a análise inteligente de soluções podem melhorar significativamente a realização das tarefas, provendo interatividade e *feedback* adaptativo inteligente."
>
> (Brusilovsky, 1998, p.8)

5.2.2 Aspectos Técnicos

Além das pesquisas que podem ser efetuadas como extensão desse trabalho, há a possibilidade de refinar o *framework* do ponto de vista técnico. Apesar de utilizar uma arquitetura baseada no modelo de três camadas, o sistema aqui proposto pode necessitar algumas alterações ou otimizações, no caso de ser aplicado, por exemplo, para uma grande quantidade de usuários e aplicações simultâneas.

As características de escalabilidade e extensibilidade presentes na arquitetura de três camadas são especialmente maximizadas com a utilização de uma ferramenta de gerenciamento de transação (TP Monitor - *Transaction Processing Monitor*). Uma das vantagens dessas ferramentas é que a aplicação torna-se independente do SGBD específico no qual os dados são armazenados, limitando a capacidade dos fabricantes de SGBD de prender os clientes a seus produtos. Os monitores de gerenciamento de transações controlam o tráfico que conecta centenas ou milhares de clientes com as aplicações e os recursos do *back-end*. Além do gerenciamento de transações, assegura-se que estas completem com precisão e disponibilizam-se serviços de balanceamento de carga, filas transacionais e tolerância a falhas (Orfali et al., 1999).

Os monitores de gerenciamento de transações em conjunto com as tecnologias de servidores de objetos distribuídos (CORBA e DCOM) estão formando uma classe de

produtos chamados Monitores de Transações de Objetos (OTM - Object Transaction Monitors). Além dos serviços de objetos – metadados, invocações dinâmicas, persistência, relacionamentos, eventos, gerenciamento de versões, segurança etc., os OTM´s possuem os serviços de gerenciamento de transações e podem vir a se tornar os coordenadores de objetos distribuídos na Internet e intranets. As funcionalidades dos OTM´s aumentam a eficiência e segurança dos sistemas cliente/servidor modernos como um todo e garantem a flexibilidade para a extensão e expansão dos mesmos (Orfali et al., 1999).

Um aspecto importante da utilização de gerenciadores de transação é a transferência da lógica da aplicação para a camada central da arquitetura de três camadas. É comum utilizar procedimentos armazenados que executam no servidor de banco de dados, mas essa é uma abordagem que tem algumas deficiências sérias.

> “Uma desvantagem dos procedimentos armazenados é que eles permitem uma menor flexibilidade do que o SQL dinâmico remoto. Além disso, os procedimentos armazenados podem ter sua eficiência diminuída se os planos de execução não forem atualizados para aproveitar as estatísticas do otimizador de consultas – o SQL dinâmico cria um plano novo para cada execução. Outra desvantagem é que não há sincronização de transações – isto é, confirmação de duas fases – entre os procedimentos armazenados; cada procedimento armazenado é uma transação separada. Outro problema é que os procedimentos armazenados são lentos – especialmente quando comparados a uma linguagem compilada. “
>
> (Orfali et al., 1999, p.223)

O desenvolvimento de aplicações cliente/servidor pode beneficiar-se dos gerenciadores de transações da mesma forma que vêm utilizando uma solução pronta para armazenamento e recuperação de dados (SGBD). Os gerenciadores de transações fornecem uma abstração importante para a camada do meio e implementam uma infra-estrutura eficiente, segura e testada.

A arquitetura de três camadas com os OTM's é bastante robusta e é uma tendência que pode se consolidar nos próximos anos. Desde os sistemas cliente/servidor mais simples até os ambientes distribuídos com dezenas de servidores espalhados pela rede podem beneficiar-se dos serviços dessa tecnologia.

Por esses motivos, tornam-se claras as vantagens de adaptar o *framework* aqui proposto para trabalhar com gerenciadores de transações e de objetos distribuídos, tais como garantir a independência do SGBD e viabilizar eventuais aumentos no número de usuários e futuras extensões do sistema.

Outra vantagem de utilizar uma solução completa para a camada de aplicação é facilitar o desenvolvimento de algumas aplicações listadas no item 5.2.1. Por exemplo, um sistema que utiliza inteligência artificial para tratar as variáveis do usuário com o objetivo de efetuar decisões pró-ativas certamente exige um processamento considerável na camada central da arquitetura de três camadas. A utilização de um gerenciador de transações e sua infra-estrutura de *software*, tende a facilitar todo o ciclo de desenvolvimento desse tipo de aplicação.

Um outro exemplo que pode ser apresentado, é uma aplicação que necessite tratar as variáveis de interação do usuário de forma síncrona - a medida que vão acontecendo – tal como agentes inteligentes voltados para a hipermídia adaptativa, a utilização da infra-estrutura de um gerenciador de transações na camada central facilitaria seu desenvolvimento significativamente.

Outra mudança que pode trazer resultados importantes para o *framework* seria utilizar o XML como padrão para armazenamento de conteúdo. Associado a um gerador de conteúdo também padronizado e juntamente com a tecnologia XSL[31], tal sistema teria controle sobre a estrutura da informação armazenada nos documentos XML, facilitando a atualização e pesquisa de conteúdo bem como a apresentação diferenciada da informação.

---

[31] Marchal (2000) descreve o XML *Stylesheet Language* (XSL) - Linguagem de guia de estilos para XML – como uma linguagem avançada para transformação de documentos XML com fins de apresentação da informação formatada em estilos.

6. FONTES BIBLIOGRÁFICAS

ALESSI, S. M.; TROLLIP, S. R. **Multimedia for learning : methods and developments**. 3rd ed. Needham Heights: Allyn & Bacon, 2001.

BERNERS-LEE, T.; CAILLIAU, R.; LUOTONEN, A.; NIELSEN, H. F.; SECRET, A. **The World-Wide Web**. In Communications of the ACM Vol. 37, n. 8, p.76-82, 1994.

BERRY, G. C.; CHASE, J. S.; COHEN, J. A.; COX, L. P.; VAHDAT, A. **Toward Automatic State Management for Dynamic Web Services**. 1999. Acessado em 10 set 2002. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/413297.html

BIUK-AGHAI, R. P. **Supporting Distance Education over the Internet.** 1998. Acessado em 10 mar 2003. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/51729.html

BOCHENSKI, B. **Implementando sistemas cliente/servidor de qualidade.** São Paulo: Makron Books, 1995.

BORLAND. **Interbase: cross-platform embedded database - Interbase Overview**. 2001. Acessado em 11 out 2001. Online. Disponível na Internet via WWW: http://www.borland.com/interbase/ib6/overview.pdf

BRUSILOVSKY, P. **Adaptive Educational Systems on the World-Wide-Web: A Review of Available Technologies.** 1998. Acessado em 12 Nov 2002. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/brusilovsky98adaptive.html

BUGAY, E. L.; ULBRICHT, V. R. **Hipermídia**. Florianópolis: Bookstore, 2000.

BUNEMAN, P. **Semistructured Data**. 1997. Acessado em 12 set 2002. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/buneman97semistructured.html

CELKO, J. **SQL for smarties: advanced SQL programming.** 2nd ed. – San Francisco: Morgan Kaufman, 2000.

CHEN, P**. The Entity-Relationship Model: Toward a Unified View of Data**. ACM Transactions on Database Systems, Vol. 1, Nr 1. New York: ACM Press, 1976.

CHUNG, J. **Objects and Relational Databases.** New York: ACM Press, 1995.

CODD, E. F. **Relational Databases: A practical foundation for productivity**, New York: ACM Press, 1982.

COSTA, S. F. **Método Científico - Os Caminhos da Investigação**. São Paulo: Editora HARBRA, 2001.

CYBIS, Walter. **Apostila do LabUtil: Recomendações para Desgin Ergonômico de Interfaces**. 2003. Acessado em 19 jul 2003. Online. Disponível na Internet via WWW: http://www.labiutil.inf.ufsc.br/apostila.htm

DIAS, M. H. **Um modelo de formalização do processo de desenvolvimento de sistemas de descoberta de conhecimento em banco de dados**. Florianópolis, 2001. Tese (Doutorado em Engenharia de Produção) - Programa de Pós-graduação em Engenharia de Produção, UFSC, 2001.

FRATERNALI, P. **Tools and Approaches for Developing Data-Intensive Web Applications: A Survey.** In ACM Computing Surveys, Vol. 31, n. 3, (Sep. 1999), p.227-262.

GILLIES, K.; WALKER, S.; DENLINGER, D.; KOTTURI, D**. Description of an XML-based Phase 1 Document.** 1999. Acessado em 10 out 2002. Online. Disponível na Internet via WWW:
http://citeseer.nj.nec.com/328742.html

HAROLD, E. R. **XML Bible**. Foster City: IDG Books, 1999.

HAYES, F. **The story so far**. 2002. Acessado em 30 abr 2002. Online. Disponível na Internet via WWW:
http://www.computerworld.com/databasetopics/data/story/0,10801,70102,00.html

HERRING, C.; REES, M.; RHODES, B. **Microsoft First Contact: The Borg Experiment**. Acessado em 30 abr 2002. Online. Disponível na Internet via WWW:
http://citeseer.nj.nec.com/337138.html

HODGINS, H. W. **Into the future: a vision paper**. 2000. Acessado em 20 de fev 2003. Online. Disponível na Internet via WWW:
http://www.learnativity.com/download/MP7.PDF

JONES, A. R. **Mastering Active Server Pages 3**. San Francisco: SYBEX, 2000.

LAKATOS, E. M.; MARCONI, M. de A. **Técnicas de Pesquisa**. São Paulo: Atlas, 1996.

KHAN, B. H. **Web based instruction**. New Jersey: Educacional Tehcnology Publications, 1997.

KORTH, H. F.; SILBERSCHATZ, A. **Sistema de Banco de Dados**. 2ª ed. rev. – São Paulo: Makron Books, 1995.

LEITE, L. L.P. **Introdução aos sistemas de gerência de banco de dados.** São Paulo: Edgard Blücher, 1980.

LENDING, D.; CHERVANY, N. L. **The Use of CASE Tools**. New York: ACM Press, 1998.

LIE, H. W.; SAARELA, J. **Multipurpose web publishing using HTML, XML, and CSS**. In ACM Computing Surveys, Vol. 42, no. 10, (Oct. 1999), p.95-101.

LOSHIN, P. **Relational Databases**. 2001. Acessado em 20 abr 2002. Online. Disponível na Internet via WWW: http://www.computerworld.com/databasetopics/data/story/0,10801,55918,00.html

MANOLESCU, I.; FLORESCU, D.; KOSSMANN, D. **Pushing XML Queries inside Relational Databases**. 2001. Acessado em 20 Fev 2002. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/manolescu01pushing.html

MARCHAL, B. **XML by example**. Indianapolis: Que, 2000.

MICROSOFT. **Microsoft Universal Data Access Web Site**. 2001. Acessado em 5 out 2001. Online. Disponível na Internet via WWW: http://www.microsoft.com/data/oledb/default.htm.

MYERS, B. A.; HOLLAN, J.; CRUZ, I.; BRYSON, S.; BULTERMAN, D.; CATARCI, T.; CITRIN, W.; GLINERT, E.; GRUDIN, J.; IOANNIDIS, Y. **Strategic directions in human-computer interaction**. In ACM Computer Surveys, Vol. 28, n. 4, p. 794-809, 1996.

NIELSEN, J. **Multimedia & Hypertext: The Internet and Beyond**. Mountain View: Academic Press, 1995.

_____. **Computer Science and Engineering Handbook**. Boca Raton: CRC Press Inc., 1996.

_____. **Usability Engineering**. California: Academic Press, 1993.

ORFALI, R.; HARKEY, D.; EDWARDS, J. **Client/Server Survival Guide**. 3rd ed. – New York: John Wiley & Sons, 1999.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of Distributed Database Systems**. New Jersey: Prentice-Hall, 1999.

PALOFF, R. M., PRATT, K. **Building learning communities in cyberspace**. San Francisco: Jossey-Bass, 1999.

RODRIGUES, D. W. L. **Uma Avaliação Comparativa de Interfaces Homem-Computador em Geometria Dinâmica**. Florianópolis, 2002. Dissertação (Mestrado em Engenharia de Produção) - Programa de Pós-graduação em Engenharia de Produção, UFSC, 2002.

ROWE, J. **Hypertext to hypermedia and beyond – the evolution continues**. In ACM SIGDOC ´97, 1997, p.237-240.

SEARCHENTERPRISELINUX.COM. **SearchEnterpriseLinux.com Definitions**. 2003. Acessado em 25 fev 2003. Online. Disponível na Internet via WWW: http://searchenterpriselinux.techtarget.com/sDefinition/0,,sid39_gci212709,00.html

SIMONS, P.; BABEL, R. **FastCGI – The Forgotten Treasure**. 2001. Acessado em 20 set 2002. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/475533.html

SOARES, L. F.; LEMOS, G.; COLCHER, S. **Redes de computadores: das LANs, MANs e WANs às redes ATM**. Rio de Janeiro: Campus, 1995.

SOUZA, A. L. **A Reinvenção das Organizações Educacionais na Sociedade do Conhecimento: o uso da Internet em Associações de Educação à Distância.** Florianópolis, 2000. Dissertação (Mestrado em Engenharia de Produção) - Programa de Pós-graduação em Engenharia de Produção, UFSC, 2000.

SVENNES, B.T. **A prestudy of the ESERNET Web**., 2001. Acessado em 10 out 2002. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/497715.html

TANENBAUM, A. S. **Redes de Computadores**. Tradução da 3 ed. Original. Rio de Janeiro: Campus, 1997.

TEOREY, T. J. **Database modeling & design**. San Francisco: Morgan Kaufman, 1999.

TITTEL, E.; GAITHER, M.; HASSINGER, S. **World Wide Web com HTML e CGI: bíblia do programador**. São Paulo: Berkeley Brasil, 1996

ULBRICHT, V. R. **Modelagem de um Ambiente Hipermídia de Construção do Conhecimento em Geometria Descritiva.** Florianópolis, 1997. Tese (Doutorado em Engenharia de Produção) - Programa de Pós-graduação em Engenharia de Produção, UFSC, 1997.

ULBRICHT, V. R.; VANZIN, T.; FERREIRA, C. L.; FIGUEIREDO, L. F. G. **Integrando as geometrias e a arte através da hipermídia**. Revista Educação Gráfica, 2001.

VALENTE, J. A. **Computadores e conhecimento: repensando a educação**. Campinas: Unicamp, 1993.

YE, W. **Dynamic Web Page Design and Implementation**. 1999. Acessado em 20 jun 2002. Online. Disponível na Internet via WWW: http://citeseer.nj.nec.com/ye99dynamic.html.

7. ANEXOS

7.1 Script SQL de geração do banco de dados

```
/******************************************************************************
 * Geradores – criam valores subsequentes para funcionarem como identificadores
******************************************************************************/

CREATE GENERATOR GEN_ANOTACAO_ID;

CREATE GENERATOR GEN_EVENTO_ID;

CREATE GENERATOR GEN_PAGINA_ID;

CREATE GENERATOR GEN_PERGUNTA_ID;

CREATE GENERATOR GEN_RESPOSTA_ID;

CREATE GENERATOR GEN_USUARIO_ID;

CREATE GENERATOR GEN_VISITA_ID;

COMMIT;

/******************************************************************************
 * Domínios – funcionam como tipos de variáveis. Seu uso padroniza os campos de um
determinado tipo garantindo a integridade referencial
******************************************************************************/

CREATE DOMAIN DESCRICAO AS
 VARCHAR(30) CHARACTER SET ISO8859_1 COLLATE ISO8859_1;

CREATE DOMAIN EMAIL AS
 VARCHAR(50) CHARACTER SET ISO8859_1 COLLATE ISO8859_1;

CREATE DOMAIN FONE AS
 VARCHAR(17) CHARACTER SET ISO8859_1 COLLATE ISO8859_1;

CREATE DOMAIN NOME AS
 VARCHAR(40) CHARACTER SET ISO8859_1 NOT NULL COLLATE ISO8859_1;

CREATE DOMAIN PAGINA_REF AS
 VARCHAR(60) CHARACTER SET ISO8859_1 COLLATE ISO8859_1;
```

```
CREATE DOMAIN PERGUNTA AS
 VARCHAR(20) CHARACTER SET ISO8859_1 COLLATE ISO8859_1;

CREATE DOMAIN USUARIO_ID AS
 SMALLINT NOT NULL;

CREATE DOMAIN USUARIO_LOGIN AS
 VARCHAR(12) CHARACTER SET ISO8859_1 NOT NULL COLLATE ISO8859_1;

CREATE DOMAIN VISITA_ID AS
 INTEGER NOT NULL;

COMMIT;

/******************************************************************************
 * Tabelas- as diversas tabelas do sistema onde os dados proprimamente ditos ficam
armazenados. As chaves primárias e estrangeiras são definidas mais abaixo
******************************************************************************/

CREATE TABLE ANOTACOES
(
        ANOTACAO_ID             INTEGER         NOT NULL,
     ANOTACAO_TEXTO             VARCHAR(32000)  CHARACTER SET ISO8859_1,
 ANOTACAO_DATA_HORA             TIMESTAMP,
    ANOTACAO_TITULO             VARCHAR(64)     CHARACTER SET ISO8859_1,
         USUARIO_ID             USUARIO_ID
);
COMMIT;

CREATE TABLE EVENTOS
(
          EVENTO_ID             INTEGER       NOT NULL,
   EVENTO_DATA_HORA             TIMESTAMP     NOT NULL,
        EVENTO_ALVO             VARCHAR(100)  CHARACTER SET ISO8859_1,
     TIPO_EVENTO_ID             INTEGER       NOT NULL,
         USUARIO_ID             USUARIO_ID,
          VISITA_ID             VISITA_ID
);
COMMIT;

CREATE TABLE TIPOS_EVENTO
(
       TIPO_EVENTO_ID             INTEGER       NOT NULL,
  TIPO_EVENTO_APELIDO             VARCHAR(30)   CHARACTER SET ISO8859_1,
TIPO_EVENTO_DESCRICAO             VARCHAR(255)  CHARACTER SET ISO8859_1
);
```

```
COMMIT;

CREATE TABLE MODULOS
(
          MODULO_ID             INTEGER         NOT NULL,
        MODULO_NOME             VARCHAR(30)     CHARACTER SET ISO8859_1,
   MODULO_DESCRICAO             DESCRICAO,
   MODULO_PEDAGOGIA             DESCRICAO
);
COMMIT;

CREATE TABLE PAGINAS
(
        PAGINA_ID               INTEGER         NOT NULL,
      PAGINA_NOME               VARCHAR(20)     CHARACTER SET ISO8859_1,
   PAGINA_PERIODO               CHAR(1)         CHARACTER SET ISO8859_1  NOT NULL,
   PAGINA_ASSUNTO               VARCHAR(60)     CHARACTER SET ISO8859_1,
        MODULO_ID               INTEGER         NOT NULL

);
COMMIT;

CREATE TABLE PERGUNTAS
(
         PERGUNTA_ID            INTEGER         NOT NULL,
       PERGUNTA_NOME            PERGUNTA,
       PERGUNTA_TIPO            INTEGER,
    PERGUNTA_ASSUNTO            VARCHAR(60)     CHARACTER SET ISO8859_1,
   PERGUNTA_RESPOSTA            VARCHAR(250)    CHARACTER SET ISO8859_1,
          PAGINA_ID             INTEGER         NOT NULL
);
COMMIT;

CREATE TABLE RESPOSTAS
(
          RESPOSTA_ID           INTEGER         NOT NULL,
         RESPOSTA_STR           VARCHAR(250)    CHARACTER SET ISO8859_1,
    RESPOSTA_CLICADAS           VARCHAR(250)    CHARACTER SET ISO8859_1,
   RESPOSTA_DATA_HORA           TIMESTAMP       NOT NULL,
         PERGUNTA_ID            INTEGER         NOT NULL,
          USUARIO_ID            USUARIO_ID,
           VISITA_ID            VISITA_ID
);
COMMIT;
```

```
CREATE TABLE USUARIOS
(
              USUARIO_ID             USUARIO_ID,
           USUARIO_LOGIN             USUARIO_LOGIN,
           USUARIO_SENHA             USUARIO_LOGIN,
            USUARIO_NOME             NOME,
          USUARIO_DTNASC             TIMESTAMP,
       USUARIO_DTCRIACAO             TIMESTAMP,
   USUARIO_ULTIMA_PAGINA             PAGINA_REF,
    USUARIO_ULTIMO_LOGIN             TIMESTAMP,
        USUARIO_ENDERECO             VARCHAR(50)  CHARACTER SET ISO8859_1,
          USUARIO_BAIRRO             VARCHAR(30)  CHARACTER SET ISO8859_1,
          USUARIO_CIDADE             VARCHAR(30)  CHARACTER SET ISO8859_1,
          USUARIO_ESTADO             VARCHAR(30)  CHARACTER SET ISO8859_1,
             USUARIO_CEP             VARCHAR(9)   CHARACTER SET ISO8859_1,
            USUARIO_FONE             FONE,
           USUARIO_EMAIL             EMAIL
);
COMMIT;

CREATE TABLE VISITAS
(
           VISITA_ID                 VISITA_ID,
    VISITA_DATA_HORA                 TIMESTAMP       NOT NULL,
    VISITA_TEMPO_STR                 VARCHAR(12)     CHARACTER SET ISO8859_1,
          USUARIO_ID                 USUARIO_ID,
           PAGINA_ID                 INTEGER         NOT NULL
);
COMMIT;

/*****************************************************************************
 * Índices- servem para a agilizar consultas comuns
*****************************************************************************/


CREATE ASC INDEX IDX_USUARIO_E_PERG__RESPOSTAS
ON RESPOSTAS (PERGUNTA_ID, USUARIO_ID);
COMMIT;

CREATE ASC INDEX IDX_USUARIO_ID__VISITAS
ON VISITAS (USUARIO_ID);
COMMIT;

CREATE ASC INDEX IDX_USUARIO_LOGIN__USUARIOS
ON USUARIOS (USUARIO_LOGIN);
COMMIT;
```

```
CREATE ASC INDEX IDX_USUARIO_NOME__USUARIOS
ON USUARIOS (USUARIO_NOME);
COMMIT;

CREATE ASC INDEX IDX_TIPO_EVENTO_APELIDO__EVENTOS
ON TIPOS_EVENTO (TIPO_EVENTO_APELIDO);
COMMIT;

/******************************************************************************
 * Procedimentos armazenados (cabeçalhos). O Código completo dos procedimentos
armazenados pode ser visto mais abaixo.
******************************************************************************/

SET TERM ^! ;
CREATE PROCEDURE CONTA_USUARIOS returns (NUM_USUARIOS Integer)
AS
BEGIN
  SUSPEND;
END^!
COMMIT^!

CREATE PROCEDURE INSERE_PAGINA (PAGINA_NOME VarChar(20), MODULO_ID Integer,
PAGINA_PERIODO Char(1)) returns (NOVA_PAG_COD Integer)
AS
BEGIN
  SUSPEND;
END^!
COMMIT^!

CREATE PROCEDURE INSERE_PERGUNTA (PAGINA_ID Integer, PERGUNTA_NOME VarChar(20),
PERGUNTA_TIPO Integer, PERGUNTA_RESPOSTA VarChar(250)) returns (NOVA_PERG_COD
Integer)
AS
BEGIN
  SUSPEND;
END^!
COMMIT^!

CREATE  PROCEDURE INSERE_USUARIO (USUARIO_LOGIN VarChar(12), USUARIO_SENHA
VarChar(12), USUARIO_NOME VarChar(40), USUARIO_DTNASC TimeStamp, USUARIO_EMAIL
VarChar(50), USUARIO_ENDERECO VarChar(100), USUARIO_BAIRRO VarChar(30),
USUARIO_CIDADE VarChar(25), USUARIO_ESTADO VarChar(2), USUARIO_CEP VarChar(9),
USUARIO_FONE VarChar(17))  returns (NOVO_USER_COD Integer)
AS
BEGIN
```

```
  SUSPEND;
END^!
COMMIT^!

CREATE PROCEDURE INSERE_VISITA (PAGINA_ID Integer, USUARIO_ID Integer,
VISITA_DATA_HORA TimeStamp, VISITA_TEMPO_STR VarChar(12)) returns (NOVA_VISITA_ID
Integer)
AS
BEGIN
  SUSPEND;
END^!
COMMIT^!

CREATE PROCEDURE INSERE_EVENTO (VISITA_ID Integer, USUARIO_ID Integer,
EVENTO_DATA_HORA TimeStamp, TIPO_EVENTO_APELIDO VarChar(30), EVENTO_ALVO
VARCHAR(100))
AS
BEGIN
  SUSPEND;
END^!
COMMIT^!

SET TERM ; ^!




/*****************************************************************************
 * Visões - Visões parciais dos dados em geral envolvendo campos de tabelas
diferentes associados através de uma cláusula de junção.
*****************************************************************************/

CREATE VIEW PERGUNTAS_PAGINAS
(
 MODULO_ID,
 PAGINA_NOME,
 PERGUNTA_ID,
 PAGINA_ID,
 PERGUNTA_NOME,
 PERGUNTA_TIPO,
 RESPOSTA_STR
) AS
select M.MODULO_ID, PAG.PAGINA_NOME, PERG.*  from
(
  (MODULOS M LEFT JOIN PAGINAS PAG ON M.MODULO_ID = PAG.MODULO_ID)
      RIGHT JOIN PERGUNTAS PERG ON PAG.PAGINA_ID = PERG.PAGINA_ID
```

```
);

COMMIT;

CREATE VIEW ULTIMAS_RESPOSTAS
(
 PERGUNTA_ID,
 USUARIO_ID,
 RESPOSTA_DATA_HORA
) AS
SELECT PERGUNTA_ID, USUARIO_ID, MAX(RESPOSTA_DATA_HORA) AS RESPOSTA_DATA_HORA FROM
RESPOSTAS
GROUP BY PERGUNTA_ID, USUARIO_ID;

CREATE VIEW ULTIMAS_ANOTACOES
(
 USUARIO_ID,
 ANOTACAO_DATA_HORA
) AS
SELECT  USUARIO_ID, MAX(ANOTACAO_DATA_HORA) AS ANOTACAO_DATA_HORA FROM ANOTACOES
GROUP BY USUARIO_ID;


CREATE VIEW ESTRUTURA_PERGUNTAS
AS
select PAG.PAGINA_NOME, PERG.PAGINA_ID, PERG.PERGUNTA_NOME, PERGUNTA_TIPO , R.*
from
(
  PAGINAS PAG LEFT JOIN PERGUNTAS PERG ON PAG.PAGINA_ID = PERG.PAGINA_ID
)  RIGHT JOIN RESPOSTAS R ON (R.PERGUNTA_ID = PERG.PERGUNTA_ID);


COMMIT;

/*****************************************************************************
 * Procedimentos Armazenados. Aumentam a eficiência por não precisarem ser
recompilados a cada vez e mantém a padronização de tratamento dos dados.
*****************************************************************************/

SET TERM ^! ;
ALTER PROCEDURE CONTA_USUARIOS returns (NUM_USUARIOS Integer)
AS
 BEGIN
  SELECT COUNT(DISTINCT USUARIO_ID) FROM USUARIOS
   INTO :NUM_USUARIOS;
END^!
```

```
COMMIT^!

ALTER PROCEDURE INSERE_PAGINA (PAGINA_NOME VarChar(20), MODULO_ID Integer,
PAGINA_PERIODO Char(1)) returns (NOVA_PAG_COD Integer)
AS
BEGIN
INSERT INTO PAGINAS (PAGINA_NOME, MODULO_ID, PAGINA_PERIODO) VALUES (:PAGINA_NOME,
:MODULO_ID, :PAGINA_PERIODO);
NOVA_PAG_COD = GEN_ID (GEN_PAGINA_ID,0);
SUSPEND;
END^!
COMMIT^!

ALTER PROCEDURE INSERE_PERGUNTA (PAGINA_ID Integer, PERGUNTA_NOME VarChar(20),
PERGUNTA_TIPO Integer, PERGUNTA_RESPOSTA VarChar(250)) returns (NOVA_PERG_COD
Integer)
AS
BEGIN
INSERT INTO PERGUNTAS (PAGINA_ID, PERGUNTA_NOME, PERGUNTA_TIPO, PERGUNTA_RESPOSTA)
VALUES (:PAGINA_ID, :PERGUNTA_NOME, :PERGUNTA_TIPO, :PERGUNTA_RESPOSTA);
NOVA_PERG_COD = GEN_ID (GEN_PERGUNTA_ID,0);
SUSPEND;
END^!
COMMIT^!

ALTER PROCEDURE INSERE_USUARIO (USUARIO_LOGIN VarChar(12), USUARIO_SENHA
VarChar(12), USUARIO_NOME VarChar(40),
USUARIO_DTNASC TimeStamp, USUARIO_EMAIL VarChar(50), USUARIO_ENDERECO VarChar(100),
USUARIO_BAIRRO VarChar(30),
USUARIO_CIDADE VarChar(25), USUARIO_ESTADO VarChar(2), USUARIO_CEP VarChar(9),
USUARIO_FONE VarChar(17))
 returns (NOVO_USER_COD Integer)
AS
BEGIN
  INSERT INTO USUARIOS (USUARIO_LOGIN, USUARIO_SENHA, USUARIO_NOME, USUARIO_DTNASC,
USUARIO_DTCRIACAO, USUARIO_ULTIMA_PAGINA,
  USUARIO_ULTIMO_LOGIN, USUARIO_ENDERECO, USUARIO_BAIRRO, USUARIO_CIDADE,
USUARIO_ESTADO, USUARIO_CEP, USUARIO_FONE, USUARIO_EMAIL)
  VALUES (:USUARIO_LOGIN, :USUARIO_SENHA, :USUARIO_NOME, :USUARIO_DTNASC, 'now',
'', 'now', :USUARIO_ENDERECO,
          :USUARIO_BAIRRO, :USUARIO_CIDADE, :USUARIO_ESTADO, :USUARIO_CEP,
:USUARIO_FONE, :USUARIO_EMAIL);

  NOVO_USER_COD = GEN_ID (GEN_USUARIO_ID,0);
  SUSPEND;
END^!
```

```
COMMIT^!

ALTER PROCEDURE INSERE_VISITA (PAGINA_ID Integer, USUARIO_ID Integer,
VISITA_DATA_HORA TimeStamp, VISITA_TEMPO_STR VarChar(12))
 RETURNS (NOVA_VISITA_ID Integer)
AS
BEGIN
  INSERT INTO VISITAS (PAGINA_ID, USUARIO_ID, VISITA_DATA_HORA, VISITA_TEMPO_STR)
VALUES
   (:PAGINA_ID, :USUARIO_ID, :VISITA_DATA_HORA, :VISITA_TEMPO_STR);

  NOVA_VISITA_ID = GEN_ID (GEN_VISITA_ID,0);
  SUSPEND;
END^!
COMMIT^!

ALTER PROCEDURE INSERE_EVENTO (VISITA_ID Integer, USUARIO_ID Integer,
EVENTO_DATA_HORA TimeStamp, TIPO_EVENTO_APELIDO VarChar(30), EVENTO_ALVO
VARCHAR(100))
AS
DECLARE VARIABLE TIPO_EVENTO_ID integer;
BEGIN
     SELECT TIPO_EVENTO_ID from TIPOS_EVENTO WHERE TIPO_EVENTO_APELIDO =
:TIPO_EVENTO_APELIDO INTO :TIPO_EVENTO_ID;

     INSERT INTO EVENTOS (VISITA_ID, USUARIO_ID, TIPO_EVENTO_ID, EVENTO_DATA_HORA,
EVENTO_ALVO) VALUES
     (:VISITA_ID, :USUARIO_ID, :TIPO_EVENTO_ID, :EVENTO_DATA_HORA, :EVENTO_ALVO);

  SUSPEND;
END^!
COMMIT^!


SET TERM ; ^!



/******************************************************************************
 * Disparadores - Disparam a execução de tarefas após a ocorrência de eventos.
******************************************************************************/

SET TERM ^! ;
CREATE TRIGGER CRIA_USUARIO FOR USUARIOS ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
  NEW.USUARIO_ID = GEN_ID(GEN_USUARIO_ID, 1);
```

```
END^!
SET TERM ; ^!


SET TERM ^! ;
CREATE TRIGGER CRIA_PAGINAS FOR PAGINAS ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
  NEW.PAGINA_ID = GEN_ID(GEN_PAGINA_ID, 1);
END^!
SET TERM ; ^!


SET TERM ^! ;
CREATE TRIGGER CRIA_VISITAS FOR VISITAS ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
  NEW.VISITA_ID = GEN_ID(GEN_VISITA_ID, 1);
END^!
SET TERM ; ^!


SET TERM ^! ;
CREATE TRIGGER CRIA_PERGUNTAS FOR PERGUNTAS ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
  NEW.PERGUNTA_ID = GEN_ID(GEN_PERGUNTA_ID, 1);
END^!
SET TERM ; ^!


SET TERM ^! ;
CREATE TRIGGER CRIA_RESPOSTAS FOR RESPOSTAS ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
  NEW.RESPOSTA_ID = GEN_ID(GEN_RESPOSTA_ID, 1);
END^!
SET TERM ; ^!


SET TERM ^! ;
CREATE TRIGGER CRIA_ANOTACOES FOR ANOTACOES ACTIVE BEFORE INSERT POSITION 0 AS
BEGIN
  NEW.ANOTACAO_ID = GEN_ID(GEN_ANOTACAO_ID, 1);
END^!
SET TERM ; ^!


SET TERM ^! ;
CREATE TRIGGER CRIA_EVENTO FOR EVENTOS ACTIVE BEFORE INSERT POSITION 0 AS
```

```
BEGIN
  NEW.EVENTO_ID = GEN_ID(GEN_EVENTO_ID, 1);
END^!
SET TERM ; ^!

/*****************************************************************************
 * Restrições – Restrições de Chave Primária (PK) e Chave Estrangeira (FK)
*****************************************************************************/

ALTER TABLE USUARIOS ADD CONSTRAINT PK_USUARIOS_ID PRIMARY KEY (USUARIO_ID);
COMMIT;

ALTER TABLE MODULOS ADD CONSTRAINT PK_MODULOS_ID PRIMARY KEY (MODULO_ID);
COMMIT;

ALTER TABLE PAGINAS ADD CONSTRAINT PK_PAGINAS_ID PRIMARY KEY (PAGINA_ID);
COMMIT;

ALTER TABLE VISITAS ADD CONSTRAINT PK_VISITAS_ID PRIMARY KEY (VISITA_ID);
COMMIT;

ALTER TABLE PERGUNTAS ADD CONSTRAINT PK_PERGUNTAS_ID PRIMARY KEY (PERGUNTA_ID);
COMMIT;

ALTER TABLE RESPOSTAS ADD CONSTRAINT PK_RESPOSTAS_ID PRIMARY KEY (RESPOSTA_ID);
COMMIT;

ALTER TABLE ANOTACOES ADD CONSTRAINT PK_ANOTACOES_ID PRIMARY KEY (ANOTACAO_ID);
COMMIT;

ALTER TABLE EVENTOS ADD CONSTRAINT PK_EVENTOS_ID PRIMARY KEY (EVENTO_ID);
COMMIT;

ALTER TABLE TIPOS_EVENTO ADD CONSTRAINT PK_TIPO_EVENTO_ID PRIMARY KEY
(TIPO_EVENTO_ID);
COMMIT;



ALTER TABLE ANOTACOES ADD CONSTRAINT FK_ANOTACOES__USUARIO_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;
```

```
ALTER TABLE EVENTOS ADD CONSTRAINT FK_EVENTOS__PAGINA_ID
  FOREIGN KEY (VISITA_ID) REFERENCES VISITAS (VISITA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE EVENTOS ADD CONSTRAINT FK_EVENTOS__USUARIO_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;

ALTER TABLE EVENTOS ADD CONSTRAINT FK_EVENTOS__EVENTO_TIPO_ID
  FOREIGN KEY (TIPO_EVENTO_ID) REFERENCES TIPOS_EVENTO (TIPO_EVENTO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE PAGINAS ADD CONSTRAINT FK_PAGINAS__MODULO_ID
  FOREIGN KEY (MODULO_ID) REFERENCES MODULOS (MODULO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE PERGUNTAS ADD CONSTRAINT FK_PERGUNTAS__PAGINA_ID
  FOREIGN KEY (PAGINA_ID) REFERENCES PAGINAS (PAGINA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE RESPOSTAS ADD CONSTRAINT FK_RESPOSTAS__PERGUNTA_ID
  FOREIGN KEY (PERGUNTA_ID) REFERENCES PERGUNTAS (PERGUNTA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE RESPOSTAS ADD CONSTRAINT FK_RESPOSTA__USER_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;
```

```
ALTER TABLE RESPOSTAS ADD CONSTRAINT FK_RESPOSTA__VISITA_ID
  FOREIGN KEY (VISITA_ID) REFERENCES VISITAS (VISITA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE VISITAS ADD CONSTRAINT FK_VISITAS__PAGINA_ID
  FOREIGN KEY (PAGINA_ID) REFERENCES PAGINAS (PAGINA_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;


ALTER TABLE VISITAS ADD CONSTRAINT FK_VISITAS__USUARIO_ID
  FOREIGN KEY (USUARIO_ID) REFERENCES USUARIOS (USUARIO_ID)
  ON DELETE CASCADE
  ON UPDATE CASCADE;
COMMIT;
```