UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CUROS DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

ESTUDO DO SGBD "CACHÉ" COM UMA APLICAÇÃO NA RESERVA DE VAGAS EM EVENTOS ACADÊMICOS VIA WEB

ALOISIO ARBEGAUS

BLUMENAU
2003

2003/2-03

ALOISIO ARBEGAUS

ESTUDO DO SGBD "CACHÉ" COM UMA APLICAÇÃO NA RESERVA DE VAGAS EM EVENTOS ACADÊMICOS VIA WEB

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Marcelo José Ferrari – Orientador

BLUMENAU
2003

2003/2-03

ESTUDO DO SGBD “CACHÉ” COM UMA APLICAÇÃO NA RESERVA DE VAGAS EM EVENTOS ACADÊMICOS VIA WEB

Por

ALOISIO ARBEGAUS

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

Presidente:	Prof. Marcelo José Ferrari, FURB
Membro:	Prof. Alexander Valdameri, FURB
Membro:	Prof. Dr.Oscar Dalfovo, FURB

Blumenau, 10 de 2003

Dedico este trabalho a minha esposa Daniele Cruz Corrêa Arbegaus pelo apoio, incentivo e carinho que recebi durante a elaboração deste trabalho; a minha família, pelo amor e incentivo em todos os momentos de minha vida; e a todos os amigos, que souberam compreender os desafios, e estiveram do meu lado ajudando a superá-los.

Que cada dia seja um novo desafio, e que tenhamos sabedoria para superá-los.

Aloisio Arbegaus

AGRADECIMENTOS

À Deus, pelo seu imenso amor e graça.

À Nossa Senhora Aparecida, pela iluminação em todos os momentos.

À minha família, que mesmo longe, sempre esteve presente.

À minha esposa, Daniele Cruz Corrêa Arbegaus, pelo apoio e compreensão.

Aos meus avós, pelos ensinamentos passados de geração para geração, especialmente ao meu avô Frederico Arbegaus.

Aos meus amigos, pelos empurrões e cobranças.

Ao meu orientador, Marcelo José Ferrari, por ter acreditado na conclusão deste trabalho.

RESUMO

Este trabalho consiste em um estudo sobre banco de dados orientado a objetos. Nesse estudo são avaliados os problemas e as facilidades encontradas durante o processo de desenvolvimento de um protótipo de sistema de reserva de vagas em eventos acadêmicos acessando o banco de dados Caché, de maneira orientada a objetos, a fim de testar o acesso via *Java Database Connectivity* (JDBC), e unificar a metodologia da programação orientada a objetos com uma forma de armazenamento não relacional.

Palavras chaves: Caché; Banco de dados; Banco de dados orientado a objetos; Eventos acadêmicos.

ABSTRACT

This work consists of a study on object-oriented database. In that study they are appraised the problems an the means found during the process of development of system prototype for reserve of vacancy in academic events access the database Caché, of way object-oriented, in order to test the access for Java Database Connectivity (JDBC), and to unify the methodology of programming object-oriented with a form of storage not relational.

Key-Words: Caché; Database; Database object-oriented; academic events.

LISTA DE ILUSTRAÇÕES

LISTA SIGLAS

API - *Application Programming Interface*

BDOO - Banco de dados orientado a objetos

BDR - Banco de dados relacional

C - Linguagem de progamação

CAD - *Computer-Aided Design*

CAM - *Computer-Aided Manufacture*

CASE - *Computer-Aided Software Enineering*

CGI - *Common Gateway Interface*

COBOL - *Commom Business Oriented Language*

CODASYL - *Conference on data systems and language*

COS - *Caché object script*

DBA - *Database Administrator*

DBTG - *Data base task group*

DDL - *Data description language*

DML - *Data manipulation language*

EJB - *Entreprise Java Beans*

ER - Entidade-Relacionamento

GUI - *Graphical User Interface*

HTML - *HyperText Markup Language*

HTTP - *HyperText Transfer Protocol*

JDBC - *Java database connectivity*

JSP - *Java server pages*

LAN - *Local area networks*

LPOO - Linguagens de programação orientadas a objetos

M - Comandos "M" da linguagem *Mumps*

OBDC - *Open database connectivity*

OCI - *Oracle calling interface*

OID - *Object Identify*

OO - Orientação a objetos

RAD - *Rapid Application Development*

SGBD - Sistema gerenciador de banco de dados

SGBDOO - Sistema gerenciador de banco de dados orientado a objetos

SQL - *Structed Query Language*

UML - *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Pinheiro (2000), o modelo de dados orientado a objetos possui diversas diferenças com relação ao modelo relacional tradicional. Essas diferenças dizem respeito à maneira de organizar a informação, representando o modelo estrutural. Além disso, um fator importante adicionado neste contexto é a possibilidade de estabelecer uma estrutura de funções e procedimentos, representando o modelo comportamental, que na verdade são os métodos dos objetos para manipular os dados armazenados. Dessa forma, o modelo de dados orientado a objetos permite o projeto estrutural e comportamental da base de dados, através da definição dos objetos com seus atributos e métodos. É com este propósito, de manipulação de objetos através da definição de atributos e métodos dos objetos, que o banco de dados *Caché* trabalha. Através de ferramentas do *Caché,* como o *Caché Studio* isto fica melhor representado, pois é nessa ferramenta que são definidas as classes com seus atributos e métodos.

Segundo Intersystems (2003a), o *Caché*, como banco de dados orientado a objetos, apresenta alta performance, sendo ideal para aplicações Java, pois oferece três diferentes modos de conexão: dados do *Caché* podem ser acessados com SQL via *Java Database Connectivity* (JDBC); classes *Caché* podem ser projetadas como classes Java e classes *Caché* podem ser projetadas como *Enterprise JavaBeans* (EJB).

Para mostrar as funcionalidades do *Caché*, suas ferramentas quando usadas com aplicações Java e modos de conexão, foi desenvolvido um protótipo que utiliza este banco de dados integrando com a WEB, que permite qualquer acadêmico conectado à internet efetuar uma reserva de vaga em um evento. Foram ainda utilizadas algumas tecnologias da internet como *Java Server Pages* (JSP) e *servlets*, que permitem um controle mais eficiente das informações para o usuário e JDBC que permite consultas SQL no banco de dados.

Para a especificação do protótipo foi utilizada a ferramenta *Rational Rose*, que utiliza a *Unified Modeling Language* (UML). A UML foi criada propondo atender às necessidades de especificação de sistemas orientados a objetos. Segundo Melo (2002), a UML proporciona uma forma padrão para a preparação de planos de arquitetura de projetos de sistemas, incluindo aspectos conceituais tais como processos de negócios e funções do sistema, além de itens concretos como classes escritas em determinada linguagem de programação, esquemas de bancos de dados e componentes de software reutilizáveis.

1.1 OBJETIVOS

O objetivo deste trabalho é mostrar como o *Caché* armazena, manipula e administra seus objetos, bem como, permite a uma aplicação WEB fazer consultas SQL no banco via JDBC.

Os objetivos específicos do trabalho são:

a) mostrar o uso da orientação a objetos desde o desenvolvimento de uma aplicação WEB até o armazenamento dos objetos no banco de dados *Caché*;
b) especificar, modelar e implementar um sistema básico para reservas de vagas em eventos acadêmicos como seminários, feiras, cursos, congressos e palestras, com interface WEB.

1.2 ESTRUTURA DO TRABALHO

Este trabalho encontra-se dividido da seguinte forma:

No capítulo 1 são apresentadas as justificativas deste trabalho, seus objetivos e sua organização.

No capitulo 2 é apresentado um breve histórico sobre sistemas gerenciadores de banco de dados incluindo seus conceitos, objetivos e a evolução de seus modelos.

No capítulo 3 são apresentados os conceitos relativos aos sistemas gerenciadores de banco de dados orientados a objetos, suas características e motivos do seu surgimento. Também é exposto como os conceitos de orientação a objetos e gerenciamento de banco de dados são adaptados aos banco de dados orientados a objeto.

No capítulo 4 é apresentado o banco de dados Caché, seus conceitos, seu modelo de dados, formas de acesso e a estrutura de seus objetos.

No capítulo 5 são apresentadas as vantagens e desvantagens dos bancos de dados quando acessados por páginas web.

No capítulo 6 é apresentada a especificação do protótipo desenvolvido.

O capítulo 7 apresentada as conclusões e sugestões para futuros trabalhos.

2 SISTEMA GERENCIADOR DE BANCO DE DADOS

Date (1991) define um Sistema de Gerenciamento de Banco de Dados (SGBD) como nada mais do que um sistema de manutenção de dados por computador que tem por objetivo principal manter as informações e disponibilizá-las a seus usuários quando solicitadas.

Para Korth (1997), um SGBD consiste em uma coleção de dados inter-relacionados e um conjunto de programas para acessa-los. O conjunto de dados referenciado como banco de dados contém as informações de uma empresa particular. Subentende-se uma empresa particular como qualquer instituição ou organização sobre a qual possam ser guardadas informações relevantes à mesma.

Junto à estrutura de um banco de dados está o modelo de dados, um conjunto de ferramentas para descrição, relacionamento, restrições e semântica de dados. Independente do modelo de dados do banco, o bom funcionamento do mesmo está intimamente ligado à definição ideal do esquema de dados. O conjunto de informações existente em um banco de dados em determinado momento denomina-se instância do banco de dados, (Baehr Jr., 1999).

2.1 EVOLUÇÃO DOS SGBD

Os precursores dos SGBD foram os sistemas de arquivo. Sem nenhuma espécie de esquema de dados, os sistemas de arquivo realizavam funções corriqueiras em arquivos, como manutenções, organizações e listagens de registros. Nos anos 50 e 60 surgiram alguns produtos de definição de dados, desenvolvidos por grandes empresas, como a IBM, General Eletric e Honeywell. Estes produtos deram origem à linguagem COBOL (*Commom Business Oriented Language* – Linguagem Comum Orientada para Negócios), desenvolvida em 1960 pela *Conference On Data Systems and languages* (CODASYL). O COBOL tinha um recurso inovador para a época: uma área do programa totalmente destinada a definição de dados, (Baehr Jr., 1999).

O CODASYL propôs uma extensão da linguagem COBOL para banco de dados e comissionou esta tarefa ao Data Base Task Group (DBTG). Esse fato propiciou o aparecimento, na década de 60, dos banco de dados de rede e hierárquico. Esses primeiros modelos de dados eram puramente navegacionais, não tinham forte embasamento teórico e não suportavam a independência física e lógica de dados. Em 1969, o DBTG definia a Data

Description Language (DDL) e a Data Manipulation language (DML), embasando a fundação dos SGBD, Khoshafian (1994).

Com a intenção de minimizar os problemas dos primeiros SGBD e propiciar maior flexibilidade para manipulação de grandes massas de dados o Dr. E.F.Codd introduziu, na década de 70, o modelo de dados relacional. Na década de 80, os produtos relacionais tornaram-se fortemente comerciais e amplamente populares, (Baehr Jr., 1999).

No final da década de 80, a maior parte dos SGBD comerciais baseavam-se no modelo relacional, hierárquico ou de rede. Contudo, existiam nessa época inúmeras propostas alternativas para banco de dados. Algumas delas limitaram-se a protótipos de laboratório e nunca chegaram a ser comercializados.

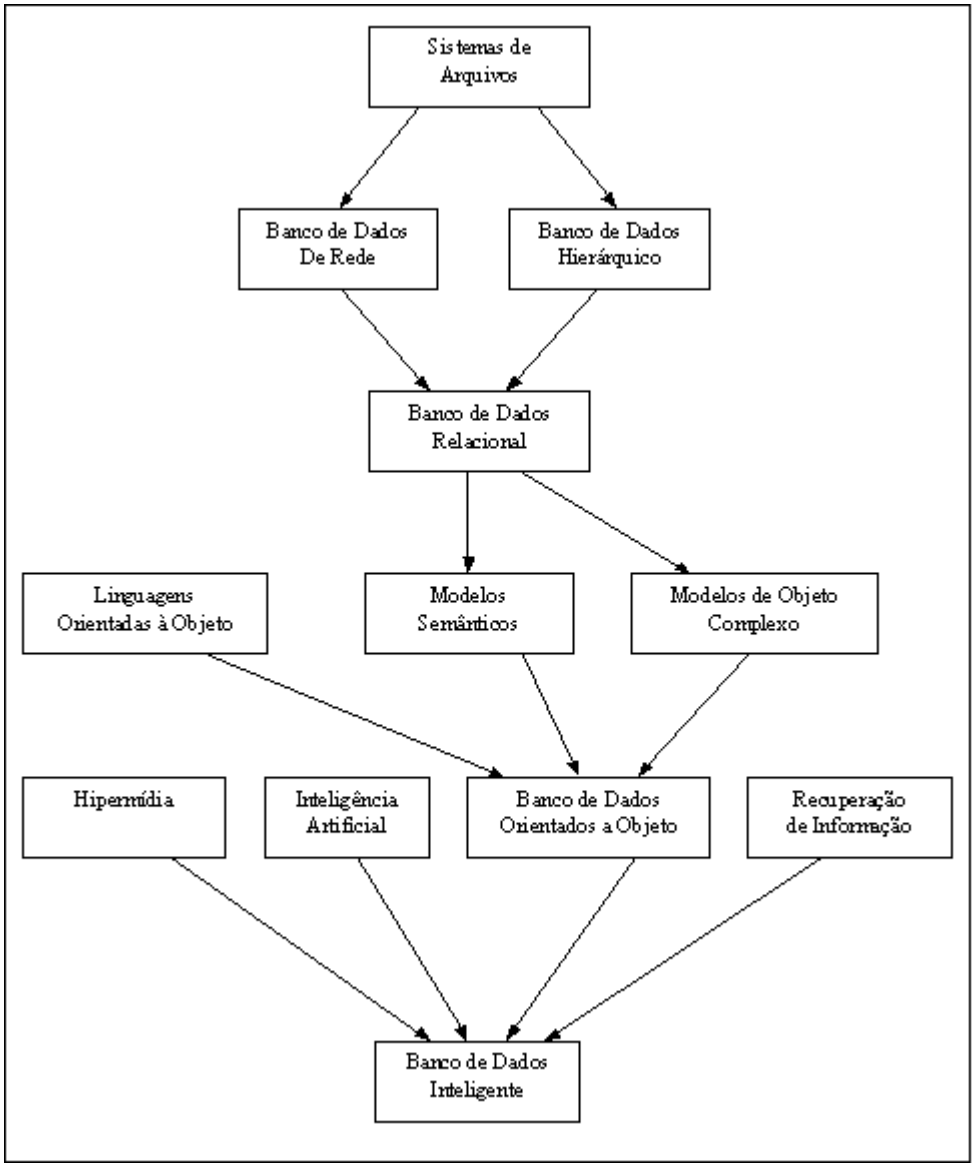
Uma das primeiras propostas alternativas para banco de dados foram os modelos de dados semânticos, que tiveram como precursor o modelo Entidade - Relacionamento (ER). Os modelos de dados semânticos tinham por objetivo modelar o mundo real o tanto quanto possível. Hoje, modelos de dados semânticos são freqüentemente utilizados para definir o esquema de dados de bancos de dados relacionais.

Similar aos modelos de dados semânticos os modelos de objetos complexos (outra proposta alternativa para banco de dados) também tentavam modelar o mundo real acrescentando mais semântica aos modelos de dados. Para tanto, eles tentavam estender o modelo de dados relacionais a fim de obter maior flexibilidade. Maiores explicações sobre modelos de objetos complexos podem ser encontrados em Khoshafian (1994) e Korth (1997).

Entretanto, a evolução dos SGBDs não está unicamente ligada à evolução e aperfeiçoamento dos modelos de dados. Na década de 90 um outro fator importante toma sua parte na história: a migração dos grandes bancos de dados de mainframes para arquiteturas cliente/servidor (processo que ficou conhecido como *downsizing*) e adesão, por parte de corporações menores, à tecnologia das redes locais (LAN do inglês *Local Area Networks*).

Nesse conturbado contexto surgem os bancos de dados orientados a objetos. Na verdade, sistemas e produtos de banco de dados orientados a objetos surgiram em meados da década de 80, mas tomaram força nos anos 90, com proliferação dos conceitos de orientação a objetos, Khoshafian (1994).

No processo de evolução dos SGBD, sintetizado na figura 2.2, é preciso uma visão ampla e lembrar dos banco de dados inteligentes. Estes bancos de dados interagem inteligência artificial, recuperação de informações, orientação a objetos e tecnologia de multimídia com bancos de dados. Dessa forma, capacidades adicionais para bancos de dados não devem limitar-se à orientação a objetos. Na próxima geração de banco de dados espera-se suporte à inferência (inteligência artificial), recuperação de textos (recuperação de informações) e tipos de dados multimídia (voz, textos, gráficos).



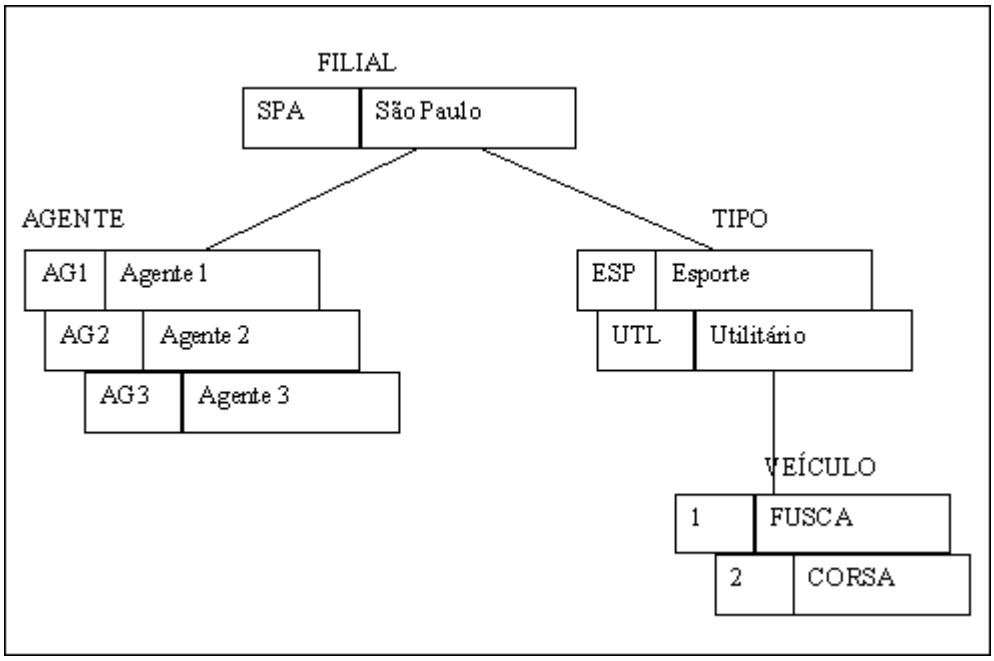
Fonte: Khoshafian (1994)

Figura 2.1 – A evolução dos Banco de Dados

2.1.1 O modelo hierárquico

Segundo Korth (1997), um banco de dados hierárquico consiste em uma coleção de registros que são conectados por meio de ligações. Cada registro é uma coleção de campos (atributos), os quais contém apenas um valor. Já Khoshafian (1994) define um banco de dados hierárquico como um conjunto de árvores (floresta) em que cada nó da árvore representa um conjunto de objetos (registros) do mesmo tipo. Date (1991) define um banco de dados hierárquico como um conjunto ordenado de árvores, mais precisamente, de um conjunto ordenado de ocorrências múltiplas de um tipo de árvore.

Cada registro, em um banco de dados hierárquico, pode ter quantos descendentes quiser, mas somente um ascendente (exceto a raiz, que não possui ascendentes). Dessa forma, as relações entre pais (nós ascendentes) e filhos (nós descendentes) são de um nó pai para zero ou muitos nós filhos (figura 2.3). Conforme a limitação de que cada registro pode ter apenas um nó pai seu conteúdo pode ser repetido várias vezes. Além de se desperdiçar muito espaço com a mesma informação, corre-se o risco de tornar inconsistente o banco se determinada atualização não for feita em todos os registros repetidos.



Fonte: adaptado de Kern (1994)

Figura 2.2 – Representação Hierárquica de Registros

O esquema de dados para o modelo hierárquico é o diagrama com estrutura de árvore. Ele consiste de dois componentes básicos:

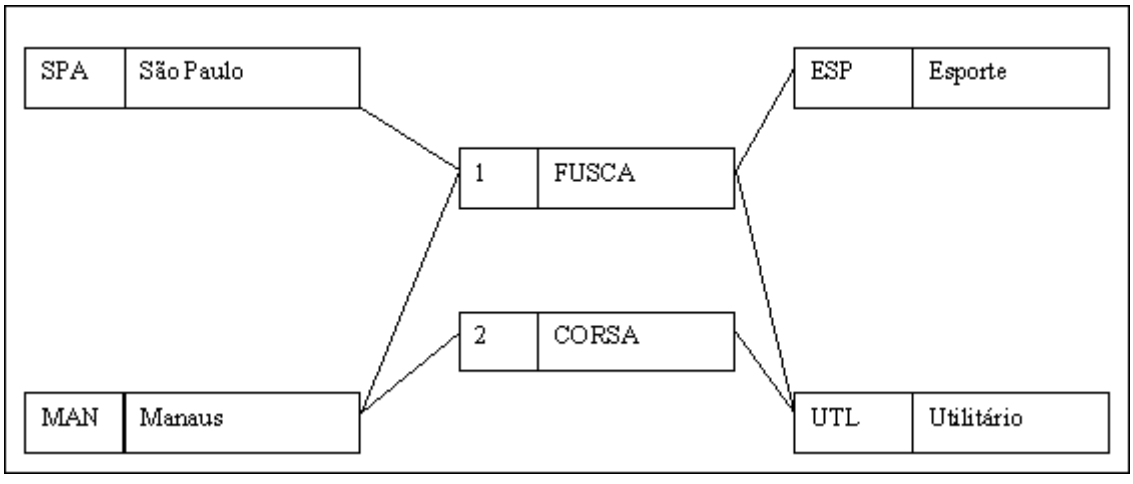
a) retângulos, que referenciam os tipos de registros;
b) linhas, que definem as ligações.

O modelo de dados mais utilizado para instanciar um banco de dados hierárquico consiste basicamente em associar um ponteiro a um registro para cada filho que esse registro possui.

2.1.2 O modelo de rede

Segundo Kern (1994), um banco de dados de rede consiste em uma visão de grão ou malha de ligações um para muitos entre os registros. Um tipo de registro pode estar envolvido em vários relacionamentos e pode ter diversos ascendentes e descendentes.

Khoshafian (1994) também considera o banco de rede um extensão do banco hierárquico, porém mais geral. Salienta que embora o único relacionamento suportado pelo banco de dados de rede seja o um para muitos, o mesmo registro poder ser filho com vários registros pais, rompedo a restrição dos banco de dados hierárquicos de que um filho só podia possuir um pai (figura 2.4). Isto foi possível através da substituição da estrutura de armazenamento em árvore por uma estrutura na forma de uma coleção de registros interligados uns aos outros por meio de ligações.



Fonte: adaptado de Kern (1994)
Figura 2.3 – Representação em Rede de Registros

O esquema de dados para o modelo de rede é o diagrama de estrutura de dados. Ele consiste de dois componentes básicos:

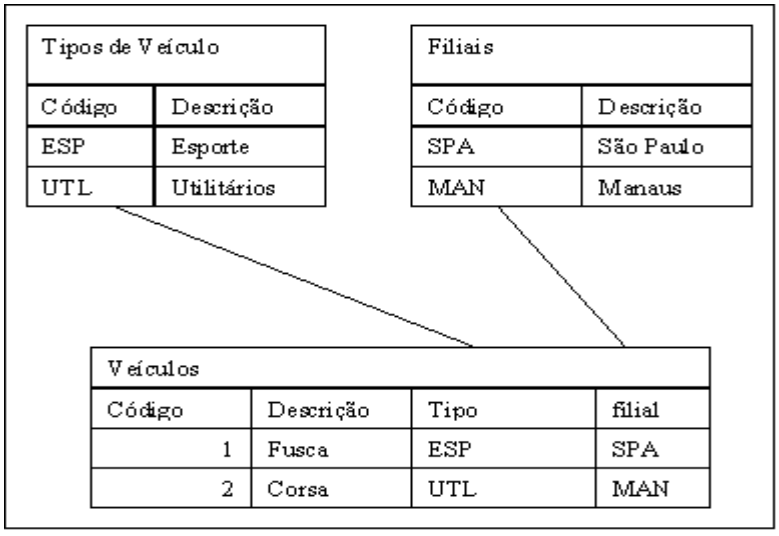
a) retângulos, que correspondem à tipos de registros;
b) linhas, que definem as ligações.

O modelo de dados mais utilizado para instanciar um diagrama de estrutura de dados consiste em implementar as ligações de um registro adicionado a ele compôs de ponteiros para cada registro aos quais está associado por meio de uma ligação. Cada registro deve ter um campo de ponteiro para cada ligação com a qual está associado.

2.1.3 O modelo relacional

O modelo de dados relacional foi criado por E.F. Codd e baseia-se em um único conceito: a tabela (figura 2.5). Segundo Date (1991) define um banco de dados relacional como um banco que só pode ser percebido pelo usuário como tabelas e nada além de tabelas. Já para Kern (1994) o modelo relacional é definido como aquele no qual os dados são percebidos pelos usuários como tabelas e as operações aplicáveis ao sistema geram tabelas a partir das primeiras. Um SGBD Relacional possui três elementos principais:

a) dados, que são representados por tabelas;
b) operadores para manipulação de dados;
c) regras de integridade das tabelas.



Fonte: adaptado de Baehr Jr. (1999)
Figura 2.4 – Um exemplo de Banco de Dados Relacional

As tabelas ou relações dos bancos de dados relacionais (BDR) são formadas por linhas ou tuplas, que indicam cada “registro” da tabela, e colunas ou atributos, que identificam os “campos” da tabela. Cada atributo possui um domínio associado a ele, ou seja, um conjunto de valores que o mesmo pode assumir (Baehr Jr., 1999).

Os banco de dados relacionais são declarativos. Isso significa que com eles as aplicações preocupam-se no que precisam do banco de dados, desconsiderando fatores como de que forma será acessado o servidor de dados (Baehr Jr., 1999).

O esquema de dados mais utilizado para especificar BDR é o ER. Ele constitui-se basicamente de :

a) retângulos, que representam as entidades. Uma entidade corresponde a uma tabela durante a especificação lógica. Esta entidade é formada de atributos que serão os campos da tabela à qual a entidade corresponde;
b) linhas/losângulos, que correspondem aos relacionamentos entre as entidades.

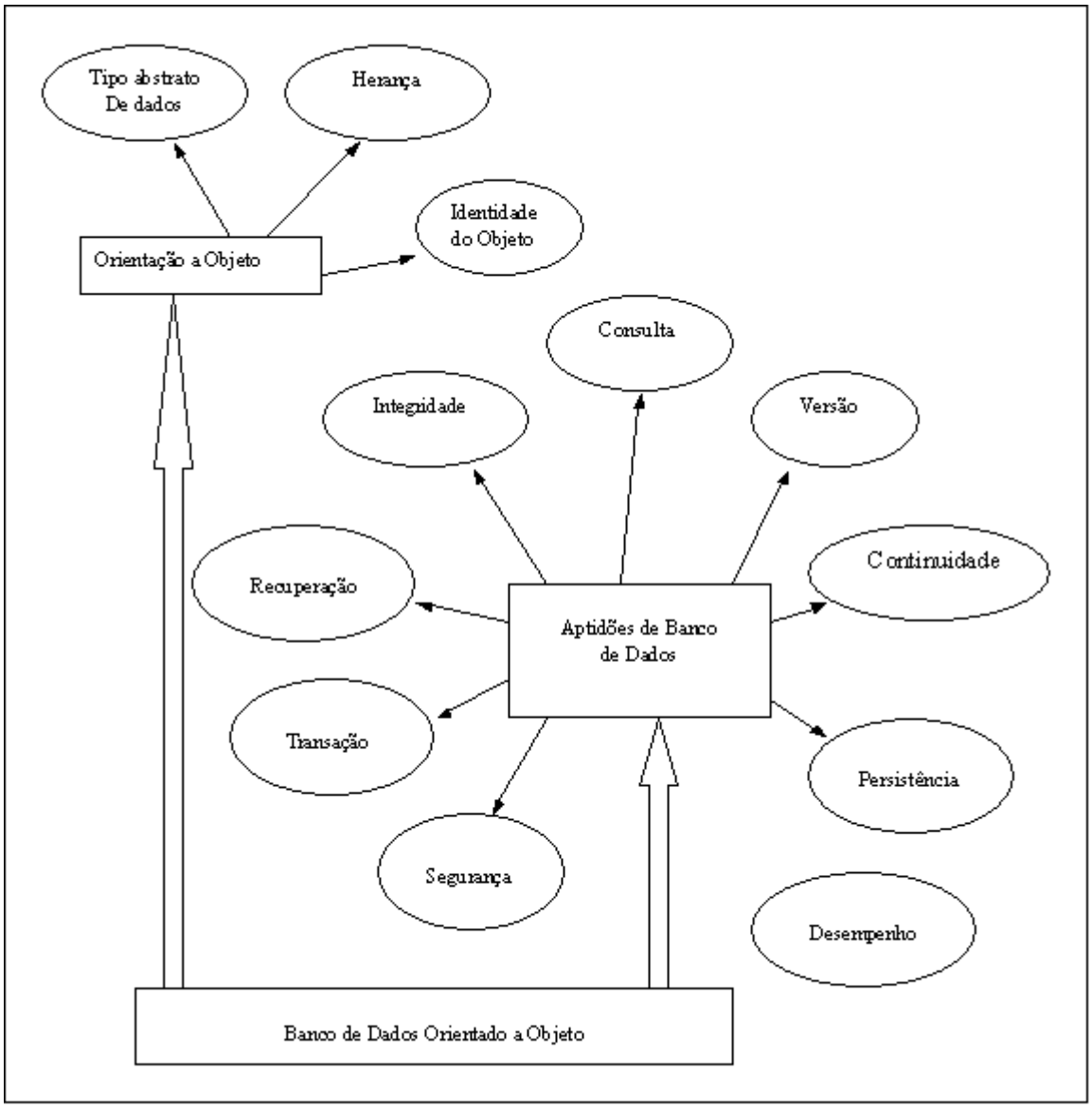
2.1.4 O modelo orientado a objeto

O modelo Orientado a Objeto tem por objetivo adicionar aos SGBD os conceitos de orientação a objetos, mais especificamente, os conceitos de tipagem de dados abstratos, herança e identidade de objeto, Khoshafian (1994). Como o enfoque principal deste trabalho é o estudo deste modelo de dados, o capítulo 3 é totalmente dedicado a ele.

Trabalhos de conclusão de curso sobre bancos de dados podem ser vistos em Biazin (1999), Baehr Jr. (1999), Silva (2001) e Uessler (1999).

3 BANCO DE DADOS ORIENTADO A OBJETOS

Os banco de dados orientados a objetos são fruto da união de duas tecnologias: Sistemas Gerenciadores de Banco de Dados e Orientação a Objetos, Kern (1994). Ele é totalmente baseado no paradigma da programação orientada a objeto (manter dados e programas armazenados no mesmo módulo) unido aos objetivos básicos dos SGBD (figura 3.1).



Fonte: Khoshafian (1994)
Figura 3.1 – Banco de Dados Orientados a Objeto

Segundo Rao (1994), um BDOO é definido com um sistema que pode ser usado para armazenar e recuperar objetos e que dispõem de facilidades para manipular esses objetos. Um BDOO torna possível armazenar um objeto na memória secundária da mesma forma que ele

existe na memória principal, sem ter que proceder mudanças de representação ou estrutura no mesmo.

Para Grein (1998), um BDOO pode ser definido como um banco de dados capaz de armazenar além de dados, como nos tradicionais sistemas de arquivos e estruturas dos bancos de dados tradicionais, outros diferentes tipos de dados que não podem ser convertidos em arquivos lineares ou bidimensionais como tabelas e sim num tipo especial de objeto. A característica marcante de um BDOO, segundo Grein (1998), é a capacidade que o banco apresenta de modelar não só os dados de estruturas complexas, mas também seu comportamento.

Os BDOO possuem importante papel dentre os SGBD por, pelos menos, três motivos:

a) são os modelos de dados mais adequados para tratamento de objetos complexos;
b) possuem maior naturalidade conceitual, através da definição dos objetos;
c) estão em concordância com fortes tendências das linguagens de programação e da Engenharia de Software.

Existem, pelo menos, três abordagens com relação à construção de Sistemas Gerenciadores de Banco de Dados Orientados a Objeto (SGBDOO). A principal incorpora aos SGBD convencionais existentes uma camada responsável por processar as solicitações de Orientação a Objeto (OO) e armazenar os métodos. A segunda defende a inclusão de características de OO nos Sistemas Gerenciadores de Banco de Dados Relacional (SGBDR) na linguagem *Structured Query Language* (SQL). Outra corrente defende a idéia de que é preciso criar uma tecnologia exclusivamente voltada para os BDOO e totalmente desvinculada dos produtos relacionais existentes.

3.1 MOTIVOS DO SURGIMENTO DO BANCO DE DADOS ORIENTADO A OBJETO

Segundo Martin (1995), os BDOO surgiram, em primeiro momento, da necessidade de se sustentar à programação orientada a objetos. As pesquisas sobre BDOO foram estimadas com o surgimento das chamadas aplicações da próxima geração (*next generation application*).

a) projeto auxiliado por computador (CAD – Computer-Aided Design): Um banco de dados CAD armazena informações sobre determinado projeto de engenharia, seus componentes bem como suas antigas versões. Uma complexa rede de objetos é formada em torno de uma estrutura que de forma nenhuma possui tamanhos fixos.

É praticamente impossível retornar todos os componentes de um único projeto ou de uma engenharia do mesmo através de uma única instrução de consulta com uma *Query Language*;

b) engenharia de software auxiliada por computador (CASE – Computer-Aided Software Enineering): Como todo to tipo de engenharia, a engenharia de software também possui fases definidas durante a implementação de um projeto: especificação de requisitos, análise de requisitos, especificação de projeto, implementação e testes de validação. Um banco de dados CASE é utilizado para armazenar todos os dados requeridos durante essas fases. Esses dados incluem módulos do sistema, interligações entre esses módulos, definições, histórico, etc. Todos esses componentes são modelados como objetos complexos e acessados através de identificadores próprios;

c) manufatura auxiliada por computados (CAM – Computer-Aided Manufacture): Um software CAM refere-se a um programa de computador responsável pelo gerenciamento de um processo produtivo. Esse gerenciamento pode ser o de uma simples máquina até uma complexa linha de produção. Os componentes envolvidos em tal processo sofrem constantes mudanças que devem ser processadas e comunicadas ao sistema. A função do banco de dados nesse processo é modelar os componentes envolvidos na forma de objetos, avaliando suas mudanças de estado bem como armazenando dados históricos sobre os mesmos;

d) banco de dados multimídia: Um banco de dados multimídia armazena dados sobre figuras, áudio, vídeo, etc. O problema no armazenamento de tais dados consiste em seu tamanho variável e indefinido.

Os SGBD “convencionais” (Rede, Hierárquico e Relacional) não estão preparados para as necessidades dessas novas aplicações. Como tais SGBD evoluíram a partir dos sistemas de gerenciamento de arquivos, eles esbarram em uma série de problemas quando da necessidade de gerenciar aplicações da próxima geração (Baehr Jr., 1999):

a) falta de poder de modelagem de dados: as aplicações convencionais possuem a informação de uma forma bem estruturada, em estruturas limitadas e transações que não duram muito tempo. Em contrapartida, as aplicações da próxima geração possuem tipos de dados mais complexos, como matrizes (arrays), objetos, definições de classes, etc. Dessa forma, o poder disponibilizado pelas linguagens de

programação orientadas a objetos não é suportado pelos bancos de dados convencionais;

b) falta de mecanismos para suportar transações longas: Nos atuais SGBD dados em alteração permanecem bloqueados durante o tempo de manutenção. Isso torna-se imperceptível para os usuários visto que tais transações são rápidas e o tempo de resposta é curto. Já em BDOO as transações podem ser bem mais longas. Existe a possibilidade de haver a interação humana com a transação, efetuando operações do tipo “O que acontece se...”. Transações da próxima geração podem passar dias bloqueados todo esse tempo. Por isso, faz-se necessário um novo modelo de gerenciamento de transações;

c) desigualdade de impedância (*impedance mismatch*): Conceito relacionado a maneira diferente com que o banco de dados e linguagens de programação podem suportar modelos de dados e paradigmas de objetos suportados, a impedância desigual requer um grande esforço para mapear os modelos de objetos suportados pelas linguagens de programação em modelos de dados nos banco de dados;

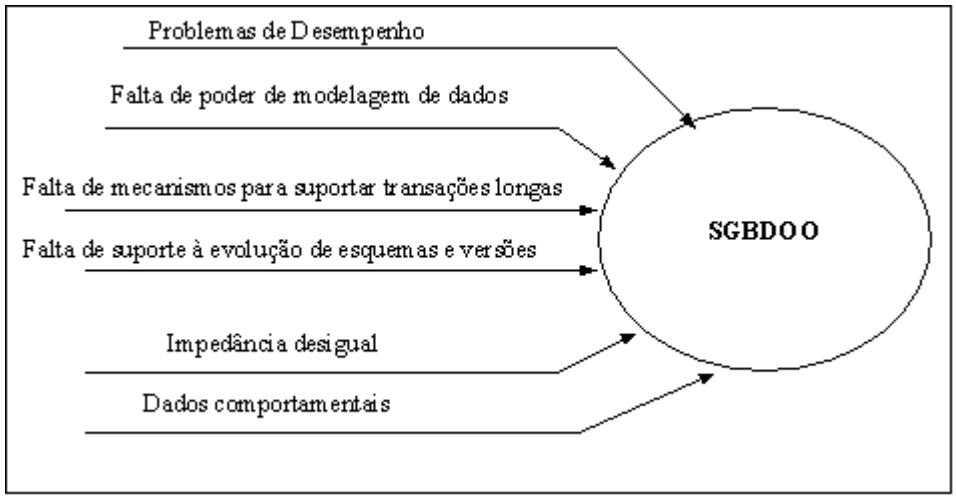
d) problemas de desempenho: Devido a complexidade dos modelos de objetos as transações que envolvem os mesmos diferem das atuais transações suportadas pelos SGBD convencionais. Enquanto uma transação normal em um SGBDR limita-se a buscas e atualizações em relações de tuplas, uma simples transação de consulta em um SGBDOO pode iniciar um processo em cadeia formando uma rede complexa de acesso a várias instâncias de objetos de uma mesma classe quanto acesso a objetos embutidos ou referenciados. A necessidade de disponibilizar tais transações em SGBD convencionais faz com que os mesmos transformem-se num emaranhado de índices para acesso associativo (*Joins*) através de instruções SQL;

e) dados comportamentais: Diferentes objetos podem responder de diferentes formas ao mesmo comando (Polimorfismo). Esse tipo de informação comportamental pode ser representada pelo armazenamento do código executável nos objetos dos bancos de dados. Nos bancos de dados orientados a objeto essa capacidade é representada pelos métodos;

f) falta de suporte à evolução de esquemas e versões: Em SGBD convencionais enxerga-se que o banco de dados possui um único estado semântico: o atual. Qualquer evolução ou alteração no esquema é uma árdua tarefa que deve ser desenvolvida pelo DBA. Nas aplicações da próxima geração, alterações/evoluções

do esquema são parte inerente da semântica da aplicação e não podem ser operações complexas.

Dessa forma, várias necessidades não satisfeitas pelos SGBD convencionais, sintetizadas na figura 3.2, convergiram no surgimento de uma nova geração de SGBD.



Fonte: Baehr Jr. (1999)
Figura 3.2 – Motivos do Surgimento dos SGBDOO

3.2 CONCEITOS DE ORIENTAÇÃO A OBJETOS PARA SGBDOO

Nos SGBDOO a noção de objetos é utilizada em nível lógico e possui características não encontradas nas linguagens de programação orientadas a objetos, como operadores de manipulação de estruturas, gerenciamento de armazenamento, tratamento de integridade e persistência de objetos. Dessa forma, a implementação dos conceitos sobre orientação a objetos, como “classe” e “herança”, sofrem algumas adaptações quando aplicados a banco de dados. Deve-se levar em consideração não somente o conceito que se está desenvolvendo, mas também os impactos que tal conceito sofrerá para se adaptar às realidades de um SGBD, (Baehr Jr. 1999).

3.2.1 Objetos para banco de dados

As abstrações de representações e das operações de um objeto são suportadas no modelo de dados orientado a objetos, Gualberto (2003). Para tal, são incorporadas às características de objetos das linguagens de programação orientadas a objeto as noções de estruturas de dados e comportamento. O estado interno do objeto é descrito pelo seu conjunto de atributos. Cada “ocorrência” do objeto no banco de dados é denominada de instância do

objeto. A parte estrutural de um objeto (em banco de dados) é muito similar ao conceito de entidade no modelo Entidade - Relacionamento.

3.2.1.1 Identidade de objetos

Segundo Korth (1997), em um SGBDOO os objetos geralmente correspondem a entidades da empresa que está sendo modelada pelo banco. Estas entidades mantêm sua identidade mesmo quando algumas de suas propriedades mudam com o passar do tempo. Diferente das tuplas de um BDR, que são identificadas unicamente pelos valores que contém, a Impossibilidade de garantir identificação de objetos através de seus atributos ou de seu comportamento motivou a definição de identificadores únicos, que persistirão independentes de alterações em atributos ou métodos que o objeto venha a sofrer.

Esta mesma identidade também é utilizada para referenciar o objeto como atributo junto aos demais objetos. Dessa forma, a identidade do objeto vem a eliminar anomalias de atualização e de integridade referencial, uma vez que a atualização de um objeto será automaticamente refletida nos objetos que o referenciam e que o identificador de um objeto não tem seu valor alterado, Gualberto (2003).

Segundo Maciaszek (1997) a identidade do objeto (OID do inglês Object Identify) é um identificador global lógico sem igual, independente do estado ou endereço do objeto. Em SGBDOO, o conceito de OID é mais “forte” que em LPOO. As LPOO foram concebidas para manipular objetos transientes (objetos que deixam de existir junto com o final do processo que os criou), diferente dos SGBD, que foram criados para manipular objetos persistentes (objetos que sobrevivem a execução do processo que os criou). Dessa forma, um OID em SGBDOO não é:

a) um endereço (como um nome de variável ou referência de memória de uma linguagem de programação) porque não é externo ao objeto;
b) uma chave (como chave primária de um BDR) porque não é um valor de dados mutável.

Segundo Baehr Jr. (1999) a identidade do objeto acompanha-o desde o momento de sua criação até o momento de sua exclusão física da base de dados. O objeto mantém uma única identidade de quantas vezes ele for instanciado em memória ou de quais são os valores de suas propriedades. Em SGBDOO, o OID é dito Persistente, ou seja, a identidade do objeto

persiste não só entre execuções de programas, mas também durante reorganizações estruturais de dados.

3.2.1.2 Objetos complexos

Segundo Gualberto (2003), objetos complexos são formados por construtores (conjuntos, listas, tuplas, registros, coleções, arrays) aplicados a objetos simples (inteiros, booleanos, strings). Nos modelos orientados a objetos, os construtores são em geral ortogonais, isto é, qualquer construtor pode ser aplicado a qualquer objeto. No modelo relacional este não é o caso, visto que só é possível aplicar o construtor de conjuntos às tuplas e o construtor de registro a valores atômicos.

Existem, basicamente, dois tipos de objetos complexos em BDOO, (Baehr Jr. 1999):

a) objetos embutidos: são “objetos filhos”, na forma de atributos, que só podem ser acessados pelo seu “objeto-pai”. São fruto de estruturas de agregação ou “todo-parte”. Objetos Embutidos não possuem OID próprio e são, em geral, armazenados na mesma estrutura física de seu “objeto-pai”. Um típico exemplo de objetos embutidos é o caso dos relacionamentos dependentes, como os “itens de um pedido de venda”. Nesse exemplo, os “itens” são objetos embutidos dos “pedidos de venda”.
b) objetos referenciados: os objetos referenciados são todos os objetos originários das regras de integridade referencial. Qualquer relacionamento, como o caso de uma “Cidade” com seu respectivo “Estado” corresponde a criação de um objeto composto do tipo referenciado. Os objetos referenciados possuem OID próprio e podem ser acessados diretamente ou através de seus objetos relacionados.

A manutenção de objetos complexos, independente de sua composição, requer a definição de operadores de gerenciamento apropriados à manipulação total de um objeto ou transitiva de alguns de seus componentes. Exemplos típicos dessas operações são: a atualização ou remoção de um objeto, as restrições de acesso e o controle de concorrência.

3.2.2 Hierarquia de classes e herança para banco de dados

Outra importante capacidade dos SGBDOO é o gerenciamento do conceito de herança dentro de uma hierarquia de classes armazenáveis. Da mesma forma que nas LPOO, nos

BDOO pode-se criar novas classes em função de classes já existentes. Todas as características com relação à classe e herança devem estar disponíveis em um SGBDOO.

Os SGBDOO armazenam as declarações das classes, confeccionando-as como parte do esquema do banco de dados, Objectstore (1999). Uma hierarquia de classes oferece muito mais flexibilidade para se mudar a estrutura de um banco de dados (incluindo novos atributos ou métodos nos objetos) bem como possibilita a evolução do esquema do banco de dados através da adição de novas classes na hierarquia.

Segundo Baehr Jr. (1999) as “Classes de Coleção” ou “Classes de Sistema” são exemplo típico das características de herança dentro de um SGBDOO. A maioria dos SGBDOO possui uma coleção própria de classes de objetos. Estas classes são organizadas numa hierarquia. Se necessário, uma determinada aplicação pode desenvolver sua própria classe e herdar atributos ou métodos de alguma classe do sistema. Em resumo, as classes de sistema dos SGBDOO implementam os métodos de armazenamento, manipulação, controle de concorrência, entre outros.

3.3 CONCEITOS DE BANCO DE DADOS PARA SGBDOO

Da mesma forma que os conceitos de OO sofrem adaptações para serem aplicados a SGBDOO, os conceitos de banco de dados são adaptados às necessidades e realidades dos BDOO. As transações de BDOO podem demorar dias para se concretizarem. Os objetos devem manter todos os seus níveis disponíveis para acessos concorrentes. As consultas de objetos envolvem estruturas complexas e esbarram no problema da comunicação via mensagens. Esses processos são implementados de maneira a garantir a integridade e concorrência dos objetos.

3.3.1 Transações

De forma simplificada, transação é um programa ou seqüência de ações que lê ou grava objetos persistentes e mantém coerente o banco de dados, Khoshafian (1994). Para tal, uma transação deve atender as propriedades ACID (atomicidade, coerência, isolamento e durabilidade). Atomicidade significa que o programa ou a seqüência de ações é totalmente executado ou nada é executado.Coerência implica em partir de um estado coerente do banco de dados (onde todas as restrições de integridade são satisfeitas) e retornar para um estado

coerente ao final da transação. Isolamento significa que as transações não devem ler dados intermediários de outras transações, mantendo-se isoladas das mesmas. Durabilidade garante que uma vez a transação executada, estão garantidos que seus efeitos e/ou atualizações serão suportadas.

Se uma transação é abortada, nenhuma mudança no estado dos objetos é remetida ao banco de dados. Todos os objetos permanecem no mesmo estado em que se encontravam quando do começo da transação. Existe pelo menos um problema no que diz respeito a execução de uma transação em SGBDOO: a transação pode não envolver somente objetos persistentes e a execução de um *Rollback* pelo SGBDOO não implica na restauração do estado inicial dos objetos não-persistentes. Esta é uma situação onde o programador deve saber quais não são os objetos persistentes durante sua transação, pois eles podem se comportar de maneira diferente.

As transações dos SGBD convencionais são curtas e atualizam ou referenciam somente alguns registros. Nos SGBDOO as transações possuem um papel mais complexo: podem ser demoradas ou necessitar de trabalhos em grupo para serem executadas. Segundo Maciaszek (1997), duas grandes diferenças arquitetônicas entre BDR e BDOO é que os BDOO suportam o acesso multi-usuário aos dados (acesso cooperativo) e freqüentemente apresentam transações de longa duração.

O modelo de transações aninhadas, introduzido por Moss em 1981, Khoshafian (1994), é utilizado para resolver os problemas das transações demoradas. Ele corresponde, basicamente, em decompor a transação principal numa série de subtransações que devem ser executadas com êxito para obter-se o resultado da transação principal. Se alguma das subtransações falhar, fica a cargo da transação pai tentar novamente a execução da subtransação ou “abortar” a operação. Quando uma transação de alto nível é desfeita todas as suas subtransações também o são, independente do fato destas terem sido executadas com sucesso.

O modelo de transações em cooperação é utilizado para resolver o problema das transações que necessitam de trabalho em conjunto para serem concluídas. As transações em cooperação permitem a visualização dos resultados imediatos uma das outras.

3.3.2 Concorrência

Proporcionar simultaneamente o acesso aos dados para diferentes usuários aplica o famoso conceito de controle de concorrência. O mais notável algoritmo de controle de concorrência existente para SGBD é o bloqueio. Nos SGBDOO, o bloqueio deve ser associado aos vários níveis das estruturas existentes, incluindo classes, instâncias e objetos complexos.

Existem basicamente dois tipos de bloqueios: o bloqueio de leitura ou bloqueio compartilhado, que permite a várias transações realizarem leituras concorrentes no mesmo objeto; e o bloqueio de gravação ou bloqueio exclusivo, que reserva o acesso (operações de leitura e gravação) de um objeto à transação que solicitou esse bloqueio. Quando uma transação mantém um bloqueio exclusivo sobre um objeto, nenhum outro bloqueio pode ser disponibilizado.

Segundo Khoshafian (1994), existem dois aspectos relevantes no que diz respeito ao controle de concorrência em SGBDOO:

a) bloqueios de hierarquia de classe: as classes em SGBDOO são organizadas em uma hierarquia de herança. Dessa forma, o bloqueio de hierarquia de classe aplica um “bloqueio implícito” a todas as subclasses de uma determinada superclasse bloqueada. As subclasses incluem os descendentes diretos da superclasse e os descendentes de sua subclasse;
b) bloqueios de objeto complexo: o problema dos objetos complexos consiste no fato de que um objeto complexo pode ter alguns de seus sub-objetos bloqueados, ao mesmo tempo, na mesma forma que ele foi bloqueado. Isso implicaria em restrições de atualização e possíveis problemas de acesso. Para resolver tal problema foram desenvolvidos vários esquemas de bloqueio para objetos complexos, dentre os quais salienta-se o bloqueio de componente de intenção. Esse bloqueio funciona como uma espécie de bloqueio de instância, ou seja, quando uma classe recebe um bloqueio de intenção ela não tem suas subclasses bloqueadas implicitamente. Somente as instâncias dependentes de um objeto-pai (complexo) são bloqueadas implicitamente no mesmo modo que o objeto-pai foi bloqueado. Outras instâncias da classes componente, que não fazem parte do objeto-pai, ficam livres para serem acessadas por outras transações. Uma vez bloqueado um nó de classe na estrutura

de hierarquia de classe no modo componente de intenção, a sub-árvore parcial criada no nó bloqueado é implicitamente bloqueado no mesmo modo do objeto-pai.

3.3.3 Recuperação

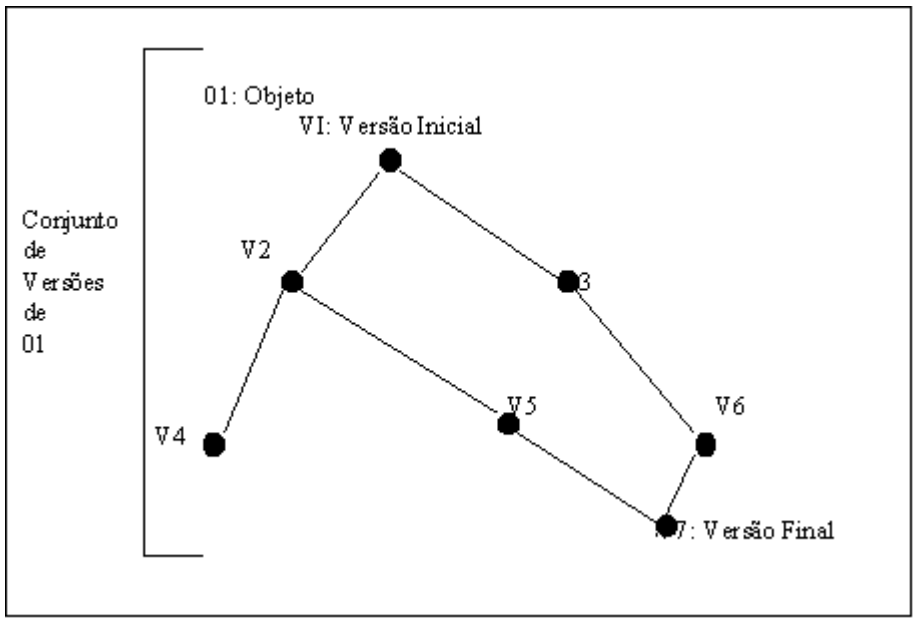
Conforme Khoshafian (1994) todas as transações são atômicas: se realizam com sucesso ou não se realizam. Isso significa que o SGBD deve garantir que transações não realizadas com sucesso ou atualização parciais dos dados não impliquem em atualizações no banco de dados persistente.

Uma das estruturas de dados mais utilizadas para o gerenciamento de recuperação é o log. O log consiste em um mecanismo que armazena imagens anteriores e posteriores dos objetos. A imagem anterior consiste no estado do objeto antes da atualização da transação e a imagem posterior é o estado do objeto após a atualização. O log é o mecanismo de recuperação mais utilizado pelos BDOO. Alguns BDOO utilizam outros mecanismos como a duplicação ou espelhamento dos dados.

3.3.4 Versionamento

Segundo Khoshafian (1994), o versinamento consiste de ferramentas e construções do SGBDOO que automatizam a construção e a organização de versões de um mesmo objeto, permitindo aos usuários acesso tanto ao estado atual quanto a estados anteriores dos objetos. O acesso a estados anteriores de determinados objetos é parte importante em diversas aplicações. Aplicações CAM, CAD e CASE necessitam, muitas vezes, manter versões de suas “engenharias” armazenadas para futuras análises e “reciclagens”. Aplicações financeiras e contábeis também necessitam armazenar antigas versões.

Depois que um objeto é versionado, uma espécie de raiz que aponta para o conjunto de todas as versões do objeto é criada. Durante o versinamento um objeto pode sofrer desde alterações de estados até modificações estruturais. A identidade do objeto é a principal propriedade comum a todas as versões do mesmo objeto.



Fonte: Adptado de Khoshafian (1994)
Figura 3.3 – Versionamento

O esquema de versionamento mais comum é a sucessão linear de gerações. Nele, as versões dos objetos são criadas seqüencialmente, formando um conjunto de versões lineares. Porém, existem casos em que são criadas em paralelo versões alternativas de um mesmo objeto. Esse esquema de versionamento é denominado de sucessão alternativa de gerações. Nesse tipo de esquema a versão final do objeto pode ser obtida fundindo-se as idéias das várias versões paralelas do mesmo.

Observando a figura 3.4, nota-se que a versão V7 do objeto O1 é uma versão fruto da sucessão alternativa das versões V5 e V6. Já, o caminho de V1 a V3 a V6 a V7 é considerado um versionamento linear de gerações entre V1 e V7.

3.3.5 Restrições de integridade

Segundo Baehr Jr. (1999), SGBD geralmente provêem mecanismos para garantir a coerência dos dados, expressados por predicados, denominados de restrições de integridade. Como as LPOO não oferecem tais mecanismos, os SGBDOO têm que incorporar tais conceitos às suas características.

As maiorias das restrições de integridade existentes nos banco de dados convencionais são também aplicáveis aos BDOO. O detalhe é que, devido às construções e ao poder de

modelagem do modelo de dados orientado a objetos, outros tipos de restrições e especificações de integridade são introduzidos aos BDOO.

3.3.5.1 Restrições de integridade de chave

Nos BDR é comum a especificação de um ou mais chaves para as tabelas, como o objetivo de identificar univocamente os registros de uma tabela em nível físico e lógico (chaves primárias) bem como otimizar os processos de consulta (chaves secundárias).

Nos BDOO a função de identificar univocamente em nível físico as instâncias de uma classe é função de OID. Porém, para identificação exclusiva em nível lógico das instâncias de uma classe os BDOO permitem a definição de uma ou mais atributos desta classe como sendo chave para a mesma. Nos BDOO as chaves também otimizam os processos de consulta.

Dessa forma, as chaves no BDOO possuem duas funções, Khoshafian (1994):

a) Otimizar os processos de consulta;

b) Servir de restrição para ambigüidade lógica de instâncias.

3.3.5.2 Restrições de integridade referencial

Os objetos de um banco de dados não vivem isolados. Normalmente, eles estão relacionados com outros objetos no banco de dados, Objectstore (1999). O relacionamento entre objetos é um componente essencial no paradigma do modelo OO. Através dele pode-se criar objetos inter-relacionados, (Baehr Jr. 1999).

Da existência das relações vem a necessidade da integridade referencial. A integridade referencial tem por objetivo assegurar que objetos não contenham relacionamentos com objetos que não existam mais no banco de dados. Para Khoshafian (1994), as especificações de restrição de integridade referencial garantem que não haja referências “pendentes” para objetos de uma classe.

Nos modelos baseados em valor (como o modelo relacional, por exemplo) o mecanismo mais utilizado para referenciar objetos entre si é o da chave estrangeira, que relacionam os objetos pelos valores de seus atributos. No conceito de chave estrangeira, o valor de um atributo num objeto é na verdade um valor de chave que se refere a outro objeto

de um grupo. Nos SGBDOO que suportam o conceito de OID do objeto não há necessidade de restrições de integridade do tipo “chave estrangeira”, visto que o OID permite referenciar objetos diretamente, Khoshafian (1994).

3.3.5.3 Restrição existencial

A restrição existencial tem por objetivo garantir que se um objeto é compartilhado referencialmente (fato só possível se o sistema suportar o conceito de OID) então esse objeto possui um domínio “ativo” (um grupo específico de objetos que no momento existem em função de determinada condição) no qual ele deve existir.

Por exemplo, se um escritório de contabilidade possui um sistema de folha de pagamento multi-empresa e no cadastro de empresas do sistema existe um campo denominado “Telefone”, consiste numa restrição existencial garantir que todo o telefone informado no campo “Telefone Comercial” do cadastro de funcionários é um telefone existente no domínio ativo do cadastro de empresas (telefones cadastrados nas empresas).

3.3.5.4 Restrição NOT NULL

Quando uma restrição *NOT NULL* é definida para um atributo, a faixa de valores que esse atributo pode receber corresponde à faixa específica para o tipo definido para o atributo. Valores *NOT NULL* (ou seja: atributo “ausente”) não são aceitos. Para um SGBDOO suportar a restrição *NOT NULL*, ele obrigatoriamente tem que suportar valores *NULL*.

3.3.5.5 Restrições de pré-condições e pós-condições de métodos

Conforme Khoshafian (1994), as restrições de precondições permitem a introdução de certas restrições nas variáveis de instância que devem ser satisfeitas antes que determinado método seja executado. Já as restrições depôs-condições permitem a definição de outras restrições que devem ser cumpridas após um método ser executado.

3.3.5.6 Restrições de cobertura

A restrição de cobertura garante que determinada superclasse não pode possuir instâncias que não sejam elementos de uma de suas subclasses. Uma maneira de se obter tal restrição consiste em definir a superclasse em questão como sendo abstrata. Dessa forma

garante-se que toda e qualquer “instância” dessa superclasse corresponde na verdade a instância de uma de suas subclasses.

Entretanto, o suporte a restrições de cobertura por meio de classes abstratas nem sempre é desejável, visto que as restrições de cobertura diferem das classes abstratas proque em alguns casos é desejável criar instâncias de superclasse. Em outras palavras, algumas vezes é desejável tornar a superclasse uma classe não abstrata (uma classe que pode ser instanciada) e mesmo assim apresentar uma restrição de cobertura.

3.3.6 Gerência de persistência

Segundo Rao (1994), a persistência pode ser definida como a habilidade de objetos sobreviverem à execução dos módulos (programas ou métodos) nos quais esses objetos são definidos e criados.

Um SGBDOO possui mecanismos para armazenamento e recuperação de objetos persistentes. Segundo Objectstore (1999), o aspecto mais importante deste mecanismo é que as aplicações não precisam mais se recuperar com os detalhes de como “mapear” um objeto da memória para o disco. Não há necessidade de se efetuarem conversões entre o formato dos dados na memória e no disco. Esta conversão é de responsabilidade do SGBDOO.

3.3.7 Consultas

Segundo Rao (1994) as consultas em SGBDOO devem possibilitar tanto a consulta de objetos simples quanto de objetos complexos. Também deve permitir a consulta a uma coleção de objetos com a intenção de recuperar determinada instância. Quando objetos complexos são consultados, o SGBDOO deve permitir a “navegação” de um objeto ao outro através dos “*links*” que representam as relações entre os objetos.

O grande problema das consultas em SGBDOO é que as LPOO requerem que toda a interface com os objetos seja realizada através de mensagens, Kern (1997). Dessa forma, em uma LPOO para se obter, por exemplo, toda a ocorrência de determinado atributo nas instâncias dos objetos de uma classe é necessário enviar uma mensagem a cada uma das instâncias. Nas linguagens de objetos para banco de dados existe o modelo de mensagens de objeto para conjuntos. Através dele é possível efetuar consultas que envolvam junções de conjuntos de objetos.

Em verdade, as linguagens para consulta de objetos são eficientes Celko (1997). Elas devem incluir tanto o modelo de passagem de mensagens de um objeto por vez, quanto o modelo de mensagens por conjunto.

4 BANCO DE DADOS NA WEB

Com o crescimento da Web, surge uma nova geração de aplicações específicas para os negócios via Internet. Estas aplicações consistem essencialmente em transações, baseadas em dados e informações interativas, ligadas em bancos de dados corporativos. As organizações interessadas em explorar estes novos empreendimentos têm a necessidade de aproveitar suas estruturas cliente/servidor, integrando e interagindo com a Web. As aplicações devem combinar potencialidade, heterogeneidade, compatibilidade, confiabilidade e segurança dos ambientes cliente/servidor e a facilidade de uso do browser. Com isto torna-se fundamental um estudo aprofundado de metodologias e de ferramentas para o desenvolvimento de soluções, visando um atendimento das necessidades crescentes do mercado, Corbellini (1998).

Para Corbellini (1998), a demanda por novas pesquisas pela comunidade de Banco de Dados deve ser direcionada para o desenvolvimento de tecnologia que ajude a solucionar problemas surgidos nos últimos anos devido à explosão do crescimento da comunicação, incluindo a *Internet* e, em particular, a Web.

4.1 INTEGRANDO WEB E BANCO DE DADOS

Existem diversas aplicações de Banco de Dados que podem ser transportadas para o ambiente Web e várias aplicações Web que podem usar Bancos de Dados como mecanismos mais eficientes para o armazenamento de informações. Neste sentido, a Web está passando rapidamente da condição de troca irrestrita de documentos, para a condição de uma plataforma de desenvolvimento de inúmeras aplicações baseadas em Banco de Dados. Além disso, vêm crescendo as soluções *Intranet* integradas a Banco de Dados: uma rede corporativa de computadores baseada nos padrões de comunicação da *Internet* pública e da Web, cujos dados são armazenados e gerenciados por SGBDs.

De modo geral pode-se enumerar as seguintes vantagens da integração Web e Banco de Dados, Lima (1997):

a) Os SGBDs são providos de mecanismos muito mais eficientes como estruturas de índices automáticos, execução otimizada de consultas, autorização de acesso, suporte a transações em ambientes multi-usuário (como consistência e controle de concorrência), mecanismos de recuperação de dados quando há falhas, facilidade para modelar e organizar tipos de dados complexos e relacionamento entre eles

(segundo o modelo relacional, por exemplo), além de usarem linguagens específicas de consulta que independem da aplicação (como por exemplo SQL - *Structured Query Language*);

b) Existe atualmente uma grande massa de dados residindo e sendo gerenciada por SGBDs. A Web veio abrir a possibilidade de dispor estas informações a um número ilimitado de usuários em redes locais ou remotas. Além disso, a Web vem possibilitando o desenvolvimento de aplicações baseadas em SGBDs capazes de conectar membros de uma organização sob uma rede heterogênea, local ou remotamente;

c) A Web está possibilitando o desenvolvimento e expansão de novas aplicações baseadas em SGBDs. São exemplos dessas aplicações compras eletrônicas via *Internet* e bibliotecas virtuais.

4.2 PROBLEMAS DE INTEGRAÇÃO WEB E BANCO DE DADOS

Segundo Lima (1997), apesar das inúmeras vantagens proporcionadas pela integração SGBD e Web o desenvolvimento de aplicações nesse ambiente ainda está repleto de problemas técnicos cujas soluções não são triviais, como por exemplo, os aspectos transacionais relativos às aplicações Web com Banco de Dados, aspectos de segurança neste novo ambiente, restrições de integridade, desempenho, desenvolvimento, portabilidade e linguagens de programação. Também são apontados problemas em aberto e, quando possível, soluções alternativas existentes.

Vale ressaltar que atualmente alguns dos problemas citados acima já foram solucionados.

4.2.1 Problema de sistemas legados

O problema de sistemas legados (*legacy systems*) é inerente a qualquer tecnologia nova como a Web. Sistemas legados são sistemas que resistem a modificação e evolução. Os SGBDs representam, no contexto atual do ambiente Web, um importante subconjunto de sistemas legados, Shklar (1995).

Até o momento, todos os SGBDs comerciais foram projetados e implementados sem levar em consideração as necessidades especiais do ambiente Web. Além disso, muito já foi

investido em SGBDs e em ferramentas de desenvolvimento. Portanto, a integração com o ambiente Web envolve a consideração dos mecanismos já desenvolvidos pelos fabricantes de SGBDs.

4.2.2 Problemas transacionais

Um dos grandes problemas da integração SGBD e Web é a natureza "sem estado" (*stateless*) das transações Web em decorrência do protocolo HTTP usado na comunicação cliente Web e servidor Web. Um servidor Web responde a uma solicitação do cliente Web retornando uma página HTML ou disparando uma aplicação externa via interface CGI, por exemplo. Uma vez que o pedido é atendido, a transação é completada e a conexão se encerra. O servidor Web não armazena nenhuma informação sobre a aplicação e/ou o usuário da aplicação. Esta sistemática é boa para distribuir documentos, pois alivia-se o servidor Web para atendimento dos demais pedidos dos clientes, sem ficar conectado ao cliente Web até que o usuário finalize todos os seus pedidos.

No entanto, surgem muitos problemas nos projetos de aplicação Web sobre Banco de Dados. De fato, é sabido que os atuais SGBDs foram projetados para atenderem pedidos de usuários que se conectam ao Banco de Dados em sessões *contínuas*, ou seja, o usuário fica conectado ao SGBD enquanto durar sua sessão, Lima (1997). A mudança para um ambiente baseado em transações *stateless* como o da Web, implica em vários questionamentos, como, por exemplo, uma *definição* formal de transação Web Banco de Dados, o *gerenciamento* da execução de transação neste ambiente de integração e a necessidade de se verificar se o *modelo* de transações usado nos atuais SGBDs é adequado ao ambiente Web Banco de Dados.

4.2.3 Segurança e acesso ao banco de dados

Outro problema importante quando se discute a integração Web e Banco de Dados é relativo à segurança nas transações entre o cliente Web, o servidor Web e o SGBD, Rahmel (1997). Três pontos se destacam: a segurança na comunicação entre cliente Web e servidor Web, a segurança no servidor Web e a segurança e acesso ao Banco de Dados, que será analisada nesta seção.

A segurança e acesso ao Banco de Dados diz respeito a uma das importantes funcionalidades dos atuais SGBDs. Pode-se ver a Web como um usuário do SGBD como

qualquer outro. Nestes termos, os mecanismos de segurança e acesso ao Banco de Dados poderiam funcionar como os já existentes. Em particular, o uso de *visões*, Elmasri (1994) proporcionado pelos SGBDs no ambiente de integração pode ser bastante útil. Além disso, alguns fatores devem ser levados em consideração com relação à segurança e acesso ao Banco de Dados, como os destacados a seguir, Lima (1997):

a) Gerenciador de acesso: a maioria dos SGBDs mantém permissões para acesso ao Banco de Dados (senhas de *login*) e aos objetos do Banco de Dados (*grants*). Caso o contexto da aplicação seja uma *Intranet* pressupõem-se que os usuários sejam limitados e assim o gerenciamento de senhas e *grants* pode ser o habitual. Mas, se o SGBD puder ser acessado por um número ilimitado de usuários (*Interne*), pode ser inviável controlar o acesso ao Banco de Dados e aos seus objetos individualmente, devido a perda de desempenho dos mecanismos convencionais. É bom lembrar que os mecanismos de acesso aos Banco de Dados tradicionais e seus objetos foram projetados para gerenciar um número limitado de usuários. Para diminuir o problema, pode-se adotar a estratégia de classificar os usuários segundo grupos com as mesmas características de acesso ao Banco de Dados e seus objetos, e dar acesso a eles segundo o grupo em que estão. Perde-se no entanto, as características de acesso individuais de cada usuário;

c) Acessos simultâneos: o número de acessos simultâneos ao Banco de Dados é geralmente limitado. Dependendo dos requisitos da aplicação, um usuário pode não conseguir completar sua transação devido a sobrecarga no gerenciador de acessos do SGBD. Alguns *gateways* de integração podem oferecer um *cache* de acessos ao Banco de Dados e os compartilhar entre os usuários, se o SGBD permitir. Assim, um mesmo usuário com características de acesso ao Banco de Dados iguais a de um outro usuário já conectado ao SGBD pode usar este mesmo canal de comunicação já aberto para realizar seus pedidos. Já outros *gateways* requerem conexões dedicadas para cada usuário;

d) Autorização: em alguns *gateways* de integração Web e Banco de Dados, a autorização para execução de uma aplicação Web que consulta ou atualiza o Banco de Dados é dada ao *software* usuário que ativou o *gateway* (geralmente o próprio servidor Web). Assim, qualquer cliente Web poderia, por exemplo, disparar a aplicação via servidor Web. Mecanismos de configuração do servidor Web e do SGBD devem estar presentes de forma a contornar este problema;

e) Recuperação: um esquema de recuperação em Banco de Dados tradicionais é ativado como conseqüência de diversos tipos de falhas, como por exemplo, erros lógicos, paradas de sistema e falhas de disco. A extensão destes mecanismos para o ambiente Web não é imediata e os *gateways* devem implementar mecanismos adicionais para tratar de falhas dos componentes da aplicação para que possa, por exemplo, ser comunicado ao usuário o resultado de uma transação solicitada. Hoje não existe nenhum mecanismo de recuperação de paradas e falhas de clientes e servidores Web. Só estão presentes os mecanismos tradicionais dos SGBDs, que podem ser insuficientes no contexto transacional Web.

4.2.4 Controle de restrições de integridade

As restrições de integridade presentes na grande maioria dos SGBDs têm por objetivo assegurar que as eventuais atualizações respeitem as regras estruturais e semânticas definidas para o Banco de Dados.

Existem basicamente dois tipos de restrições de integridade, Korth (1997): *restrições de modelo*, também denominadas de restrições de chave e *restrições de domínio*. As restrições de modelo são geralmente fáceis de serem testadas e dizem respeito à organização interna do Banco de Dados, não sendo afetadas pelo ambiente Web. Já as restrições de domínio são restrições sobre os valores que podem ser assumidos, por exemplo, por atributos de tabelas de SGBDs relacionais. Embora sejam fáceis de serem testadas pelo SGBD isoladamente, podem gerar problemas quando se usa o ambiente Web.

De fato, num típico ambiente cliente/servidor como é o da Web, é desejável, para efeitos de desempenho, que as restrições de domínio sejam satisfeitas no cliente, antes de se enviar os dados contidos, por exemplo, num formulário HTML, para o servidor Web e daí para serem validadas pelo SGBD. Isto minimiza o tráfego na rede, já que diminui a freqüência com que o SGBD recuse alguma operação que não satisfaça a restrição de domínio imposta e retorne essa mensagem ao cliente para que seja corrigida. O grande problema é que a Web não foi projetada inicialmente para lidar com consistência de dados, por mais simples que isso seja. Os atuais formulários HTML aceitam somente campos do tipo texto e mesmo uma simples verificação preliminar de preenchimento obrigatório de algum campo é impossível de ser implementada sem o uso de ferramentas proprietárias incorporadas no cliente Web.

Para lidar com o problema da restrição de domínio no cliente Web é necessário recorrer às linguagens de programação, na maioria das vezes proprietárias, como *Java*, *JavaScript* e *VBScript*, entendidas pelos clientes Web, o que pode diminuir a portabilidade da aplicação. Caso contrário, haverá perda de desempenho e maior complexidade no desenvolvimento da aplicação que deve tratar erros decorrentes do não preenchimento das condições decorridas da restrição de domínio geradas pelo desenvolvedor da aplicação no SGBD.

Mesmo quando a integridade de domínio pode ser mantida no lado cliente, existe ainda o problema de mantê-la de forma a evitar ambigüidades entre as restrições implementadas no cliente Web e aquelas definidas no SGBD. Cabe observar também que nem toda restrição de domínio imposta no Banco de Dados é passível de ser tratada no lado cliente Web. Por exemplo, restrições de domínio que são avaliadas pelo SGBD com base nos dados armazenados se enquadram nesta categoria. Neste caso não resta outra alternativa senão tratá-las no *gateway*.

4.2.5 Problemas de desempenho

O problema de desempenho relacionado ao ambiente Web refere-se ao tempo de resposta a uma requisição do usuário, o que geralmente depende do desempenho dos vários componentes da Web, Murta (1996). Três desses componentes afetam diretamente uma transação Web Banco de Dados: o desempenho do cliente Web, o tráfego na rede e o desempenho do servidor Web.

O desempenho no cliente Web refere-se à capacidade de se exibir os dados enviados pelo servidor Web, considerando-se os diversos tipos de mídias possíveis, o gerenciamento do *cache* local e a execução de códigos associados aos dados enviados, como por exemplo, *applets* Java ou códigos JavaScript.

O tráfego na rede refere-se ao tempo necessário para prover acesso remoto, escolha da melhor rota para conexão com o servidor Web e transmissão dos dados na rede. Este é um problema potencial do ambiente Web principalmente se a transação for executada em horários de intenso tráfego na rede.

E por fim, o servidor Web que, como analisado em Murta (1996), é um dos componentes determinantes no tempo de resposta de uma transação cliente, já que ele é afetado pela carga de pedidos, pela plataforma de *hardware* e pelo ambiente de *software* (sistema operacional, subsistema de gerência de arquivos e execução de programas CGI, entre outros).

Além destes fatores pode-se identificar outros três que afetam particularmente o desempenho de aplicações Web Banco de Dados. São eles: otimização de consultas, número de usuários e balanceamento de carga. A otimização de consultas será abordada no próximo capítulo por ser uma importante funcionalidade dos atuais SGBDs. Os outros dois são descritos brevemente abaixo:

a) Número de usuários: aplicações Web podem atingir um grande número de usuários simultâneos, o que pode degradar o desempenho do servidor Web, do *gateway* ou até mesmo do próprio SGBD. Isto requer arquiteturas com alto desempenho que sejam bastante escaláveis através do aumento de CPUs disponíveis e possibilidade de processamento paralelo, por exemplo;
b) Balanceamento de carga: é imprevisível a carga de processamento que pode ser encontrada em aplicações Web Banco de Dados e seu gerenciamento envolve um grande número de fatores, entre os quais a habilidade da aplicação em lidar com consultas complexas sobre Bancos de Dados volumosos. Os mecanismos mais sofisticados devem usar arquiteturas de processamento distribuído que podem distribuir pedidos de usuários entre as várias máquinas físicas disponíveis.

Além das considerações acima sobre o desempenho de aplicações Web com Banco de Dados, cabe destacar ainda, o uso de procedimentos armazenados (*stored procedures*) no SGBD. A maioria dos SGBDs relacionais proporcionam esta funcionalidade, que consiste num conjunto de comandos SQL pré-compilados. Eles são usados para otimizar o desempenho de operações e consultas mais comuns. Se uma interação da aplicação Web Banco de Dados for utilizada com muita freqüência, procedimentos armazenados no SGBD devem ser utilizados sistematicamente.

4.2.6 Desenvolvimento e portabilidade

No que diz respeito ao ambiente de desenvolvimento e portabilidade de aplicações Web baseadas em Banco de Dados os seguintes pontos merecem destaque, Lima (1997):

a) Aplicações Web Banco de Dados devem ser fáceis de serem estendidas a novas versões HTML. Adicionalmente, elas devem ser flexíveis caso ocorram mudanças, por exemplo, nas versões do protocolo HTTP ou da interface CGI;

b) É recomendável a existência de mecanismos eficientes para transferir variáveis de entrada do cliente Web (*e.g.*, dos formulários) para as consultas (*e.g.*, SQL) no servidor de Banco de Dados. Este é um ponto particularmente importante em virtude do fato de que a transferência de dados de um formulário HTML para o *gateway* via interface CGI, por exemplo, não ser trivial;

c) A estrutura para desenvolvimento de aplicações Web Banco de Dados deve ser flexível, pouco dependente do esquema do Banco de Dados, com modelagem voltada para o ambiente Web, sem grandes conhecimentos de interfaces, como por exemplo, CGI ou APIs de servidores Web ou de programação do Banco de Dados, padronizada (com uso da *metabase* do SGBD, se possível) e portável (sem ferramentas proprietárias do SGBD e da Web);

d) A falta de metodologias para desenvolvimento de aplicações é uma característica do ambiente Web atualmente, por ser uma tecnologia muito recente. Metodologia aqui é entendida como um procedimento formal adotado para converter as necessidades do usuário em código da aplicação, o que envolve em geral modelagem, projeto, teste e documentação. Particularmente, a modelagem de uma aplicação Web Banco de Dados deve ser melhor estudada. Atributos no Banco de Dados podem incorporar características genéricas de formatação de entradas e saídas no formato HTML de forma a facilitar o desenvolvimento. Estas características poderiam, por exemplo, ser incorporadas na metabase do SGBD, podendo ser acessadas pelos programas da aplicação em tempo de execução das consultas ou atualizações. Hoje em dia alguns *gateways* Web Banco de Dados, Hadjefthymiades (1996), já proporcionam técnicas como SQL dinâmico (*dynamic* SQL) para tornar o ambiente de desenvolvimento mais flexível sob o ponto de vista de manutenção da aplicação;

e) A portabilidade no ambiente Web Banco de Dados não apresenta robustez. Mesmo padrões considerados abertos, uma característica prevista para o ambiente Web, ainda não se firmaram, gerando grandes problemas de portabilidade. Por exemplo, não existe atualmente nenhuma versão padrão HTML, estando disponíveis pelo menos duas versões: HTML 2.0 e HTML 3.0. Mudar de uma versão para outra

mais nova pode ser um problema durante a manutenção da aplicação se o *browser* não suportar a versão mais recente. Além disso, existem os acréscimos de recursos próprios (*tags* proprietárias) dos principais *browsers* Web disponíveis (*Netscape Navigator* e *Internet Explorer*).

5 O BANCO DE DADOS CACHÉ

Observa-se que os bancos de dados relacionais (BDR) quando utilizados em aplicações críticas, intensivas em transações, apresentam sérias deficiências no que diz respeito a desempenho e escalabilidade. Além disso, as tabelas bidimensionais dos BDR, apesar de extremamente fáceis para concepção, têm sido consideradas muito simples para modelar os dados do mundo real, Intersystems (2003a). Em relação às "antigas aplicações para banco de dados", as atuais aplicações corporativas tem novas exigências, Intersystems (2002):

a) são quantitativamente diferentes, pois apresentam bases de dados e volumes de transações enormes;
b) são qualitativamente diferentes, pois possuem maior riqueza e complexidade dos dados.

Extremamente econômico e eficaz como ambiente de produção, o *Caché* é um banco de dados que utiliza recursos do modelo de dados multidimensional para romper as limitações de desempenho e modelagem do modelo relacional. Produto da Intersystems Corporation, os aplicativos desenvolvidos em *Caché* oferecem, Intersystems (2002):

a) alta disponibilidade e segurança de dados;
b) eliminação de qualquer armazenamento desnecessário de informações;
c) alto desempenho e escalabilidade na disponibilização de Interfaces Gráficas para Usuários (aplicações GUI, do inglês *Graphical User Interface*) ou *World Wide Web* (WWW);
d) modificação dos modelos de dados de maneira eficiente;
e) superioridade de recursos em relação ao modelo relacional;
f) um fácil e rápido modelo de aplicação utilizando objetos;
g) uma nova LPOO: o *Caché Object Script*;
h) alta performance, através dos objetos;
i) rápido desenvolvimento de aplicações;
j) exportação nativa de objetos *Caché* como objetos Java;
k) exportação nativa de objetos *Caché* como objetos *ActiveX*, principalmente para *Visual Basic*;
l) dualização flexível de servidores;
m) desempenho elevado para sistemas *Unix*, via servidores de processos;

n) sistema GUI de 32 bits, para configuração e gerenciamento do *Caché* em todas as suas plataformas (figura 5.1).

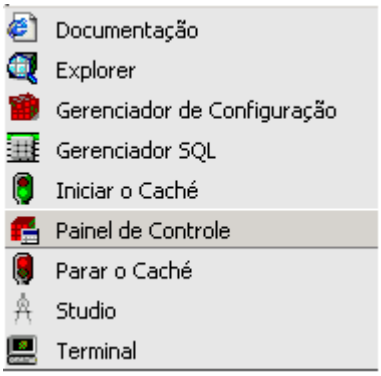


Figura 5.1 – A barra de ferramentas do *Caché*

O *Caché* é um banco de dados leve e rápido. Possui uma estrutura interna de "diretórios" denominados de NameSpace. Cada NameSpace pode armazenar, individualmente, objetos, programas e dados. Além disso, ele é livre de declarações e totalmente dinâmico. Livre de declarações significa que não é necessário declarar o número de dimensões e os limites de valores dos dados no modelo; não existem limitações no que diz respeito a esses dois aspectos. Totalmente dinâmico significa que não é necessário se preocupar com a administração de espaço em memória ou disco durante a eliminação ou adição de registros e atributos; também pode-se combinar manipulações simultâneas em estruturas de dimensões diferentes.

5.1 O MODELO DE DADOS MULTIDIMENSIONAL

O modelo de dados multidimensionais torna a modelagem de dados muito mais simples através da possibilidade de se apresentar o mundo real como ele realmente é, sem confina-lo a estruturas de dimensões restritas, como tabelas, por exemplo. No modelo de dados multidimensional a mesma estrutura física pode tomar quantas dimensões forem necessárias para modela-la de forma a atender as reais necessidades da aplicação. Por exemplo, para modelar uma Nota Fiscal em um modelo de dados relacional necessita-se de, pelo menos, duas tabelas: uma para os itens da nota e outra para sua capa. Transpondo essa mesma abordagem para o modelo multidimensional, pode-se imaginar a nota armazenada em um "cubo", que tem quantas faces sejam necessárias para modela-la. Isso significa um ganho

em performance, visto que os dados, fisicamente e logicamente, encontram-se na mesma estrutura, o que diminui as operações de leitura/gravação em disco, Intersystems (2003a).

As matrizes multidimensionais são a melhor forma de se representar estruturas complexas, Intersystems (2003a). A fim de obter melhor desempenho no processamento de transações, o *Caché* implementa o conceito de “matrizes esparsas”. As matrizes esparsas são estruturas que organizam o banco de forma que o espaço seja ocupado apenas por dados realmente existentes; dados não existentes não ocupam espaço.

O modelo multidimensional do *Caché* mapeia os dados em estruturas do tipo árvore, onde cada índice pode indicar o início de uma subárvore. Qualquer ramo abaixo de um nó está automaticamente relacionado com ele.

5.2 AS FORMAS DE ACESSO AOS DADOS DO CACHÉ

No núcleo do sistema *Caché* existe um ambiente de execução de banco de dados transacional (orientado a transações), baseado no modelo multidimensional de dados, de alto desempenho e com suporte a três caminhos de acesso aos dados: direto, SQL ou por objeto (figura 5.2). Dessa forma, ferramentas RAD (do inglês Rapid Application Development-Desenvolvimento Rápido de Aplicativos) como *Borland Delphi* e o *Microsoft Visual Basic* ou linguagens como o C++ e o Java podem utilizar do tradicional acesso via tabelas, enriquecer seus modelos de dados com os modelos de objetos ou otimizar seus processos com o acesso direto. Os mesmos dados podem ser acessados simultaneamente pelas três formas de acesso, com total controle de concorrência.

5.2.1 O acesso SQL

Indiscutivelmente, as atuais aplicações relacionais continuarão por um longo tempo a representar um a grande massa de dados. Assim, com a intenção de garantir compatibilidade e facilidade de migração, o *Caché* implementou o acesso relacional através do SQL. Dessa forma, o dicionário de dados do *Caché* permite a definição de visões e tabelas para a estrutura de dados multidimensional.

No entanto, o fato de se utilizar o *Caché* SQL como forma de acesso não significa sacrificar o desempenho e a escalabilidade, visto que o *Caché* SQL é implementado sobre uma base dados multidimensional e é otimizado através de comandos ‘M’ (da linguagem

Mumps), o que garante, segundo testes comparativos, um desempenho até 5 vezes mais rápido que as tradicionais aplicações relacionais, Intersystems (2002).

O *Caché* apresenta três interfaces para o acesso SQL:

a) ODBC (*Open DataBase Connectivity*): O ODBC é o mais utilizado padrão de interface de acesso a banco de dados relacionais, tanto por ferramentas de desenvolvimento de aplicações (como o Borland Delphi e o Microsoft Visual Basic), quanto de geração de relatórios (como o Microsoft Excel). O ODBC tornou-se "padrão' de mercado" devido à sua simplicidade para extração de informações de banco de dados;
b) JDBC (Java DataBase Connectivity): muito ao ODBC, o JDBC apresenta a vantagem de ser facilmente utilizado através da Internet pois é baseado na linguagem JAVA;
c) OCI (Oracle Calling interface): Interface desenvolvida pela InterSystems, o OCI permite a interligação direta entre o *Caché* e o *Oracle*. Através dela, uma estação Cliente *Caché* pode manipular tabelas em um Servidor *Oracle*. Esta interface é ideal para uma migração gradual de aplicativos *Oracle* para *Caché*.

5.2.2 O caché weblink

Além das interfaces disponibilizadas pelas três formas de acesso do *Caché*, a InterSystems desenvolveu a ferramenta *Caché WebLink*, que permite o acesso ao *Caché* através da WWW.

O *Caché WebLink* permite a utilização de qualquer uma das três formas de acesso ao *Caché*. As aplicações feitas em *WebLink* farão chamadas a rotinas armazenadas no servidor *Caché*, que poderão estar escritas em COS, SQL e/ou Sintaxe de Objeto. O servidor *Caché*, por sua vez, recebe a solicitação e retorna páginas Web criadas automaticamente, contendo os dados solicitados. Esta operação pode ser realizada tanto em ambiente Intranet quanto Internet.

5.2.3 O acesso ao objeto

Em contrapartida aos bancos de dados relacionais, que em alguns casos oferecem apenas recursos limitados em OO, o *Caché* dispõem uma estrutura completa de suporte à

tecnologia OO, incluindo herança múltipla, encapsulamento e polimorfismo. Os objetos *Caché* são projetados para desenvolvimento em ambiente de produção, oferecendo recursos para procesamento de transações sob os mais exigentes requisitos, como estruturas de dados complexas e processamentos de transações em ambiente baseado em rede.

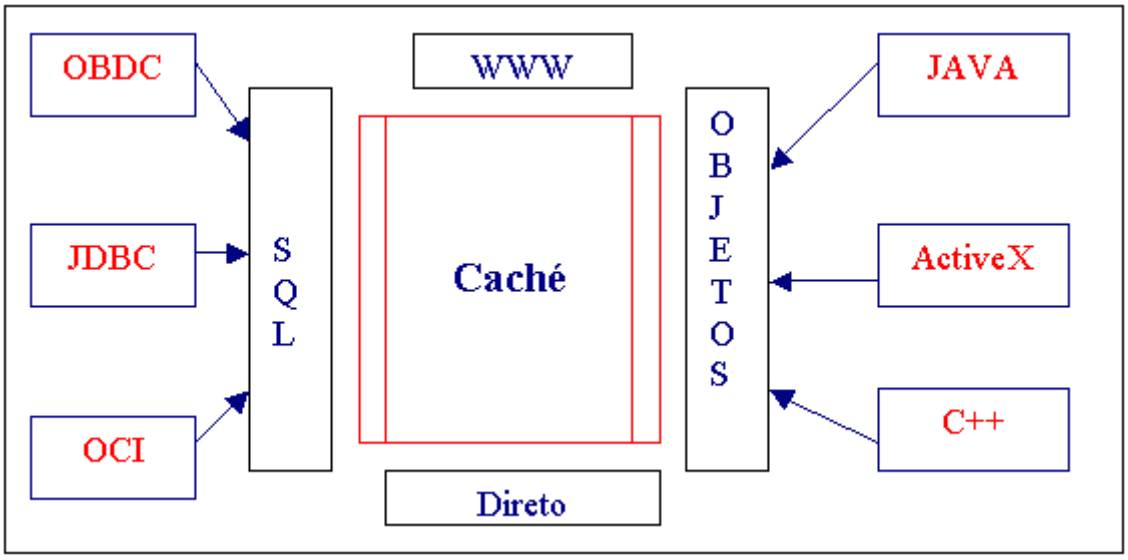
Os objetos *Caché* oferecem:

a) pleno suporte à herança, inclusive herança múltipla;
b) *advanced data types* (Tipos de Dados Avançados) para oferecer suporte eficiente aos tipos de dados nativos do sistema, de tipos de dados especiais conforme as necessidades de cada aplicação;
c) Um modelo completo de OO, incluindo OID, referências entre objetos e objetos "embutidos".

5.2.4 O Acesso direto

O acesso direto ao *Caché*, obtido através da linguagem orientada a objetos *Caché Object Script* (COS), disponibiliza um caminho direto à estrutura de dados do banco. Com o COS os desenvolvedores podem escrever rotinas de otimização de acesso e lógica de sistemas, executando-as através das ferramentas de acesso remoto do *Caché* ou por ferramentas de desenvolvimento via componentes *ActiveX*. Como resultado, os usuários obtém o menor tempo de acesso possível e a otimização máxima de seus processsos. O acesso direto garante o melhor desempenho possível do modelo de dados multidimensional.

O COS é uma extensão da linguagem Mumps (M) " padrão". Inúmeros conceitos e comandos foram incorporados ao M padrão a fim de disponibilizar, com o COS, uma única ferramenta para desenvolvimento de aplicações e definição de dados. O COS é a linguagem de objetos do *Caché*. Através dela são implementados e manipulados os objetos do *Caché*, Intersystems (1997).



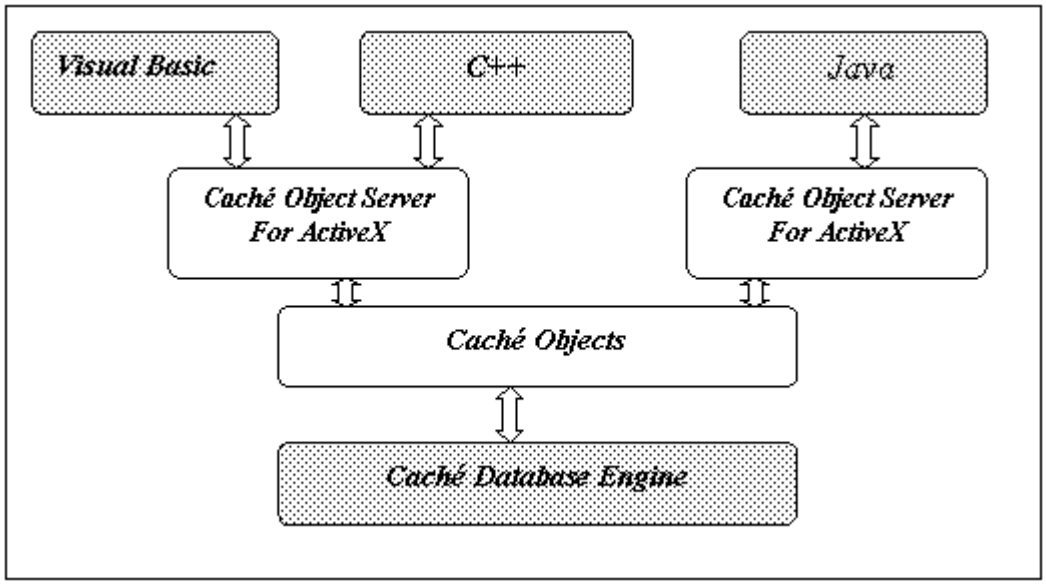
Fonte: Baher Jr.(1999)
Figura 5.2 O Caché e suas diversas formas de acesso

5.3 O CACHÉ OBJECTS

O *Caché Objects* é o componente (figura 5.3) da estrutura do banco de dados *Caché* responsável por disponibilizar, simultaneamente, o desempenho e o poder de modelagem da estrutura multidimensional de dados do *Caché* associada às características da tecnologia OO. Com o *Caché Objects* pode-se desenvolver aplicações contendo alto desempenho aliado às vantagens de desenvolvimento da tecnologia OO. Segundo Intersystems (1998) o *Caché Objects* é, ao mesmo tempo, uma LPOO e um SGBD que 'rodam' em cima da estrutura de dados do banco *Caché*.

Dentre as características do *Caché Objects* incluem-se:

a) fácil e rápido desenvolvimento de aplicações, utilizando Objetos;
b) apresentação de uma nova LPOO: O COS;
c) exposição de objetos *Caché* como objetos nativos Java;
d) exposição de objetos *Caché* como objetos nativos *ActiveX*, especialmente para *Visual Basic*;
e) alto desempenho.



Fonte: Intersystems (1998)
Figura 5.3: Visualização do Caché Objects

5.3.1 Estrutura do caché objects

O *Caché Objects* é um sistema (figura 5.4) formado pelos seguintes subsistemas:

a) *ClassDictionary*: O ClassDictionary é responsável pela armazenagem das definições de classe do usuário e do sistema *Caché*. Cada *NameSpace* do *Caché* possui uma *ClassDictionary*;

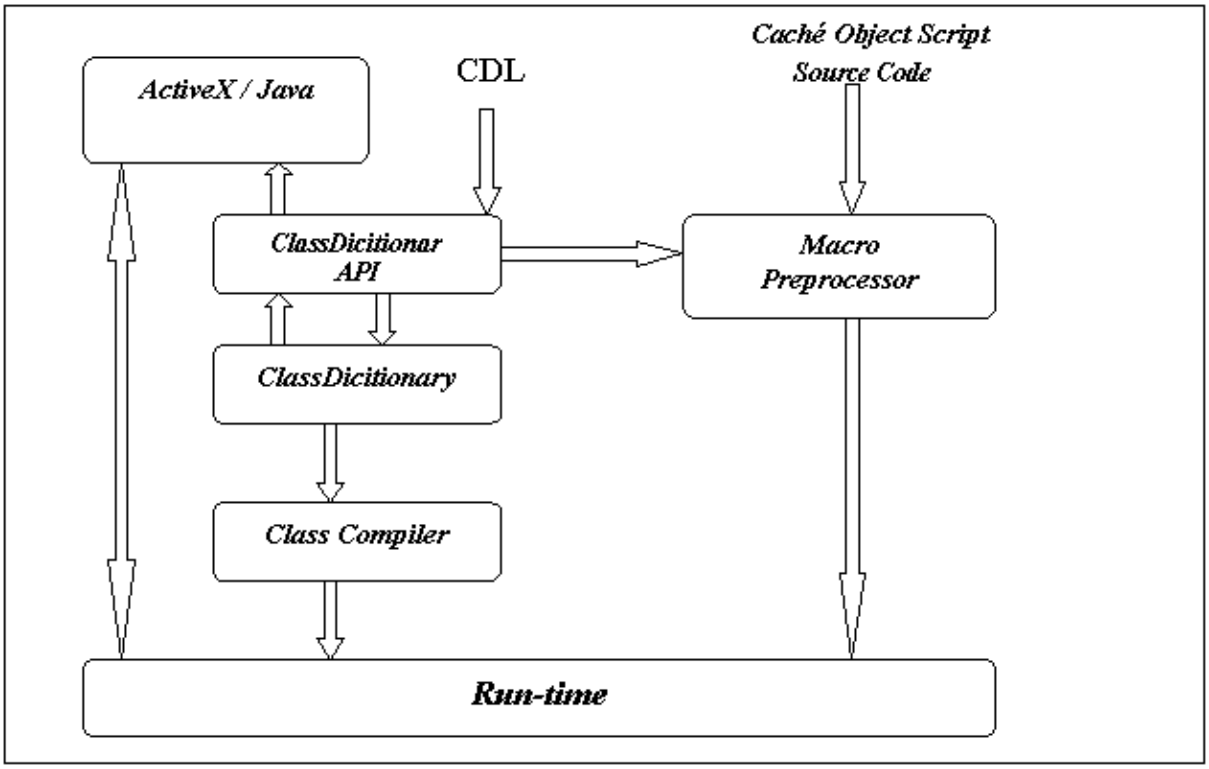
b) *ClassDictionary Application Program Interface (ClassDictionary API):* O ClassDictionary API, consiste em um conjunto de programas COS responsáveis pela comunicação entre o *ClassDictionary* e o restante dos componentes do *Caché Objects*;

c) *ClassCompiler:* O *Classcompiler* compila as definições de classe armazenadas no *ClassDictionary*;

d) *Caché Object Server for ActiveX:* O *Caché Object Server for ActiveX* é um componente *ActiveX* que permite a exposição de objetos *Caché* como objetos nativos *ActiveX* para utilização através de ferramentas de desenvolvimento como o *Borland Delphi* e o *Microsoft Visual Basic*;

e) *Caché Object Server for Java:* O *Caché Object Server for Java* permite a exposição de objetos *Caché* como objetos nativos Java;

f) *Macro Preprocessor:* Componente responsável pela “tradução” do código fonte COS para as funções da API do ClassDictionary.



Fonte: Intersystems (1998)
Figura 5.4: Os Subsistemas do *Caché Objects*.

5.3.2 O modelo de objetos do caché objects

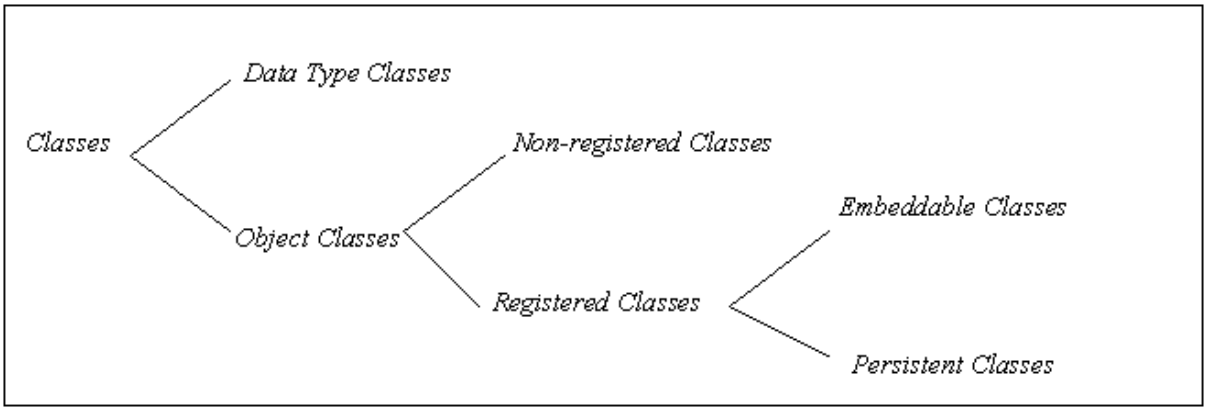
As operações fundamentais do *Caché Objects* estão baseadas na definição de classes de objetos e subseqüente criação, armazenagem, recuperação e manipulação de instâncias específicas dessas classes. Dentro do *Caché Objects* as classes estão divididas em vários tipos (figura 5.5). A divisão básica existente é a de Classes de Objetos (*Object Classes*) e Classes de Tipos de Dados (*Data Type Classes).*

As Classes de Objetos representam entidades específicas do mundo real e modelam como estas interagem com o mundo externo. Elas dividim-se em Classes Registradas (*Registered Classes)* e Classes Não Registradas *(Non-registered Classes).* As Classes Registradas permitem o armazenamento de suas instâncias em disco. As classes Não Registradas geralmente especificam classes abstratas e não podem ser instanciadas e/ou armazenadas.

As Classes Registradas ainda se subdividem em Classes Persistentes (Persistent Classes) e Classes embutidas (*Embeddable Classes).* As Classes Persistentes possuem para cada uma de suas instâncias um OID próprio através do qual podem ser manipuladas e

identificadas independente de seu estado interno. As Classes embutidas estão relacionadas às Classes Persistentes na forma de atributos destas, não possuem OID próprio e só podem ser manipuladas como atributos de Classes Persistentes.

As Classes de Tipos de Dados especificam valores literais, como inteiros, cadeias e datas para atributos de Classes de Objetos. Elas não possuem definições de métodos e nunca precisam ser instanciadas. Suas instâncias existem implicitamente no sistema *Caché*. Classes de Tipos de Dados não possuem OID e são identificadas por seus valores.



Fonte: Intersystens (1998)
Figura 5.5: Os Tipos de Classes do Caché Objects

O *Caché Objects* oferece amplo suporte a herança, inclusive com herança múltipla. Desta forma, classes de usuário podem ser definidas em função de classes de sistema já existentes. Existem duas formas de se definir classes de usuários no *Caché Objects*:

a) através do *Caché Object Architect*, uma ferramenta com interface GUI que serve para otimizar o processo de criação e manipulação de classes**;**

b) através do *Caché Class Description Language* (CDL), uma linguagem de definição de classes apresentada em um arquivo texto no formato ASCII.

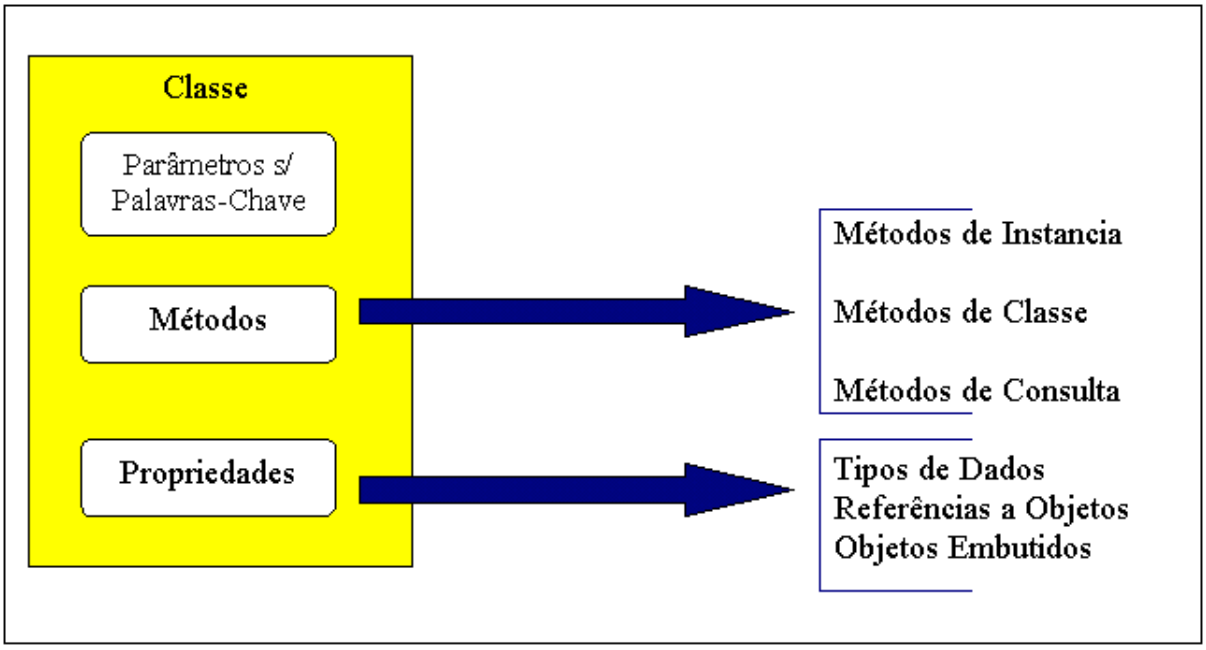
5.3.3 Classes no caché objects

As classes do *Caché Objects* são formadas por um conjunto de parâmetros/palavras-chave, métodos e propriedades (figura 5.6). Os parâmetros/palavras-chave de uma classe definem toda à parte “comportamental” da classe quando de sua opinião. Através dos parâmetros das classes pode-se definir índices, atributos chaves e até mesmo os valores (propriedades) que devem formar o OID do objeto. Com as palavras-chave define-se o tipo da classe, suas(s) superclasse(S), atribui-se uma descrição para a mesma, entre outras funções.

As propriedades (nome dado pelo *Caché* aos atributos de uma classe) modelam o conteúdo de uma classe. O valor das propriedades de uma classe determina seu estado interno. O *Caché Objects* permite ao usuário o acesso e validação direta das propriedades da instância de uma classe. As propriedades de uma classe podem ser tipos de dados atômicos (como inteiros e datas), referências a instâncias de objetos de outras classes (relacionamento) ou objetos embutidos (objetos que só podem ser acessados na forma de atributos de classe).

Os métodos definem como a classe se comunica com o meio externo. Existem três tipos de métodos no modelo de objetos do *Caché Objects*:

a) métodos de instância: Os métodos de instância são métodos implementados em COS e que só podem ser invocados através de um objeto instanciado. Os métodos de instância agem sobre a instância do objeto que o invocou;
b) métodos de classe: Implementados em COS, os métodos de classe são métodos invocados sem a presença de um objeto instanciado da classe. Eles geralmente são utilizados para instanciar objetos;
c) métodos de consulta: Os métodos de consulta, também conhecidos como *Query* no sistema *Caché*, agem sobre todo o conjunto de instâncias em disco dos objesto da classe e são implementados em COS e SQL. Os métodos de consulta são utilizados para processo de recuperação, análise e consulta de instâncias de objetos.



Fonte: Baehr Jr. (1999)
Figura 5.6: Conteúdo de uma Classe do *Caché Objects*

Trabalhos de conclusão de curso utilizando *Caché* podem ser vistos em Baehr Jr. (1999), Meyer (1999) e Obenaus (2000).

6 DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo será dividido em duas etapas. Na primeira parte, a Especificação, conterá a forma como o protótipo foi projetado e as ferramentas utilizadas. Na segunda parte será apresentada a implementação.

6.1 ESPECIFICAÇÃO

6.1.1 Análise de requisitos

O protótipo permitirá ao acadêmico:

a) Efetuar a reserva de vaga: o participante deverá efetuar sua reserva no evento escolhido, seja ele, seminário, feira, congresso, etc., escolhendo as opções no sistema;
b) Emitir comprovante de inscrição no evento: após confirmada a inscrição no evento o sistema emitirá um comprovante, que será utilizado pelo participante para garantir sua entrada no evento.

Para o desenvolvimento do protótipo, especificação e apresentação de um controle de reserva de vagas, utilizou-se a *Unified Modeling Language* (UML) composto de diversos diagramas, dos quais:

a) diagrama de caso de uso;
b) diagrama de classe;
c) diagrama de seqüência.

6.1.2 Diagrama de caso de uso

Na figura 6.1 está representado o diagrama de caso de uso, que demonstra as possíveis interações do usuário com o protótipo, que são:

a) Efetuar reserva: participante deve efetuar reserva, podendo este ser acadêmico ou não;
b) Emitir comprovante: o sistema emitirá um comprovante de reserva para o participante.

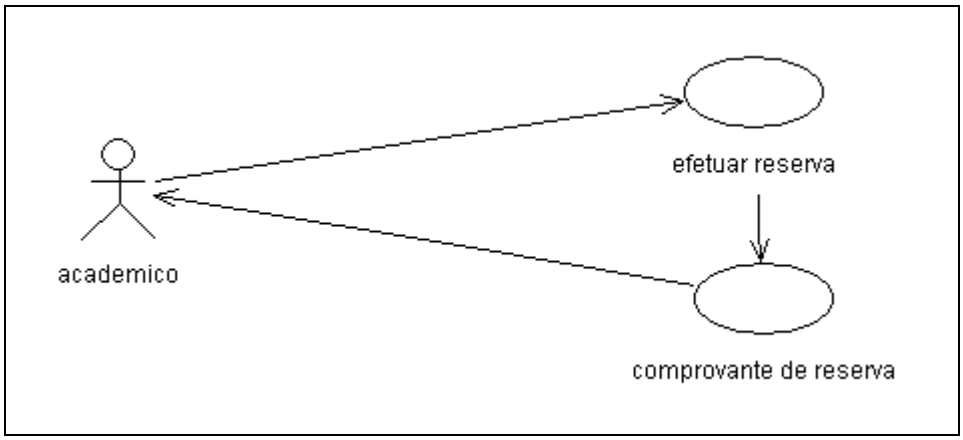


Figura 6.1 – Diagrama de caso de uso para o protótipo

6.1.3 Diagrama de classe

Na figura 6.2 está representado o diagrama de classe. Para representação deste diagrama foi utilizada a ferramenta *Rational Rose*.

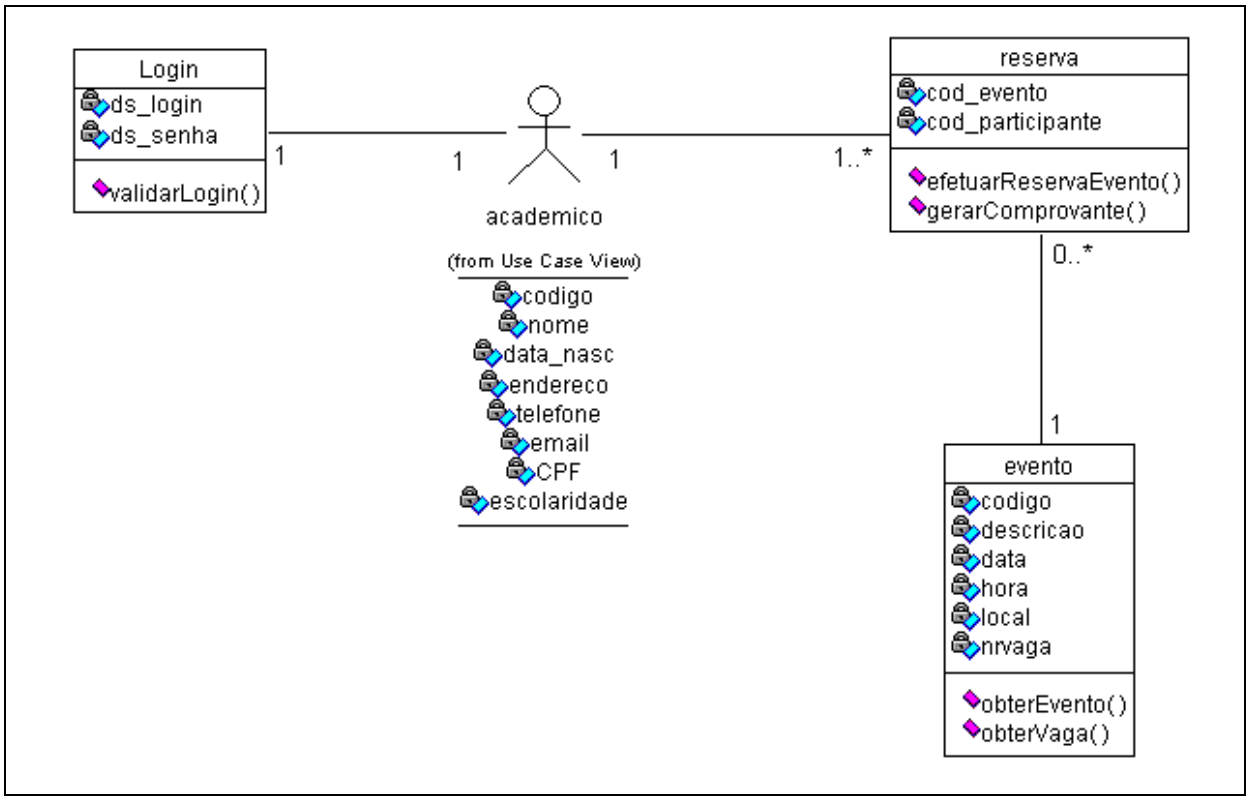


Figura 6.2 – Diagrama de classe para o protótipo

Para a implementação deste trabalho foram criadas as seguintes classes:

a) Acadêmico: classe utilizada para armazenar as informações do acadêmico;
b) Login: classe utilizada para validar as informações do acadêmico;
c) Evento: classe utilizada para armazenar informações sobre os eventos;
d) Reserva: classe encarregada de armazenar as informações referentes às reservas efetuadas.

6.1.4 Diagrama de seqüência

Através do diagrama de seqüência é representado os objetos e a troca de “mensagens”, conforme Figura 6.3, que demonstra o caso de uso “Efetuar Reserva”.

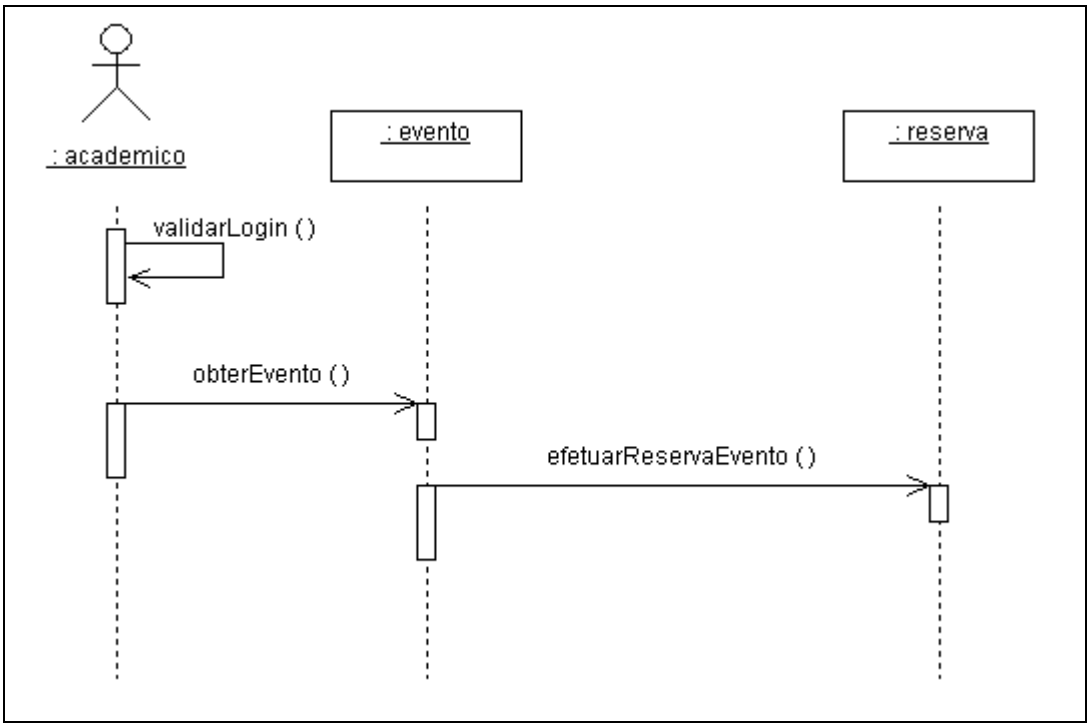


Figura 6.3 – Diagrama de seqüência – Efetuar reserva

O caso de uso “Emitir Comprovante” está demonstrado na Figura 6.4.

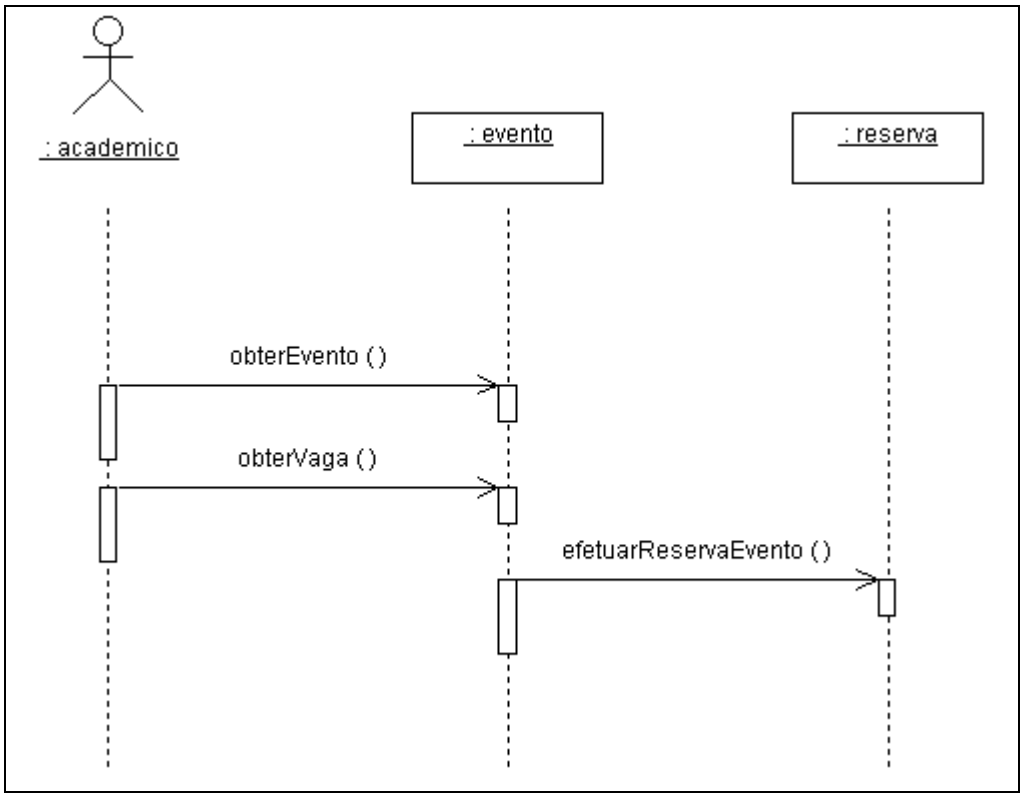


Figura 6.4 – Diagrama de seqüência – Emitir comprovante

6.2 TÉCNICAS E FERRAMENTAS UTILIZADAS

Este trabalho foi implementado para reproduzir uma das formas de acesso ao banco de dados *Caché*. A forma escolhida foi o acesso SQL via JDBC.

As ferramentas utilizadas para a programação do protótipo foram o Jbuilder 9 e o banco de dados *Caché*.

6.2.1 Ambiente de desenvolvimento Borland Jbuilber 9

O Jbuilder é um ambiente de desenvolvimento para linguagem Java desenvolvido pela Borland, e atualmente está na versão 9. Construído em padrões abertos, como a plataforma J2EE (Java 2 Enterprise Edition), o Jbuilder oferece grande flexibilidade para o desenvolvimento e a implementação de aplicações com o suporte a diversos sistemas operacionais.

Foi escolhido este ambiente pelo fato de ser uma linguagem visual compatível com a forma de acesso ao banco de dados proposto, e com o tema proposto.

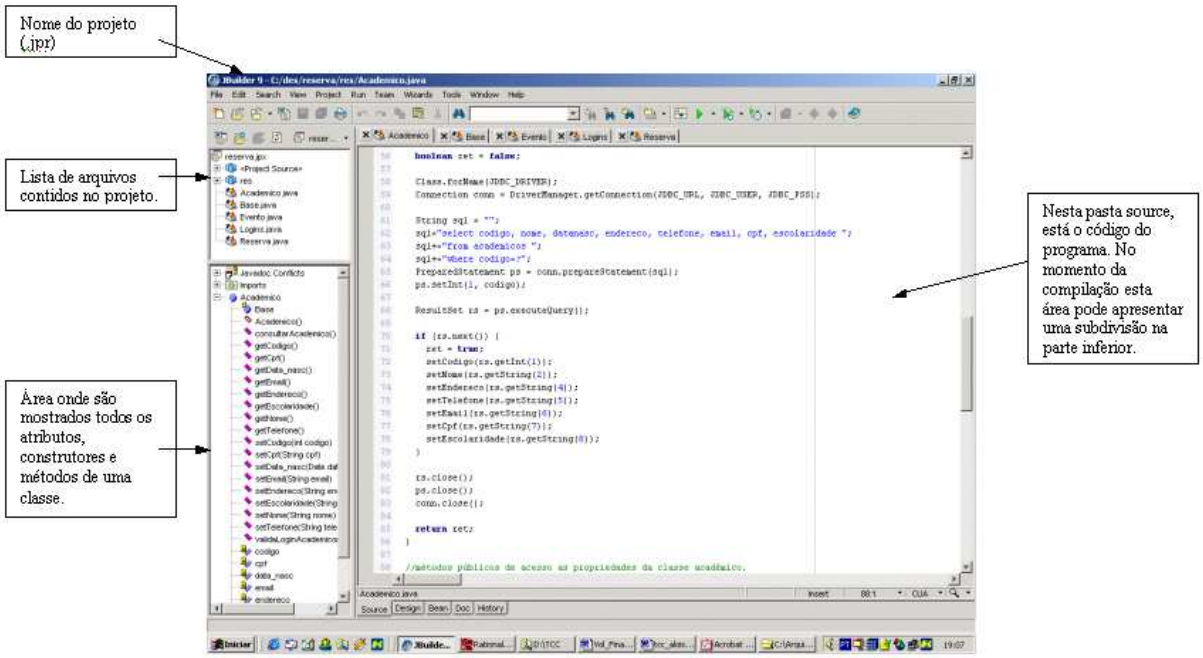


Figura 6.5 – Ambiente de desenvolvimento Jbuilder 9

6.2.2 Ambiente de definição de classes do caché

O *Caché Studio* (figura 6.6) é um ambiente com interface GUI que permite a manipulação completa das classes do *Caché Objects.* O *Caché Studio* possui *Wizards* para facilitar a criação de Classes, Atributos, Métodos e *Queries*. Também é possível exportar e importar, através do *Caché Studio*, definições de classes *Caché* como definições de classes em C ou Java.

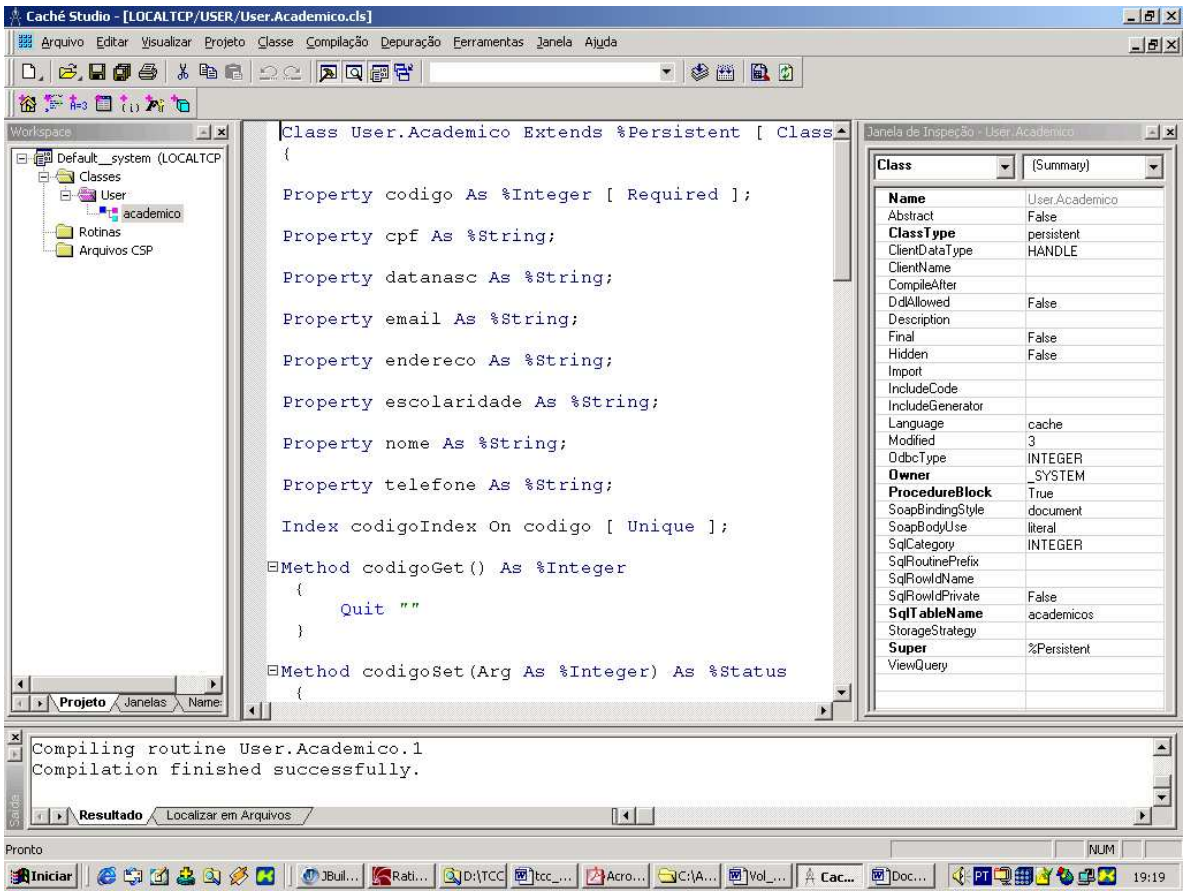


Figura 6.6 – O Caché Studio

6.3 IMPLEMENTAÇÃO

Este capítulo servirá para a apresentação das técnicas e ferramentas utilizadas na elaboração e efetiva implementação do protótipo.

6.3.1 Integração Web via JDBC com o caché

Segundo Intersystems (2003b), o driver JDBC do servidor *Caché* está no arquivo CacheJDBC.jar. Este arquivo pode ser encontrado no sub-diretório /Java, dentro do diretório de instalação do *Caché*. Este arquivo deve estar na variável de ambiente CLASSPATH antes de compilar ou executar as aplicações.

```
set CLASSPATH=c:\cachesys\java\CacheJDBC.jar;%CLASSPATH%
```

A primeira coisa a fazer é carregar o driver JDBC a ser utilizado. O nome da classe a ser carregada é com.intersys.jdbc.CacheDriver. Caso contrário o *ClassLoader* via disparar uma ClassNotFoundException quando o programa for executado, Intersystems (2003b).

```
//java code
try {
  Class.forName("com.intersys.jdbc.CacheDriver");
}
catch (ClassNotFoundException ex) {
  System.out.println(ex.getMessage());
}
```

Após este passo, o driver está pronto para estabelecer conexões com o servidor *Caché*.

O segundo passo é estabelecer uma conexão uma conexão com o servidor *Caché*. A URL abaixo endereça um servidor *Caché* na máquina local, respondendo pela porta *default*, 1972; e ainda aciona o *NameSpace* "User", Intersystems (2003b).

```
"jdbc:Cache://127.0.0.1:1972/User"
```

O segmento de código abaixo mostra como estabelecer uma conexão com o servidor *Caché*:

```
//java code
String url = "jdbc:Cache://127.0.0.1:1972/User";
String user = "_SYSTEM";
String password= "SYS";

Connection conn = DriverManager.getConnection(url, user, password);
```

6.3.2 Operacionalidade da implementação

Para que o acadêmico efetue sua reserva de vagas em um ou mais eventos, este deverá seguir a seguinte seqüência de atividades:

Efetuar *Login* conforme figura 6.7.

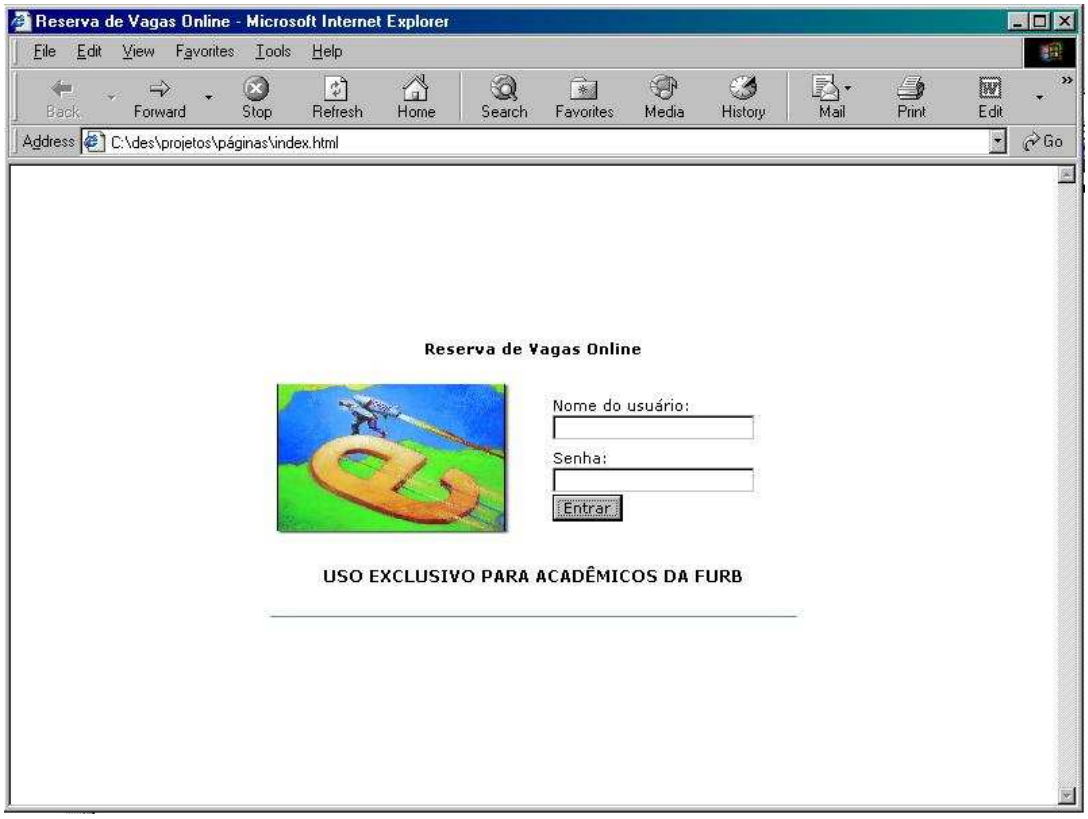


Figura 6.7: Login do sistema.

Após o *login* o acadêmico é recepcionado com uma tela de boas bindas, onde este deverá clicar no link “reserva de vagas” para prosseguir a efetivação da reserva, conforme figura 6.8.

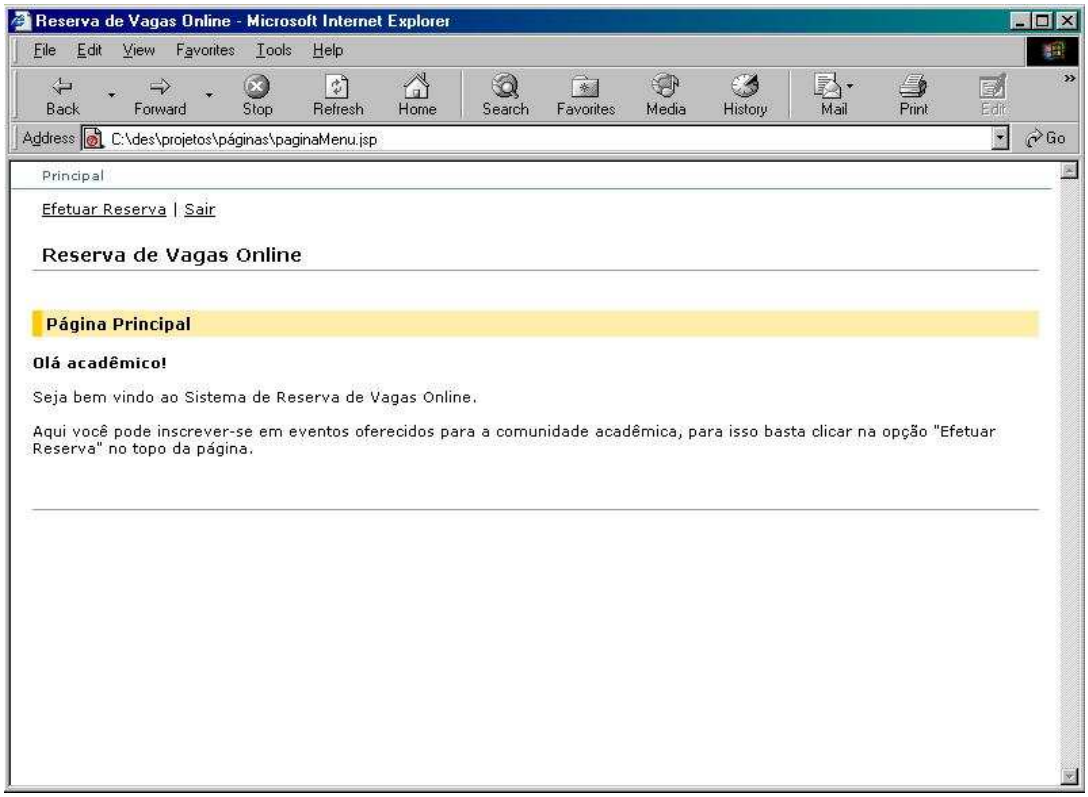


Figura 6.8: Boas vindas ao acadêmico

Seguindo a seqüência da reserva o acadêmico é levado a página onde estão listados os eventos oferecidos pela instituição(figura 6.9).

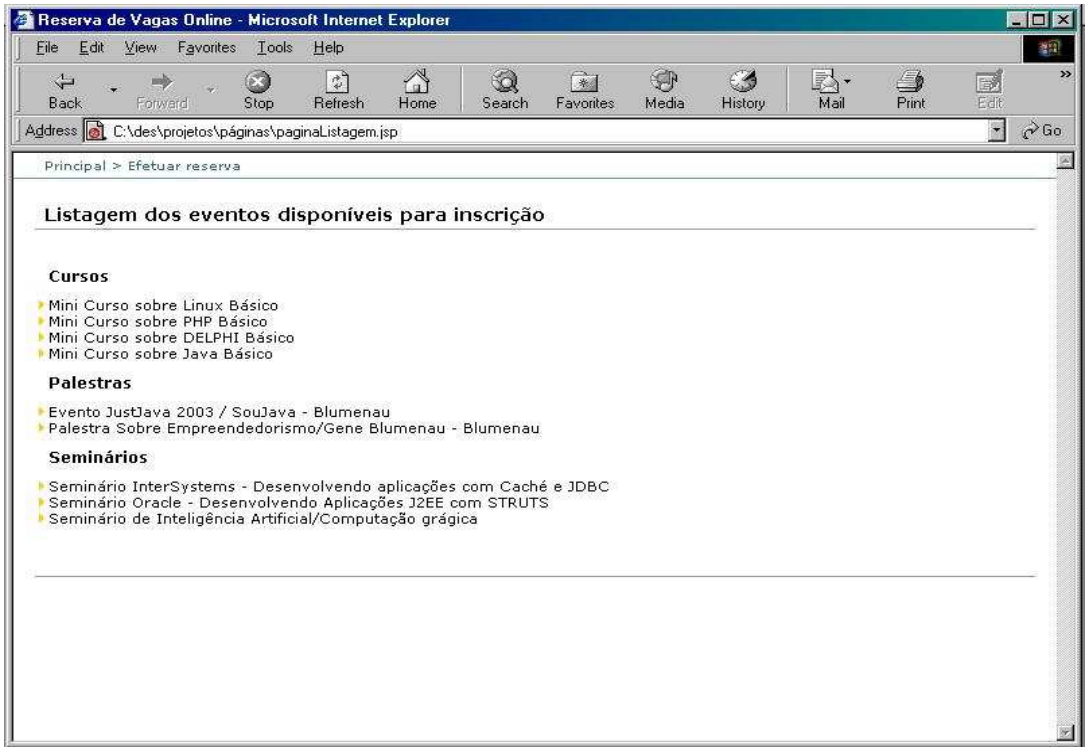


Figura 6.9: Lista de eventos

Escolhendo o evento, o acadêmico verá detalhadamente as informações referentes a este evento antes de confirmar a inscrição(figura 7.0).

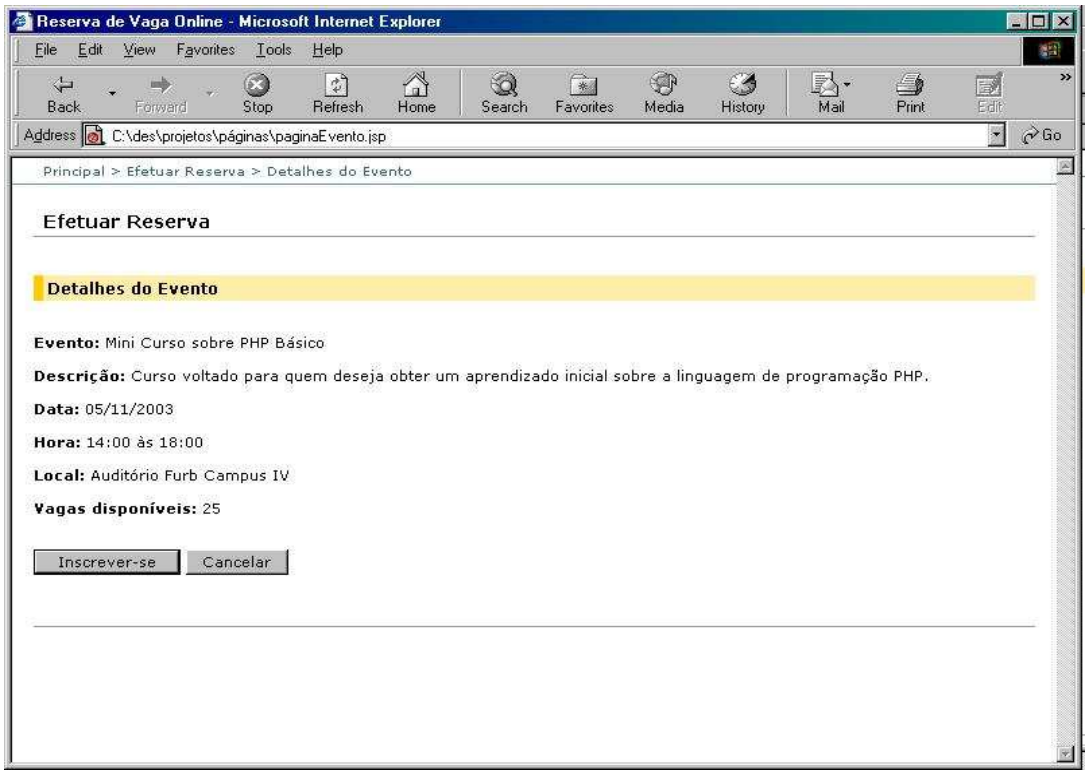


Figura 7.0: Informações detalhadas do evento

Efetivando a inscrição o sistema emite um comprovante ao acadêmico, com seu número de inscrição, finalizando assim o processo de inscrição em eventos (figura 7.1).

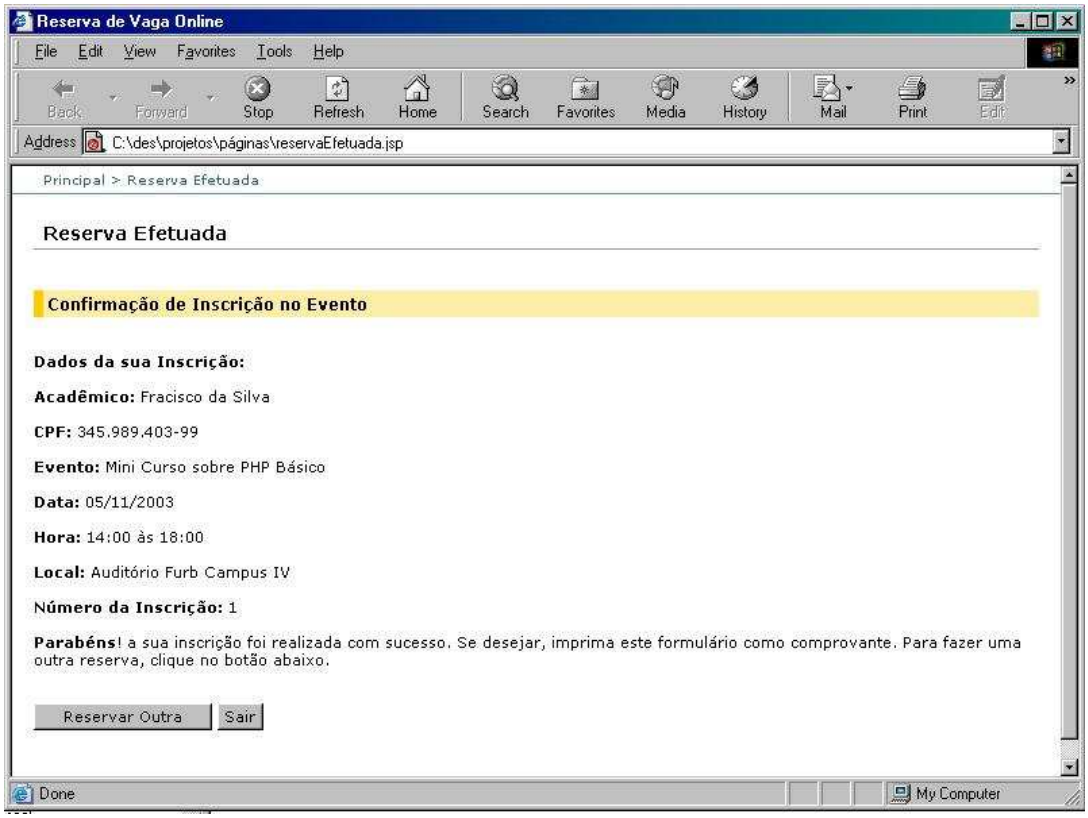


Figura 7.1: Comprovante de inscrição no evento

7 CONCLUSÕES

Neste capítulo serão apresentadas as conclusões finais sobre o trabalho e sugestões para trabalhos futuros.

7.1 CONSIDERAÇÕES SOBRE O BANCO DE DADOS ORIENTADO A OBJETOS DO CACHÉ

O modelo de dados orientado a objetos do *Caché* consiste basicamente em mapear as estruturas de armazenamento em árvore na forma de estruturas de classe, armazenando-as no *Class Dictionary* e manipulando-as através de rotinas geradas a partir da especificação da classe.

Em relação ao desempenho, não foi possível avaliar o que o banco pode revelar, isto porque o protótipo não implica em uma alta taxa de exigência de armazenagem e recuperação de informações. Todavia pelas pesquisas e comparações estudadas pela empresa proprietária do *Caché*, o ganho de tempo é muito grande em comparação com outro banco de dados atual no mercado, considerando os mesmos programas e volume de informações.

De forma geral, conseguiu-se atingir os objetivos propostos, através da implementação do protótipo e de estudos sobre o banco de dados utilizado, bem como realizando o teste na forma de acesso que o banco de dados permite.

7.2 LIMITAÇÕES

As limitações do protótipo são:

a) funcionar somente com banco de dados cujos fabricantes disponibilizem o driver JDBC, uma vez que a forma de acesso é via JDBC;
b) somente acadêmicos poderem participar dos eventos, pois deverão estar devidamente cadastrados no sistema.

Um problema encontrado foi com a versão do banco de dados *Caché* 4.1.6, que não funcionou com o acesso via JDBC, mesmo seguindo as instruções do tutorial. Sendo assim, a versão utilizada para o protótipo foi a 5.0.4, que atendeu perfeitamente ao objetivo deste protótipo.

7.3 SUGESTÕES PARA FUTUROS TRABALHOS

A título de sugestão, relacionou-se alguns tópicos para uso no desenvolvimento, que poderiam ser de grande utilidade:

a) desenvolvimento de sistemas complexos para averiguar a performance do banco de dados, já que o protótipo em si não conseguiu comprovar esta característica;
b) utilizar as outras formas de conexões com Java, seja projetando as classes do *Caché* como classes Java ou como *Enterprise JavaBeans* (EJB);
c) a implementação do mesmo protótipo em outro tipo de acesso que o banco de dados permita, como por exemplo, utilizando a ferramenta *WebLink* do *Caché*;
d) a implemantação de outros conceitos (reserva de vagas em outra áreas que não a acadêmica) para demonstração pelas facilidades encontradas no desenvolvimento deste.

REFERÊNCIAS BIBLIOGRÁFICAS

BAEHR Jr., Ivo. **Protótipo de sistema para gerenciamento de ordens de serviço acessando um banco de dados orientado a objetos e um banco de dados relacional**. Blumenau, 1999. 109f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, 1999.

BIAZIN, Alexandre. **Desenvolvimento de aplicações utilizando banco de dados orientado a objetos**. Blumenau, 1999. 73f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, 1999.

CELKO, Joe; CELKO, Jakie. **Verdades e mentiras sobre banco de dados de objetos.** Byte Brasil, São Paulo, n. 10, p.86-89, out. 1997.

CORBELLINI, Humberto. **Definição de uma metodologia para geração de aplicações e integração de banco de dados na Web**. Disponível em: < http://www.inf.ufrgs.br/pos/SemanaAcademica/Semana98/humberto.html>. Acesso em: 07 out. 2003.

DATE, C.J. **Introdução a sistemas de banco de dados**. Rio de Janeiro: Campus, 1991.

ELMASRI, R., NAVATHE, S. B. **Fundamentals of Database Systems**. Second Edition. The Benjamin/Cummings Publishing Company, Inc. 1994.

GREIN, Dirceu; TOMIFA, Katia Franncelino. **Sistemas de gerenciamento de banco de dados orientado a objetos.** Disponível em: <http://www.pr.gov.br/celepar/celepar/batebyte/edicoes/1998/bb78/estagiario.htm>. Acesso em: 07 out. 2003.

GUALBERTO, João. **Trabalho sobre banco de dados orientado a objetos.** Disponível em: < http://www.ufpa.br/sampaio/curso_de_sbd/bdoo/apostila/indice.html>. Acesso em: 07 out. 2003.

HADJEFTHYMIADES, S. P. e MARTAKOS, D. I. **A generic framework for the deployment of structured databases on the World Wide Web.** Disponível em: < http://www5conf.inria.fr/fich_html/paper-sessions.html >. Acesso em: 10 nov. 2003.

INTERSYSTEMS CORPORATION INC. **Caché Programming guide**. Cambridge: InterSystems Corporation, 1997.

INTERSYSTEMS CORPORATION INC. **User´s Guide**. Cambridge: InterSystems Corporation, 1998.

INTERSYSTEMS CORPORATION INC. **Uma nova era na tecnologia dos bancos de dados**. São Paulo: InterSystems Corporation, 2002. Disponível em: <

http://www.intersystems.com.br/downloads/ WhitepaperUmanovaeradeBD.pdf>. Acesso em: 18 jan. 2002.

INTERSYSTEMS CORPORATION INC. **Caché**. Cambridge: InterSystems Corporation, 2003a.

INTERSYSTEMS CORPORATION INC. **Caché e JDBC**. São Paulo: InterSystems Corporation, 2003b. Disponível em: < http://www.intersystems.com.br/downloads/ Cache_JDBC.pdf>. Acesso em: 21 out. 2003b.

KERN, Vinícius. **Banco de dados relacionais**. São Paulo: Érica, 1994.

KHOSHAFIAN, Setrag. **Banco de dados orientado a objeto**. Rio de Janeiro: Infobook, 1994.

KORTH, Henry F., SILBERSCHATZ, Abraham. **Sistema de banco de dados**, 2ªed. São Paulo: Makron Books, 1997.

LIMA, Iremar Nunes. **O ambiente web banco de dados:** Funcionalidades e arquiteturas de integração. 1997. 92f. Dissertação (Mestrado de Informática). Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 1997.

MACIASEK, Leszek A. **Relational versus objetct database: contention or coexistence?** Disponível em: < http://www.comp.mq.edu.au/courses/comp866/oovsrel.html>. Acesso em: 07 out. 2003.

MARTIN, James; ODELL, James J.. **Análise e projeto orientados a objeto.** São Paulo: Makron Books, 1995.

MEYER, Bruno. **Protótipo de sistema de rastreabilidade de produtos utilizando o Banco de Dados Caché**. Blumenau, 1999. 58f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, 1999.

MURTA, C. D., ALMEIDA, J. M. e ALMEIDA, V. A. F. **Análise de Desempenho de um Servidor WWW**, Anais do XXII Seminário Integrado de Software e Hardware, IX CTD, XV CTIC, Recife, PE, 1996, pp. 391-402.

OBENAUS, Maurício Rogério. **Protótipo de uma aplicação comercial utilizando banco de dados Caché com interface WEB**. Blumenau, 2000. 56f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, 2000.

OBJECTSTORE. **Object-oriented database systems.** Disponível em: <http://www.rs6000.ibm.com/resource >. Acesso em: 07 out. 2003.

PINHEIRO, Carlos André Reis. Fundamentos de bancos de dados orientados a objetos. **Developers cioMagazine**, São Paulo, v. 1, n. 44, p. 18-20, abril 2000.

RAHMEL, Dan. **Systems and Methodologies for Identifying and Protecting Weak Spots in Your Web-Enabled Database.** Disponível em: < http://www.dbmsmag.com/9704i03.html >. Acesso em: 10 nov. 2003.

RAO, Bindu Rama. **Object-oriented databases.** Estados Unidos: McGraw-Hill, 1994.

SHKLAR, Leon. **Web Access to Legacy Data.** Disponível em: < http://www.cs.rutgers.edu/~shklar/web-legacy/tutorial.html >. Acesso em: 10 nov. 2003.
SILVA, Fernando. **Protótipo de um sistema gerenciador de banco de dados orientado a objetos**. Blumenau, 2001. 64f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, 2001.

UESSLER, Sidney. **Estudo das características do Banco de Dados orientado a objetos Jasmine**. Blumenau, 1999. 61f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, 1999.