UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE SISTEMA PARA GERENCIAMENTO DE ORDENS DE SERVIÇO ACESSANDO UM BANCO DE DADOS ORIENTADO A OBJETOS E UM BANCO DE DADOS RELACIONAL

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

IVO BAEHR JUNIOR

BLUMENAU, JUNHO/1999

1999/1-21

PROTÓTIPO DE SISTEMA PARA GERENCIAMENTO DE ORDENS DE SERVIÇO ACESSANDO UM BANCO DE DADOS ORIENTADO A OBJETOS E UM BANCO DE DADOS RELACIONAL

IVO BAEHR JUNIOR

ESTE TRABALHO DE CONCLUSÃO DE CURSO FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

Prof. Marcel Hugo - Orientador na FURB

Prof. José Roque Voltolini da Silva - Coordenador do TCC

BANCA EXAMINADORA

Prof. Marcel Hugo

Prof. Oscar Dalfovo

Prof. Everaldo Artur Grahl

DEDICATÓRIA

À minha noiva, Fernanda Schmitt, pelo apoio, incentivo e carinho que recebi durante a elaboração deste trabalho.

AGRADECIMENTOS

Agradeço a Deus,

que me deu força nos momentos de dificuldade.

Agradeço meus familiares,

por tudo que me ensinaram.

Agradeço o professor orientador Marcel Hugo,

pelas valiosas contribuições para com esse trabalho.

Agradeço a empresa Qualix Soluções e Serviços, distribuidora do Banco de Dados Caché,

pela atenção dedicada a este trabalho.

Agradeço meus amigos,

que sempre estiveram ao meu lado nos momentos de dúvida.

Enfim, agradeço a todos que de uma forma ou de outra contribuíram para a elaboração deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

ADT	- *Abstract DataType* (Tipo Abstrato de Dados)
API	- *Application Program Interface* (Interface de Programação da Aplicação)
BDOO	- Banco de Dados Orientado a Objeto
BDR	- Banco de Dados Relacional
CAD	- *Computer-Aided Design* (Projeto Auxiliado por Computador)
CAM	- *Computer-Aided Manufacture* (Manufatura Auxiliado por Computador)
CASE	- *Computer-Aided Software Enineering* (Engenharia de Software Auxiliada por Computador)
CDL	- *Class Definition Language* (Linguagem de Definição de Classes)
COBOL	- *Common Business Oriented Language* (Linguagem Comum Orientada para Negócios)
CODASYL	- *Conference On DAta SYstems and Languages*
COS	- *Caché Object Script*
DAG	- *Directed Acyclic Graph* (Gráfico Acíclico Direcionado)
DBA	- *Database Administrator* (Administrador do Banco de Dados)
DBMS	- *Database Management Systems* (Sistema Gerenciador de Banco de Dados)
DBTG	- *Data Base Task Group*
DDL	- *Data Description Language* (Linguagem de Definição de Dados)
DML	- *Data Manipulation Language* (Linguagem de Manipulação de Dados)
ER	- Entidade – Relacionamento
GUI	- *Graphical User Interface* (Interface Gráfica para Usuário)
IDC	- *International Data Corporation*
IDE	- *Integrated Development Environment* (Ambiente Integrado de Desenvolvimento)
IEC	- *International Electrotechnical Commission*
ISO	- *International Organization for Standardization*
JDBC	- *Java DataBase Connectivity* (Conexão Java para Banco de Dados)
LAN	- *Local Area Networks* (Redes Locais)
LPOO	- Linguagens de Programação Orientadas a Objeto
M	- Mumps

ODMG	- *Object Database Management Group*
OCI	- *Oracle Calling Interface* (Interface de Chamada ao Oracle)
ODBC	- *Open DataBase Connectivity* (Conexão Aberta para Banco de Dados)
OID	- *Object Identifier* (Identificador do Objeto)
OIS	- *Office Information Systems* (Sistemas de Informação para Escritórios)
OO	- Orientação a Objeto
OQL	- *Object Query Language* (Linguagem de Consulta de Objetos)
OS	- Ordens de Serviço
RAD	- *Rapid Application Development* (Desenvolvimento Rápido de Aplicação)
SGBD	- Sistema Gerenciador de Banco de Dados
SGBDOO	- Sistema Gerenciador de Banco de Dados Orientados a Objeto
SGBDR	- Sistema Gerenciador de Banco de Dados Relacional
SQL	- *Structured Query Language* (Linguagem de Consulta Estruturada)
UDT	- *User DataType* (Tipo de Dados do Usuário)
WWW	- *World Wide Web*

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 para gerenciamento de ordens de serviço acessando o banco de dados Caché, produto da empresa InterSystems Co, de maneira relacional e orientada a objetos, a fim de sugerir quais são as características que um desenvolvedor deve observar e avaliar, na forma de critérios, em um sistema gerenciador de banco de dados para agilizar ao máximo o processo de desenvolvimento de software aplicativo. As características a serem observadas terão por base aquelas definidas pela NBR 13596 - Qualidade de Software Produto.

ABSTRACT

This work consists of a study on object-oriented database. In that study they are appraised the problems and the means found during the process of development of a system prototype for management of orders of service access the database Caché, product of InterSystems Co company, of way relational and object-oriented, in order to suggest which are the characteristics that a deployment should observe and to evaluate, in the form of approaches, in a system database management to activate to the maximum the process of development of software application. The characteristics be she observed they will have for base those defined by NBR 13596 - Quality of Software Product.

1 INTRODUÇÃO

1.1 Origem

Atualmente, está-se presenciando a emergência da tecnologia da orientação a objetos em todos os segmentos da computação: linguagens de programação, simulação, interfaces gráficas para usuários e, é claro, banco de dados [KHO94]. No mesmo instante, depara-se com o paradigma do projeto de sistemas utilizando técnicas de orientação a objetos armazenando seus dados em banco de dados relacionais, modelando a "realidade" na forma de tabelas [MAR95].

Porém, o mundo não é uma coleção de tabelas. Um modelo comum da realidade é uma hierarquia de montagem encontrada em componentes. Por exemplo, uma "peça" é decomposta em componentes, que são decompostos em subcomponentes adicionais e assim sucessivamente. Esses dados são, com certeza, difíceis de se manipular utilizando tabelas. De fato, ao utilizar uma instrução na linguagem *Structured Query Language* (SQL), não se consegue retornar todos os componentes de uma peça com um simples comando de consulta (*query*). Em contraste, um banco de dados orientado a objetos suporta tanto a capacidade dos objetos referirem-se diretamente uns aos outros, como a facilidade de serem processados pelas linguagens de programação [KHO94].

Além das vantagens com relação a armazenamento de dados complexos, tais como *computer-aided design* (CAM - projeto auxiliado por computador) e *computer-aided manufacture* (CAD - manufatura auxiliada por computador), os banco de dados orientados a objetos apresentam "vantagens" sobre os bancos de dados relacionais no que diz respeito a redundância – reduzida através do uso da herança – e de desempenho – com os acessos diretos. Em contrapartida, depara-se com pontos que podem ser considerados "desvantagens", tais como o encapsulamento dos dados e a complexidade da estrutura dos objetos [MAR95].

O banco de dados Caché oferece múltiplos caminhos para acesso aos dados armazenados em um modelo de dados. Acesso relacional, por meio do SQL, acesso direto por meio da linguagem Caché Object Script (COS) ou acesso por objetos. Todos os três métodos podem acessar simultaneamente os mesmos dados, com total concorrência. Uma vez que o dado se encontre no banco de dados, você possui inúmeras maneiras de acessá-lo. Dessa

forma, pode-se ter o mesmo modelo de dados trabalhando ambas as tecnologias (Relacional e Orientada a Objetos) tornando a avaliação de qual tecnologia possui melhor desempenho um tanto quanto mais fácil [INT97a] [INT97b].

Por fim, este trabalho visa confrontar os problemas e facilidades encontrados durante o desenvolvimento de um protótipo de sistema para gerenciamento de ordens de serviço acessando o banco de dados Caché de maneira relacional e orientada a objetos, a fim de sugerir quais são as características que um desenvolvedor deve observar e avaliar, na forma de critérios, em um sistema gerenciador de banco de dados para agilizar ao máximo o processo de desenvolvimento de software aplicativo. As características a serem observadas terão por base aquelas definidas pela NBR 13596 - Qualidade de Software Produto.

1.2 Objetivos

O principal objetivo desse trabalho é especificar e implementar um protótipo de sistema para gerenciamento de ordens de serviço acessando o banco de dados Caché de maneira relacional e orientada a objetos. Pretende-se avaliar os problemas e facilidades encontradas a fim de sugerir quais são as características, na forma de critérios, que um desenvolvedor deve observar e avaliar em um sistema gerenciador de banco de dados para agilizar ao máximo o processo de desenvolvimento de software aplicativo.

Como objetivos secundários do trabalho tem-se:

a) implementar no protótipo os conceitos de manipulação de banco de dados orientados a objeto;
b) implementar no protótipo o acesso relacional e orientado a objetos aos dados no Caché;
c) sugerir, com base na NBR 13596 - Qualidade de Software Produto, critérios de avaliação para sistemas gerenciadores de banco de dados.

1.3 Organização

Este trabalho encontra-se divido da seguinte forma:

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

No capítulo 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 objeto, 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 estão descritos os critérios de avaliação utilizados para o levantamento das características que um desenvolvedor deve observar em um sistema gerenciador de banco de dados.

No capítulo 6 é apresentada a especificação do protótipo desenvolvido neste trabalho bem como o resultado da aplicação dos critérios de avaliação.

Finalmente, no capítulo 7 são apresentadas as conclusões e sugestões para futuros trabalhos.

2 SISTEMA GERENCIADOR DE BANCO DE DADOS

Segundo [KOR97] um Sistema de Gerenciamento de Banco de Dados (SGBD) consiste em uma coleção de dados inter-relacionados e um conjunto de programas para acessá-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.

Para [KER94], Sistema Gerenciador de Banco de Dados ou SGBD (ou ainda DBMS do inglês *Database Management Systems*) é um sistema que serve para gerenciar dados armazenados organizadamente, permitindo todo tipo de operação de manutenção e consulta nesses dados. Essas operações são realizadas através de ferramentas e instruções de alto nível onde o usuário diz o que fazer e o SGBD se encarrega de como faze-lo.

Já [DAT90] define um 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.

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. O projeto geral de um banco de dados denomina-se de esquema do banco 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 de instância do banco de dados.

2.1 Motivos do Surgimento do SGBD

Antes do surgimento dos SGBD os sistemas eram desenvolvidos voltados para um único setor da empresa e resumiam-se a um aglomerado de programas aplicativos encarregados de extrair e adicionar registros nos devidos arquivos [GRO96]. Sendo assim, a visão gerencial de toda empresa era constituída de uma série de sub-sistemas, cada um com seus arquivos e respectivos programas de manipulação. Esse típico sistema de processamento de arquivos, onde os registros são guardados em diversos arquivos e uma grande gama de

programas é escrito para manipulá-los possui um grande número de desvantagens [KOR97] [KER94]:

a) redundância e inconsistência de dados. O fato de definirem-se diversos arquivos para atender cada um dos setores da empresa aliado à implementação de vários programas para acessar tais dados cria tanto o problema da redundância quanto o da inconsistência visto que, diferentes setores podem necessitar de um mesmo cadastro, defini-lo de forma diferente e ainda implementar diferentes processos para acessá-los;

b) dificuldade no acesso aos dados. Como o sistema de informação da empresa num todo consiste em uma série de arquivos, a junção de dados a fim de obter-se uma informação gerencial (como a variação dos estoques, por exemplo) necessita de todo um esforço dirigido na obtenção de todos os dados necessários nos diversos arquivos existentes;

c) dados isolados. Devido a existência de vários arquivos fica difícil a implementação de novos aplicativos para recuperação de dados específicos localizados em pontos distintos;

d) problemas com acesso concorrente. Com acesso multi-usuário aos dados unido ao fato de diversos programas aplicativos se encarregarem de manipular os dados diretamente surge o perigo do acesso concorrente sem restrições de bloqueio, gerando inconsistências;

e) problemas de segurança. Novamente o fato do acesso aos dados ser gerenciado pelos próprios programas aplicativos dá liberdade para que qualquer usuário tenha acesso a informações confidenciais;

f) problemas de integridade. Implementar restrições de integridade, como valores máximos ou mínimos, em sistemas onde a gravação dos dados é feita diretamente pelos programas aplicativos demanda muito mais trabalho quando da necessidade de uma alteração;

g) aumento do custo de manutenção. Como a mesma tarefa está sendo realizada em diversos lugares, a manutenção de tais tarefas demanda mais tempo de trabalho.

2.2 Objetivos do SGBD

Além de sanar as desvantagens citadas anteriormente e disponibilizar as informações aos usuários quando necessário, pode-se dizer que os SGBD possuem mais dois importantes objetivos:

a) disponibilizar a abstração dos dados [KOR97]: Um dos grandes objetivos dos SGBD, a abstração dos dados tem por finalidade omitir do usuário detalhes de como os dados são manipulados e mantidos. Para executar eficazmente tal objetivo os SGBD escondem tais detalhes complexos através de três níveis de abstração (figura 2.1):

- nível físico: é o mais baixo nível de abstração e descreve como os dados estão armazenados fisicamente no banco. Esse nível está ligada ao modelo de dados;
- nível conceitual: nesse nível são definidos quais dados serão armazenados no banco e quais são os relacionamentos entre eles. Independe ao usuário desse nível se a modelagem de tais dados necessitar de estruturas físicas complexas. Esse nível está ligado ao esquema de dados;
- nível de visões: Este é o mais alto nível de abstração e tem por objetivo fornecer diferentes visões de um mesmo banco de dados. Através desse nível é possível disponibilizar a cada usuário apenas o relevante e importante ao mesmo, omitindo informações que não lhe interessem ou que sejam confidenciais;

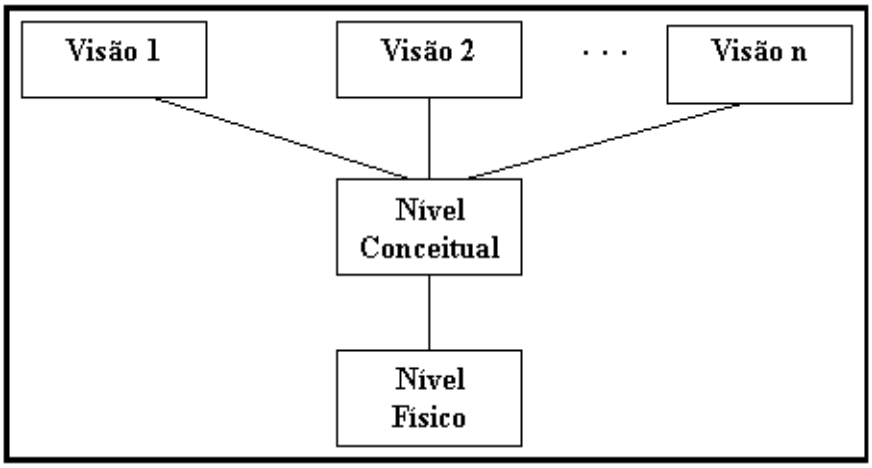

Figura 2.1 - Os Três Níveis de Abstração de Dados (Fonte: KOR97).

b) disponibilizar a independência entre dados e programas: Segundo [DAT90], a independência dos dados pode ser definida como a imunidade das aplicações à estrutura de armazenamento e à estratégia de acesso, ou seja, a forma como os dados são armazenados e a maneira como são acessados não estão embutidos nem na lógica nem no código da aplicação. Diz-se que uma aplicação depende dos dados quando alguma mudança na estrutura de armazenamento ou na forma de acesso implique em afetar diretamente a aplicação. Existem pelo menos dois motivos para que tal empecilho não exista em um SGBD:

- aplicações diferentes necessitam de diferentes visões dos dados: abstração em nível de visões;
- o *Database Administrator* (DBA) deve ter autonomia para modificar tanto a estrutura física de armazenamento quanto as formas de acesso ou ambas, sem comprometer as aplicações já existentes: abstração em nível físico.

2.3 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 (*Common 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 dos dados.

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 bancos 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 [KHO94].

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.

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 comercializadas.

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 ou de rede. Maiores explicações sobre modelos de dados semânticos podem ser obtidos em [KHO94] e [KOR97].

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 relacional a fim de obter maior flexibilidade. Complementos sobre modelo de objetos complexos podem ser encontrados em [KHO94] e [KOR97].

Entretanto, a evolução dos SGBD não está unicamente ligada a 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 o bancos de dados orientados a objetos. Na verdade, sistemas e produtos de bancos 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 e o processo de downsizing [KHO94].

No processo de evolução dos SGBD, sintetizado na figura 2.2, é preciso uma visão ampla e lembrar dos bancos 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 de multimídia (voz, textos, gráficos).

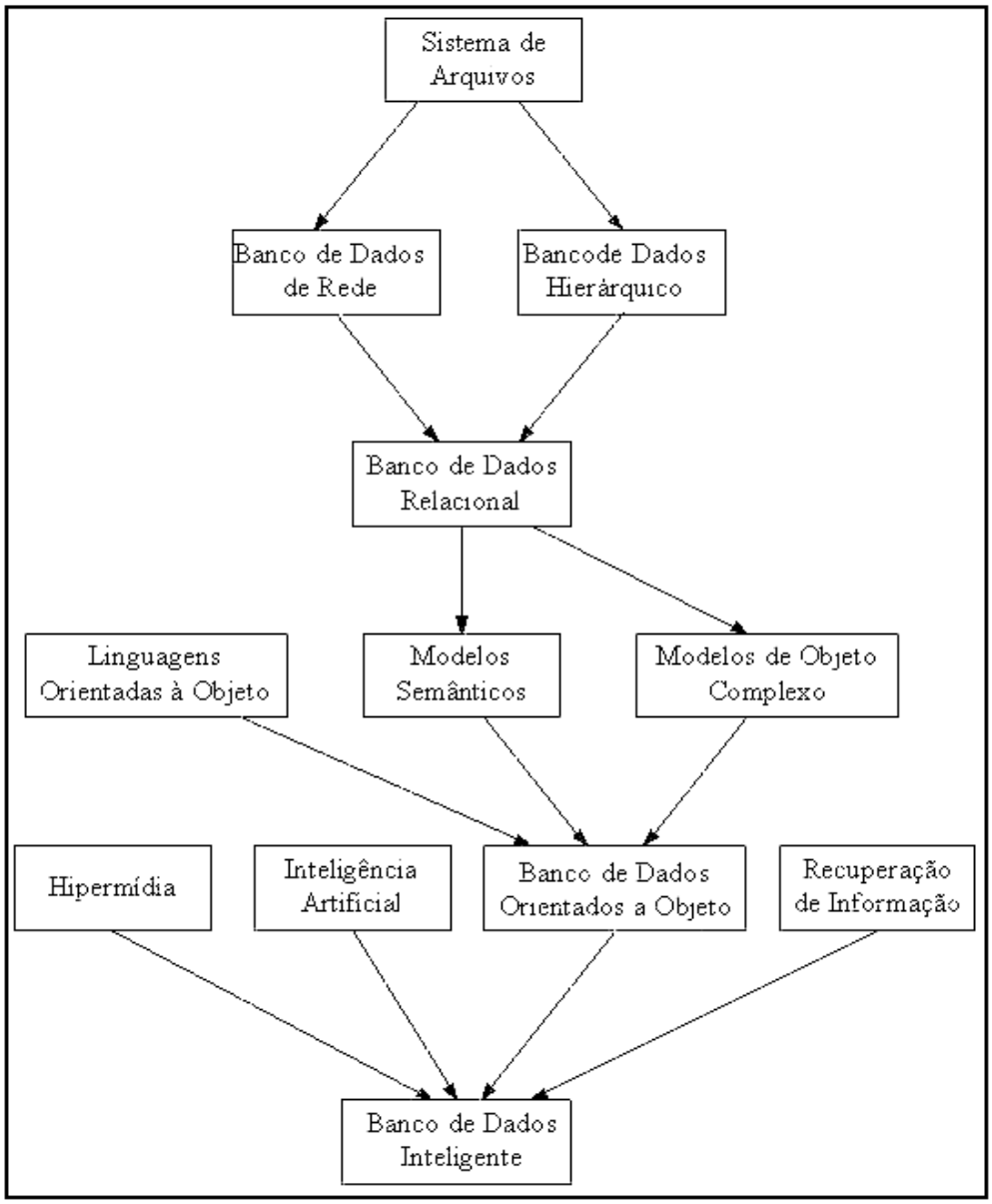

Figura 2.2 - A Evolução dos Banco de Dados (Fonte: KHO94).

2.3.1 O Modelo Hierárquico

Os SGBD do modelo hierárquico foram provavelmente os primeiros sistemas disponíveis comercialmente. Segundo [KOR97], 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 [KHO94] 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. [DAT90] define que um banco de dados hierárquico compõe-se de 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.

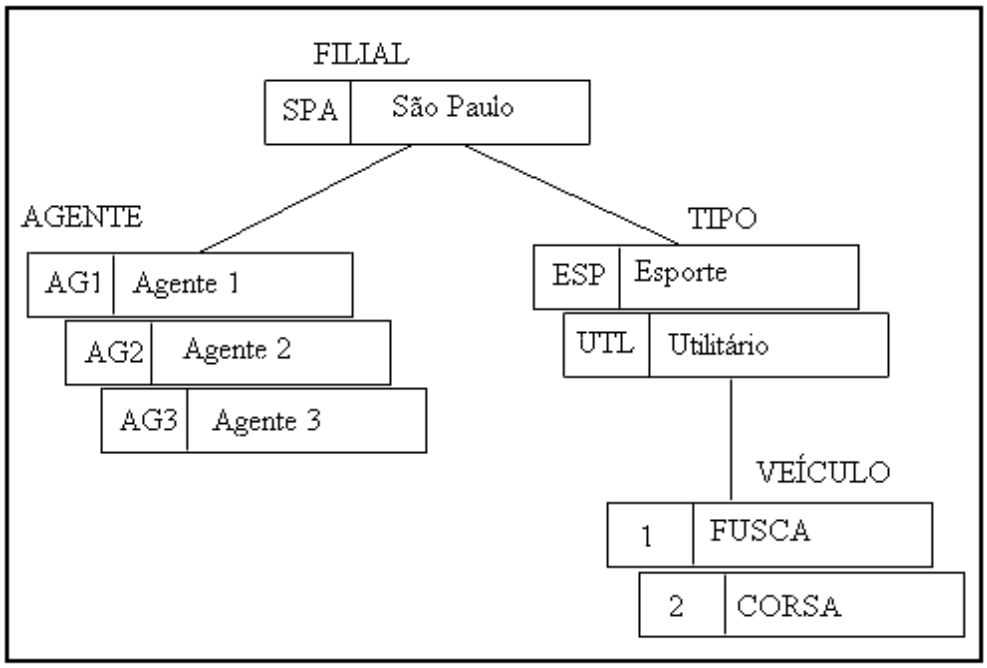

Figura 2.3 - Representação Hierárquica de Registros (Adaptado de KER94).

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.

Complementos para o aqui exposto podem ser encontrados em [KOR97], [KHO94], [DAT90] e [KER94].

2.3.2 O Modelo de Rede

Segundo [DAT90], o banco de dados de rede pode ser considerado uma forma ampliada do banco de dados hierárquico. Enquanto nos bancos hierárquicos um registro-filho tem exatamente um pai, no banco de dados de rede, um registro-filho pode ter quantos registros pais quiser, inclusive nenhum.

Conforme [KER94], um banco de dados de rede consiste em uma visão de grafo ou malha de ligações um para muitos entre os registros. Um tipo de registro pode estar envolvido em várias relacionamentos e pode ter diversos ascendentes e descendentes.

O trabalho de [KHO94] também considera o banco de rede uma 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 pode ser filho com vários registros pais, rompendo 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.

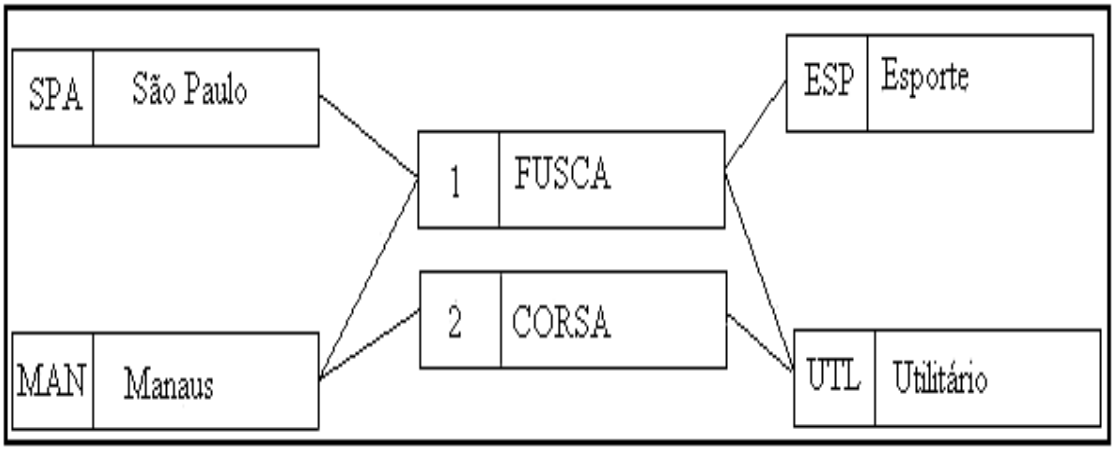

Figura 2.4 - Representação em Rede de Registros (Adaptado de KER94).

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 adicionando a ele campos de ponteiros para os registros 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.

Complementos sobre banco de dados de rede podem ser encontrados em [KOR97], [KHO94], [DAT90] e [KER94].

2.3.3 O Modelo Relacional

O modelo de dados relacional foi criado por E.F. Codd e hoje domina o mercado [RUM94]. Baseia-se em um único conceito: a tabela (figura 2.5). [RUM94] define um SGBD Relacional como um programa de computador responsável pelo gerenciamento dessas tabelas. Segundo [KER94] 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. Já [DAT90] define um banco de dados relacional como um banco que só pode ser percebido pelo usuário como tabelas e nada além de tabelas. [KOR97] define banco de dados relacional como uma coleção de tabelas, cada qual identificada por um nome único.

Um SGBD Relacional possui três elementos principais:

a) dados, que são representados por tabelas;

b) operadores para a manipulação dos dados;

c) regras de integridade das tabelas.

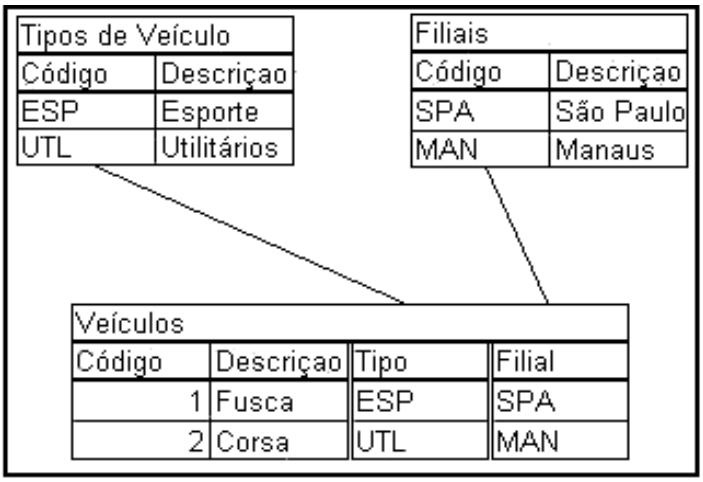

Figura 2.5 - Um Exemplo de Banco de Dados Relacional

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.

Os bancos 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.

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.

Maiores detalhes sobre banco de dados relacionais podem ser encontrados em [RUM94], [KOR97], [KHO94], [DAT90] e [KER94].

2.3.4 O Modelo Orientado a Objeto

O modelo Orientado a Objeto ou Modelo Relacional Avançado [RUM94] 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 [KHO94]. O estudo de banco de dados orientado a objeto é o enfoque principal desse trabalho e o capítulo 3 é totalmente dedicado a eles.

3 BANCO DE DADOS ORIENTADO A OBJETO

Os bancos de dados orientados a objeto são fruto da união de duas tecnologias: Sistemas Gerenciadores de Banco de Dados e Orientação a Objetos [KER94]. 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).

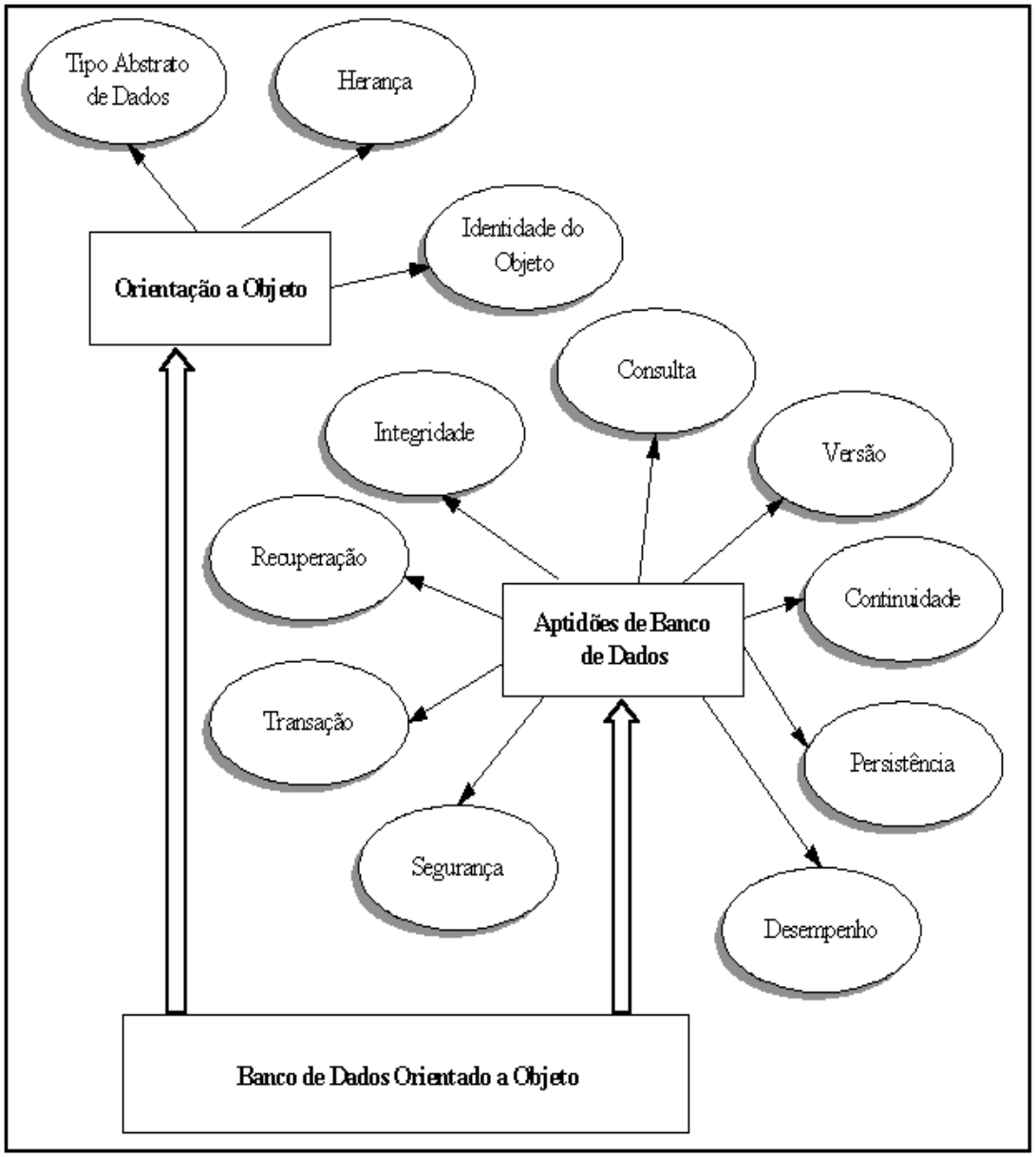

Figura 3.1 - Banco de Dados Orientado a Objeto (Fonte: KHO94).

Utilizando os conceitos de orientação a objetos os usuários de banco de dados podem manter os detalhes de implementação de seus módulos encapsulados nos Tipos Abstratos de Dados, compartilhar referências a outros objetos através do Identificador do Objeto bem como utilizar módulos já existentes para expandir seus sistemas com a utilização da Herança. Através das aptidões dos banco de dados os usuários possuem garantia durante transações concorrentes, coerência dos dados, obtêm o estado em que os objetos se encontram, entre outras.

O trabalho de [KHO94], apesar de contestar a existência de uma definição única para a própria "orientação a objeto", define Banco de Dados Orientado a Objeto (BDOO) como uma extensão dos conceitos de orientação a objetos e aptidões de bancos de dados, onde:

a) a Orientação a Objetos é igual a união dos conceitos de tipagem de dados abstratos, herança e identidade do objeto;

b) as Aptidões dos Bancos de Dados são a continuidade, concomitância, transações, recuperação, filtragem, atualização, integridade, segurança e o desempenho.

Segundo [RAO94], define-se um BDOO como 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 [GRE98], 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 relacionais, 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 [GRE98], é a capacidade que o banco apresenta de modelar não só os dados de estruturas complexas mas também seu comportamento.

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

a) são os modelos de dados mais adequados para o 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 primeira 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) existentes, principalmente através da implementação de características como herança e classe 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 [MAR95], os BDOO surgiram, em primeiro momento, da necessidade de se sustentar a programação orientada a objetos. Programas Smalltalk e C++ precisavam de um repositório para o que chamavam de dados persistentes, ou seja, dados que prevaleciam mesmo depois de encerrado determinado processo. Porém, as pesquisas sobre BDOO foram realmente estimuladas 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 o tipo de engenharia, a engenharia de software também possui fases bem 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 computador (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. Dados para um banco de multimídia surgem de correios eletrônicos de áudio, aplicações gráficas, entre outras;

e) sistema de informação para escritórios (OIS – *Office Information Systems*): Segundo [KHO94] um escritório inteligente incorpora as características de automação de escritório e documentação por imagem. Por natureza o modelo de escritório inteligente é orientado a objeto. Cada objeto do escritório pertence a uma classe. Essas classes podem ser especializadas ou generalizadas. Os objetos podem se referenciar a outros objetos. Identificadores de objetos surgem para identificar unicamente as instâncias dos objetos. O papel do banco de dados nesse caso é armazenar esses objetos e permitir consultas sobre todos os componentes do escritório, como memorandos, documentos, agendas, etc.

Os SGBD “convencionais” (Rede, Hierárquico e Relacional) não estão preparados para as necessidades dessas novas aplicações. Como tais SGDB 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:

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 pela linguagens de programação orientadas a objeto 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 efetuando uma alteração. Dessa forma, é inadmissível que os dados permaneçam 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 banco de dados e linguagens de programação podem suportar modelos de dados e paradigmas de objetos, a impedância desigual requer um grande esforço para mapear os modelos de objetos suportados pelas linguagens de programação em modelos de dados nos bancos 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.

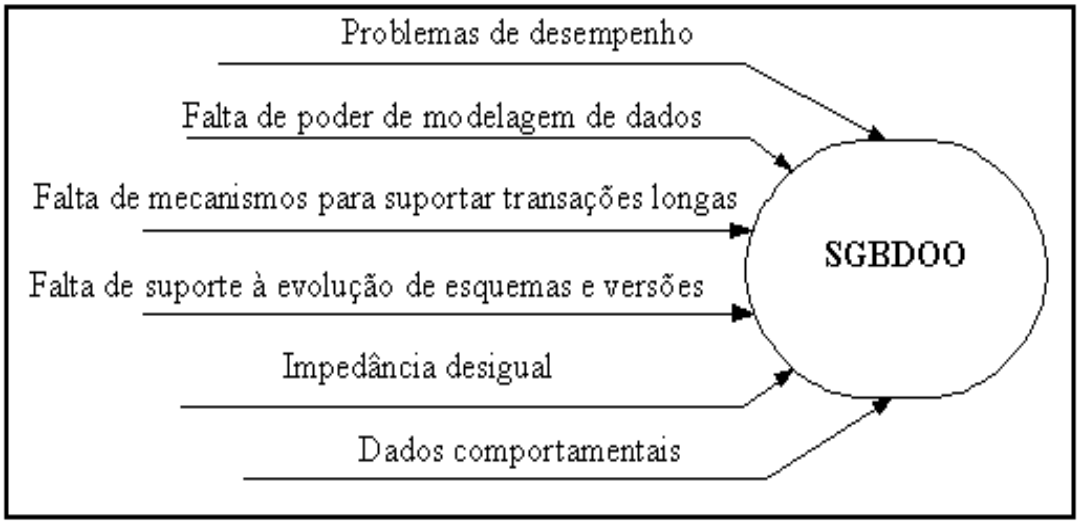

Figura 3.2 - Motivos do Surgimento dos BDOO

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 dados ou, melhor dizendo, 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.

Segundo [SAL92] as principais características de um SGBDOO, além daquelas de um SGBD convencional, são: objetos complexos, identidade de objetos, encapsulamento, tipos

e/ou classes, herança, "*late binding*" ou acoplamento tardio, extensibilidade e completude computacional.

3.2.1 Objetos para Banco de Dados

As abstrações da representação e das operações de um objeto são suportadas no modelo de dados orientado a objetos [DIA96]. Para tal, são incorporadas às características de objetos das linguagens de programação orientadas a objeto as noções de estruturas de dados e de 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 Objeto

Nas LPOO a identidade do objeto (OID do inglês *Object Identify*) corresponde, em geral, ao endereço físico de sua instância. Segundo [KOR97], em um SGBDOO os objetos geralmente correspondem a entidades da empresa que está sendo modelada pelo banco. Estas entidades mantém 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, visto que, qualquer atualização em valores dos atributos que um objeto vier a sofrer, não causará impacto nos objetos que o referenciam, pois seu identificador nunca é alterado.

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). Segundo

[MAC95] o OID é um identificador global lógico sem igual, independente do estado ou endereço do objeto. 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 a primária de um BDR) porque não é um valor de dados mutável;

c) um *register ID* (como o *ROWID* do *Oracle*) porque não é uma coluna lógica.

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 na base de dados que o identifica perante os demais objetos de sua classe, independente 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

Objetos complexos ocorrem de forma corriqueira em um BDOO. Basta alguma restrição de integridade referencial, que já está criada uma estrutura de objetos complexos. Os SGBDOO devem oferecer mecanismos para se recuperar por completo um objeto complexo do dispositivo de armazenamento para a memória. Também deve ficar a cargo do SGBDOO a função de reajustar, se necessário, os ponteiros internos dos objetos complexos quando de sua manipulação entre o disco e a memória, mantendo íntegro seus relacionamentos [RUD93].

Existem, basicamente, dois tipos de objetos complexos em BDOO:

a) objetos embutidos: os 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 a 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 a 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 [OBJ99]. 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 [OBJ99] as “Classes de Coleção” ou “Classes de Sistema” são um 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.

3.2.3 Extensibilidade e Completude Computacional

Estas são duas características marcantes para construção de um BDOO, conforme [GRE98] e [DIA96]. A primeira garante que os tipos oferecidos pelo sistemas permitam a construção de novos tipos e que não exista distinção entre os “tipos do sistema” e os novos

tipos definidos pelos usuários. A segunda implica que a linguagem de manipulação de objetos de um BDOO pode exprimir qualquer função computacional.

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 coerência dos objetos.

3.3.1 Transações

Em resumo transação é um programa ou uma seqüência de ações que lê ou grava objetos persistentes e mantém coerente o banco de dados [KHO94]. Para tal, uma transação deve atender as propriedades ACID (atomicidade, coerência, isolamento e durabilidade). Atomicidade significa que o programa ou a seqüências de ações é totalmente executada 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: uma transação pode não envolver somente objetos persistentes e a execução de um *Rollback* pelo SGBDOO não implica na restauração do estado original dos objetos não-persistentes. Esta é uma situação onde o programador deve saber quais são e 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 [MAC98], duas grandes diferenças arquitetônicas entre os BDR e os 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 [KHO94], é utilizado para resolver os problemas das transações demoradas. Ele compreende, 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 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 [KHO94], 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 suas subclasses;

b) bloqueios de objeto complexo: o problema dos objetos complexos consiste no fato de que um objeto complexo pode ter alguns de seus subobjetos 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 classe 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 subárvore parcial criada no nó bloqueado é implicitamente bloqueado no mesmo modo do objeto-pai.

3.3.3 Recuperação

Conforme [KHO94] 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 atualizações 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 [KHO94], o versionamento 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 versionamento 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.

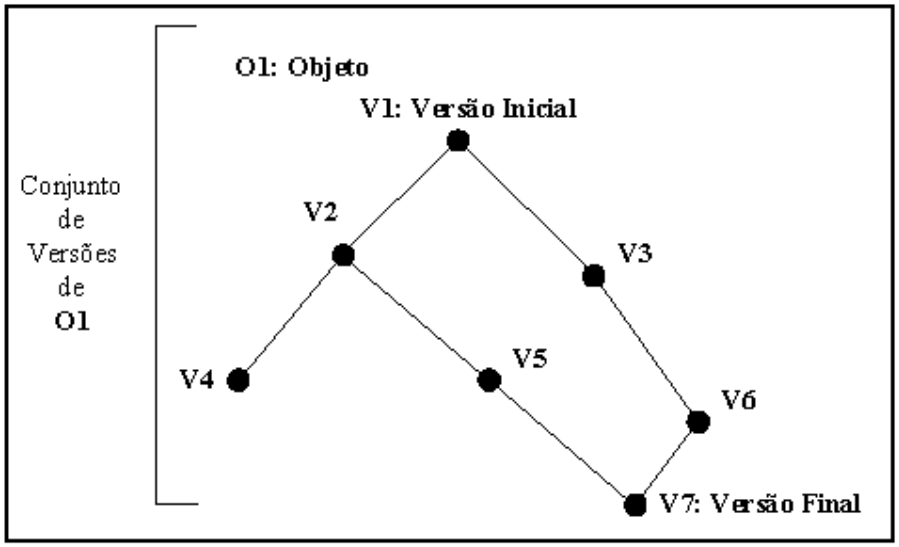

Figura 3.4 – Versionamento (Adaptado de KHO94).

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

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.

A maioria 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 uma ou mais chaves para as tabelas, com 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 do OID. Porém, para identificação exclusiva em nível lógico das instâncias de uma classe os BDOO permitem a definição de um ou mais atributos desta classe como sendo chave para a mesma. Nos BDOO as chaves também otimizam os processos de consultas.

Dessa forma, as chaves nos BDOO possuem duas funções [KHO94]:

a) Otimizar os processos de consulta;
b) Servir de restrição para ambigüidade lógica de instâncias.

3.3.5.2 Restrição de Integridade Referencial

Os objetos de um banco de dados não vivem isolados. Normalmente, eles estão relacionados com outros objetos no banco de dados [OBJ99]. Segundo [RUD93], o relacionamento entre objetos é um componente essencial no paradigma do modelo OO. Através dele pode-se criar objetos inter-relacionados.

Da existência das relações advêm a necessidade da integridade referencial. Conforme [RUD93] a integridade referencial tem por objetivo assegurar que objetos não contenham relacionamentos com objetos que não existam mais no banco de dados. Para [KHO94], 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 relaciona 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 [KHO94]. Segundo [RUD93] a habilidade e eficiência dos relacionamentos diretos por meio de OID são uma das principais melhorias do modelo de dados orientado a objeto em relação ao modelo de dados relacional.

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 especificada para o tipo definido para o atributo. Valores *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 Restrição de Precondições e Pós-Condições de métodos

Conforme [KHO94], 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 de pós-condições permitem a definição de outras restrições que devem ser cumpridas após um método ser executado.

3.3.5.6 Restrição de Disjunção

A restrição de disjunção garante que duas classes não podem ter um elemento em comum. Por exemplo, se as classes "Funcionário Próprio" e "Funcionário Terceirizado" são disjuntas, não pode existir um objeto "Funcionário" que pertença a ambas as classes.

3.3.5.7 Restrição 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 instâncias 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 porque em alguns casos é desejável criar instâncias da 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 [RAO94], 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 [OBJ99], o aspecto mais importante deste mecanismo é que as aplicações não precisam mais se preocupar 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. Está conversão é de responsabilidade do SGBDOO.

Segundo as colocações de [KHO94] e [RAO94], a gerência de persistência é uma das principais características da LPOO para banco de dados e deve ser ortogonal ao tipo de objeto, ou seja, todo o objeto deve poder persistir.

3.3.7 Consultas

Segundo [ROA94] 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 [KER97]. 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 mensagem 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 difíceis [CEL97]. Elas devem incluir tanto o modelo de passagem de mensagem de um objeto por vez quanto o modelo de mensagens por conjunto.

3.4 Outras Considerações sobre SGBDOO

Os SGBDOO foram um dos grandes aparecimentos na área das Ciências da Computação nos anos 80. Os objetos eram a "onda" do momento e todos estavam interessados neles [CEL97]. Porém os primeiros SGBDOO careciam de funções de gerenciamento de dados como, por exemplo, mecanismos de back-up e recuperação. Os profissionais da computação também estavam meio confusos com relação ao aparecimento dos BDOO. Isso fez com que a maioria dos SGBDOO não passassem de experiências de laboratórios. Nesse ponto, os conhecimentos "populares" sobre BDOO estacionaram e todos ficaram com uma imagem congelada de que os BDOO não passavam de experiências de laboratório e serviam apenas para alguns nichos de mercado.

As considerações abaixo tem por objetivo levantar alguns pontos relevantes da tecnologia de BDOO através das quais percebe-se que os BDOO não são mais simples experiências de laboratórios e têm obtido grande amadurecimento.

3.4.1 Os BDOO não tem Embasamento Teórico

Os BDOO não possuem nenhum teoria por trás de seus conceitos, diferente dos bancos de dados hierárquico (apoiados na estrutura de "árvore"), dos bancos de dados em rede (baseados no "grafo") e dos bancos de dados relacionais (apoiados na "matemática dos conjuntos") [GRE98]. Porém, conforme [CEL97], nem tudo que não possui um embasamento teórico sólido é inútil. O cálculo numérico é um bom exemplo: produz resultados corretos e úteis sem nenhuma base teórica.

3.4.2 Desempenho dos BDOO

É de conhecimento geral que o longo período transcorrido entre a concepção do modelo de dados relacional (1970) até o seu uso prático (1985) deve-se aos sensíveis problemas de desempenho que este modelo apresenta [MAC95]. Ainda hoje, aplicações que envolvem dados altamente complexos sofrem com o desempenho dos BDR.

Em contrapartida, os BDOO tem desempenho consideravelmente superior em relação aos BDR, especialmente em aplicações que envolvem alta conectividade dos dados. Isto ocorre principalmente devido a dois fatores:

a) OID como referência: as relações dos objetos agregados no banco de dados são referenciadas por OID e automaticamente convertidas pelo sistema na forma endereços físicos em disco [MAC95];

b) *Swizzling* de Ponteiro: Quando um objeto armazenado em disco é trazido para a memória seu endereço em memória é automaticamente "ligado" (*swizzled*) com o endereço em disco provendo acesso rápido a toda sua estrutura.

3.4.3 Os BDOO são para Aplicações Específicas

Isso é quase verdade. De fato, conceitualmente, todo e qualquer SGBD é projetado para atender determinadas necessidades. Sempre deve-se ter em mente que jamais os dados de determinada aplicação devem se "adaptar" ao banco de dados. Na verdade, para cada determinado segmento existe um SGBD que atende às necessidades da aplicação em questão. O mercado de BDR vangloria-se dos seus 90% de margem financeira no mercado de banco de dados, mas se esquece do detalhe de que apenas 12% de todo o processamento de dados empresariais é feito em produtos relacionais [CEL97].

Inúmeras são as aplicações que estão em ascensão e são intimamente ligadas aos SGBDOO. Entre elas pode-se citar os videogames, as aplicações multimídia e, principalmente, a Internet que apoiada no forte fato do Java ser uma linguagem orientada a objetos torna-se intimamente ligada aos SGBDOO.

3.4.4 Os BDOO Evitam a Redundância de Código

Através das regras de normalização dos BDR evita-se a redundância de dados e suas possíveis conseqüências. Porém, a normalização não impede a redundância de código. Os SGBDOO utilizam a herança para diminuir a redundância de métodos. Dessa forma, através da herança e conseqüente reutilização de código, os modelos OO diminuem tanto a quantidade de código redundante quando de dados, diminuindo os custos de manutenção e desenvolvimento de software [MAR95].

3.4.5 Os BDOO não Possuem uma Linguagem Padrão

Segundo [CEL97], o SQL é a única linguagem padrão para banco de dados e é relacional. Os BDOO não possuem tal "linguagem padrão". Isso obriga os desenvolvedores a

trabalharem com linguagens proprietárias que as vezes são altamente complexas. Daí ressalta-se o argumento de que os BDOO não são bons para o desenvolvimento sério. Isso é uma verdade convencionada e pode ser "quebrada" a qualquer momento. Um grupo de fabricantes de BDOO denominado de *Object Database Management Group* (ODMG) produziu, em 1993, um padrão linguagem de consulta para BDOO denominada de *Object Query Language* (OQL). Vários fabricantes de BDOO já estão incorporando o suporte ao OQL em seus produtos [CEL97]. Dessa forma a existência de portabilidade entre BDOO por meio de uma linguagem padrão é apenas uma questão de tempo.

3.4.6 Os BDOO já são Escaláveis

No início das pesquisas sobre BDOO a escalabilidade era um dos grandes problemas dos protótipos. A maioria podia rodar apenas na memória principal e se a máquina apresentasse alguma falha e viesse a cair, cairiam junto. Hoje os BDOO são "bancos de dados de verdade" e apresentam características de segurança, back-up e recuperação como qualquer BDR existente.

Segundo [HUM95], a essência da escalabilidade consiste em um sistema de software poder ser desintegrado em componentes menores, estes componentes serem separadamente desenvolvidos e o sistema totalmente reintegrado. Em SGBD, a escalabilidade contempla as necessidades de gerência e manutenabilidade do sistema como um todo, abrangendo áreas que dizem respeito desde o gerenciamento de transações até a persistência de dados.

4 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 [INT99]. Em relação às "antigas aplicações para banco de dados", as atuais aplicações corporativas tem novas exigências [IPS99b]:

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 [IPS99b]:

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 4.1).

Figura 4.1 - 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.

4.1 A InterSystems Corporation

A InterSystems, fornecedora do Caché, possui tecnologia presente em 28 países utilizada por cerca de 7.000 instituições. São mais de 30.000 instalações com mais de 2.000.000 de usuários licenciados. Segundo pesquisa realizada pelo *International Data Corporation* (IDC) em 1998, empresa de pesquisas sediada em Framinghamm - Massachusetts, a InterSystems é um dos 10 maiores distribuidores de bancos de dados pós-relacionais (bancos de dados com modelos de dados que surgiram após o modelo de dados relacional [KHO94]) para sistemas UNIX no mundo e um dos fornecedores que mais crescem na área de banco de dados para UNIX.

Dentre os clientes InterSystems encontram-se algumas das mais prestigiosas companhias do mundo como o Chase Manhattan Bank**,** a Shell International Petroleum, a Justiça Federal do Brasil, a Payco, entre outras.

4.2 O Modelo de Dados Multidimensional

O modelo de dados multidimensional torna a modelagem de dados muito mais simples através da possibilidade de se representar o mundo real como ele realmente é, sem confiná-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 modelá-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 modelá-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 [INT99].

As matrizes multidimensionais são a melhor forma de se representar estruturas complexas [INT99]. A fim de obter o 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.

4.3 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 4.2). 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.

4.3.1 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 obtêm o menor tempo de acesso possível e a otimização máxima de seus processos. 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 [INT97a] [INT97b].

4.3.2 O Acesso SQL

Indiscutivelmente, as atuais aplicações relacionais continuarão por um longo tempo a representar uma 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, o que garante, segundo testes comparativos, um desempenho até 5 vezes mais rápido que as tradicionais aplicações relacionais [IPS99b] [INT99].

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 similar 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 um migração gradual de aplicativos Oracle para Caché.

4.3.3 O Acesso 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 processamento de transações sob os mais exigentes requisitos, como estruturas de dados complexas e processamento 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 das mais diversas aplicações, o Caché permite a definição, além dos 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".

4.3.4 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.

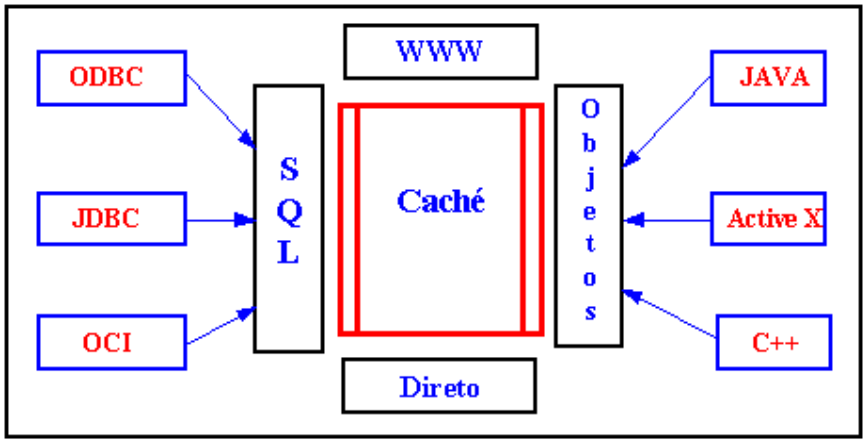

Figura 4.2 - O Caché e suas Diversas Formas de Acesso.

4.4 O Caché Objects

O Caché Objects é o componente (figura 4.3) 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 [INT98b] 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.

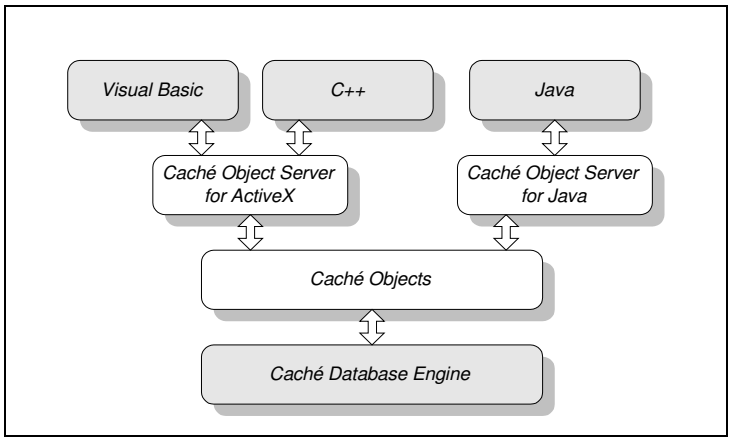

Figura 4.3 - Visualização do Caché Objects (Fonta: INT98b).

4.4.1 Estrutura do Caché Objects

O Caché Objects é um sistema (figura 4.4) 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 (Anexo 1) 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 (Anexo 2).

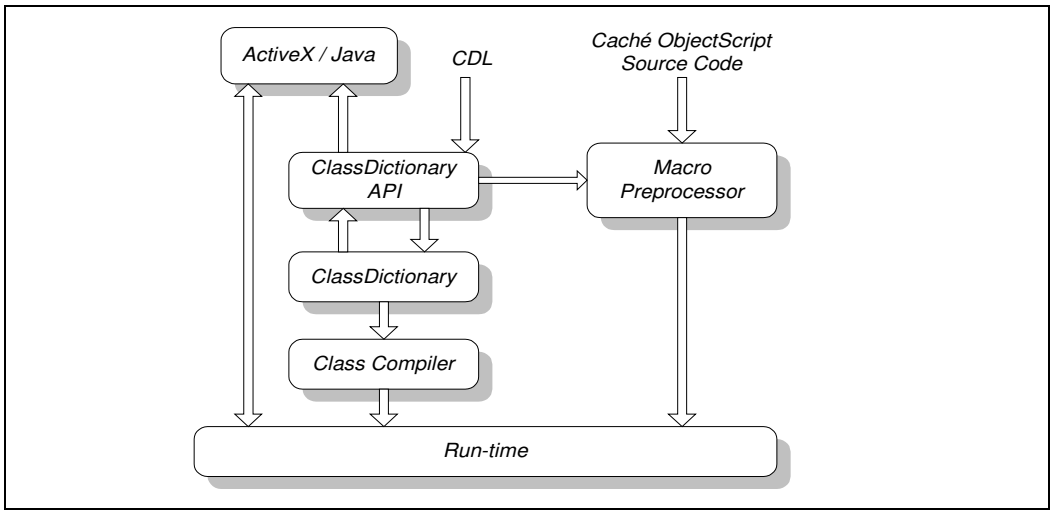

Figura 4.4 - Os Subsistemas do Caché Objects (Fonte: INT98b).

4.4.2 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 dividas em vários tipos (figura 4.5). 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 dividem-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.

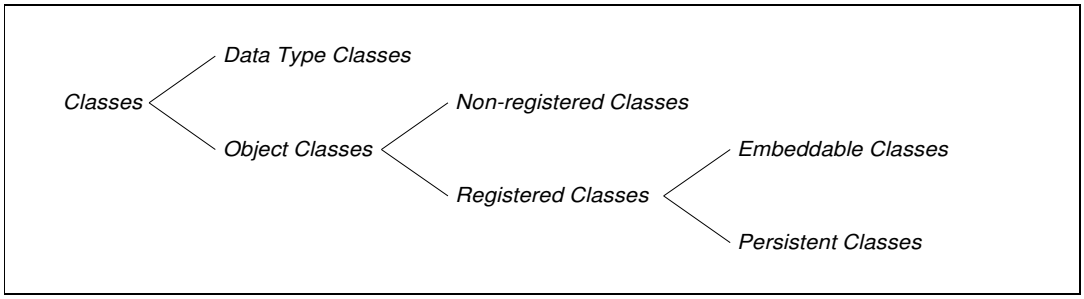

Figura 4.5 - Os Tipos de Classes do Caché Objects (Fonte: INT98a).

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 (Anexo 3);

b) através do *Caché Class Description Language* (CDL), uma linguagem de definição de classes apresentada em um arquivo texto no formato ASCII (Anexo 4).

4.4.3 Classes no Caché Objects

As classes do Caché Objects são formadas por um conjunto de parâmetros/palavras-chave, métodos e propriedades (figura 4.6). Os parâmetros/palavras-chave de uma classe definem toda a parte "comportamental" da classe quando de sua compilação. 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, sua(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 (relacionamentos) 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 objetos da classe e são implementados em COS e SQL. Os métodos de consulta são utilizados para processos de recuperação, análise e consulta de instâncias de objetos.

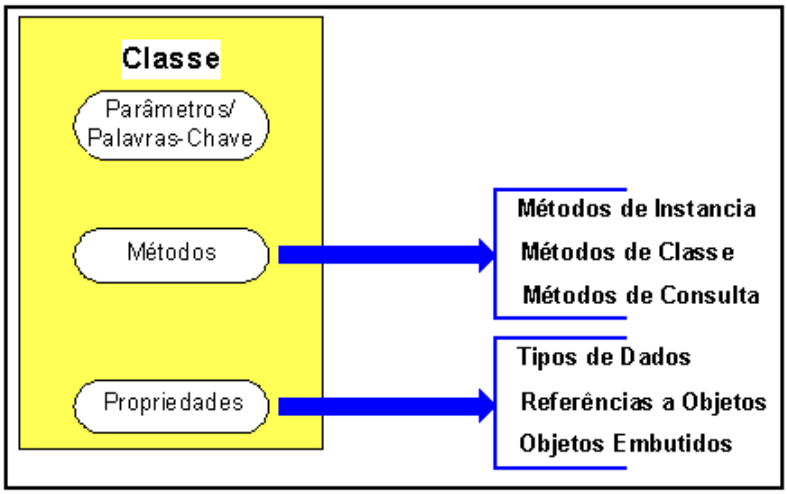

Figura 4.6 - Conteúdo de uma Classe do Caché Objects

5 CRITÉRIOS DE AVALIAÇÃO PARA SGDB

Os critérios de avaliação aqui apresentados tem por objetivo auxiliar o desenvolvedor na escolha do SGDB que mais lhe facilite e otimize o processo de desenvolvimento de sistemas aplicativos. Esses critérios estão fundamentados teoricamente nos critérios de avaliação da norma NBR 13596/1996 - Qualidade de Software Produto [ABN96] - e nos critérios de avaliação de SGDBOO descritos no relatório *Object-Oriented Database Management Systems* [RUD93].

Para proceder a seleção dos critérios de avaliação definidos nesse trabalho observaram-se os seguintes aspectos:

a) problemas e facilidades encontrados durante o desenvolvimento do protótipo apresentado como parte integrante deste trabalho e descrito no capítulo 6;

b) características diversas da norma NBR 13596, que não possuem impacto direto no processo de desenvolvimento da aplicação mas que agem de forma secundária nesse contexto, como por exemplo: gerência de segurança, controle de concorrência e recuperação, entre outros;

c) características diversas dos critérios de avaliação de SGBDOO descritos em [RUD93] que também podem ser aplicados a todo tipo de SGBD, independente do seu modelo de dados;

d) características mensuráveis de análise em decorrência do levantamento bibliográfico apresentado neste trabalho.

5.1 A Norma NBR 13596

A norma NBR 13596, publicada em Agosto de 1996, é a versão brasileira da norma internacional ISO/IEC 9126, que foi publicada em 1991 pela ISO (*International Organization For Standardization*) em conjunto com a IEC (*International Electrotechnical Commission*). A NBR 13596 consiste de um conjunto de características que devem ser verificadas em um software para que ele seja considerado um software de qualidade [BAR96].

A NBR 13596 é apresentada na forma de 6 grandes grupos de características, cada um divido em subcaracterísticas [BAR96]:

a) funcionalidade: Consiste na capacidade do software em satisfazer quaisquer funções adequando estados e funções quando usado sob condições específicas. Em resumo, tem por objetivo verificar se o software satisfaz as necessidades. São subcaracterísticas da funcionalidade: adequação, acurácia, interoperabilidade, conformidade e segurança dos dados;

b) confiabilidade: Consiste na capacidade do software em manter seu desempenho quando utilizado sob condições específicas. Em resumo, tem por objetivo verificar se o software é imune a falhas. São subcaracterísticas da confiabilidade: maturidade, tolerância a falhas e recuperabilidade;

c) usabilidade: Consiste na capacidade do software em ser fácil de usar e satisfazer o usuário quando utilizado sob condições específicas. Em resumo, tem por objetivo verificar se o software é fácil de usar. São subcaracterísticas da usabilidade: intelegibilidade, apreensibilidade e operacionalidade;

d) eficiência: Consiste nos recursos utilizados pelo software para alcançar a performance requerida sob condições específicas. Em resumo, tem por objetivo verificar se o software é rápido. São subcaracterísticas da eficiência: tempo e recursos;

e) manutenibilidade: Consiste nos recursos necessários para fazer modificações específicas no software. Em resumo, tem por objetivo verificar se o software é fácil de ser modificado. São subcaracterísticas da manutenibilidade: analisabilidade, modificabilidade, estabilidade e testabilidade;

f) portabilidade: Consiste na capacidade de se transferir o software para outros ambientes (organizacional, hardware e/ou software). Em resumo, tem por objetivo verificar se o software é fácil de ser utilizado em outro ambiente. São subcaracterísticas da portabilidade: adaptabilidade, capacidade de instalação, conformidade e capacidade de substituição.

5.2 O Relatório *Object-Oriented Database Management Systems*

O relatório *Object-Oriented Database Management Systems* tem por objetivo disponibilizar a compreensão de assuntos pertinentes a SGBDOO. Espera-se que os critérios de avaliação apontados em [RUD93] sirvam como primeira instância de avaliação do processo de seleção de um SGBDOO para utilização em determinada aplicação. [RUD93] salienta que não existe um único SGBDOO melhor em todas as situações de avaliação descritas em seu relatório.

Segundo [RUD93], a avaliação de um SGBDOO deve contemplar quatro áreas:

a) funcionalidade: uma análise das capacidade funcionais do SGBD deve ser realizada para verificar se o SGBD provê capacidades suficientes para satisfazer as necessidades atuais e futuras de determinado software aplicativo. Dentre as capacidades funcionais, incluem-se as características básicas dos SGBD (como controle de concorrência, recuperação de dados, integridade, gerência de transações, segurança entre outras) assim como características da orientação a objetos (como herança, objetos persistentes, versionamento entre outras);

b) usabilidade: a usabilidade trata do desenvolvimento da aplicação e do processo de manutenção. Dentre os pontos de avaliação da usabilidade incluem-se as ferramentas de desenvolvimento (ferramentas e linguagem de definição dos dados, ferramentas de administração dos dados, etc.) bem como a facilidade com que aplicações de banco de dados podem ser desenvolvidas e mantidas (*wizards* para construção de cadastros e consultas, etc.). Outros assuntos que englobam a usabilidade também são a maturidade do produto e o suporte do vendedor;

c) plataforma: a plataforma é com certeza o critério de avaliação mais facilmente mensurável. A plataforma visa avaliar se um SGBD está ou não está disponível no hardware e no sistema operacional de objetivo da aplicação;

d) desempenho: sem dúvida o desempenho pode ser considerado o critério de avaliação mais importante. Características de desempenho envolvem um levantamento sobre o número de usuários interativos, a taxa de atualização e acesso do banco de dados, o tamanho do banco de dados, as configurações de hardware da rede, entre outras.

5.3 Os Critérios de Avaliação

Para facilitar a aplicação dos critérios de avaliação, os mesmos foram divididos em dois grupos:

a) critérios de avaliação do software aplicativo: os critérios de avaliação do software aplicativo tem por objetivo levantar pontos relevantes no que diz respeito às características do software que será desenvolvido, como por exemplo a natureza dos dados;

b) critérios de avaliação do software SGBD: os critérios de avaliação do SGBD tem por objetivo levantar as características básicas que o SGBD deve atender para suprir as necessidades do software que será desenvolvido, como por exemplo: funcionalidade (controle de concorrência, gerência de recuperação, etc.), usabilidade (linguagem de DDL e DML, linguagem de consulta, etc.), entre outras.

5.3.1 Critérios de Avaliação do Software Aplicativo

Para avaliação do software aplicativo sugere-se a aplicação dos critérios descritos a seguir.

5.3.1.1 Natureza dos Dados

A avaliação da natureza dos dados tem por objetivo levantar as características dos dados que serão manipulados pela aplicação. Consiste em avaliar a natureza dos dados verificar se:

a) a aplicação possui manipulação de dados de multimídia (Som, Imagem, objetos BLOB – *Binary Large Objects*): Se a resposta a esta pergunta for sim, o SGBD mais indicado para aplicação é o SGBDOO, pois SGBDR convencionais não possuem suporte à manipulação de dados multimídia;

b) é possível transcrever os dados do mundo real da aplicação para a forma de tabelas sem tornar o modelo de dados complexo: O objetivo deste critério é avaliar se as tabelas geradas a partir dos dados do mundo real da aplicação não resultam um complexo modelo de dados relacional, formado por um emaranhado de tabelas, inúmeras relações dependentes onde, qualquer dado que se queira extrair da base de dados, necessita de uma complexa expressão SQL na forma de um combinado de

Joins entre tabelas. Com certeza não é viável trabalhar com um modelo de dados relacional complexo. Nesses casos, a melhor opção para modelagem dos dados reais da aplicação são os SGBDOO, pois a tecnologia OO oferece inúmeros recursos (herança, classes, polimorfismo, entre outras) para manipulação de estruturas de dados complexas;

c) a estrutura dos dados da aplicação é estável ou sofre constantes mudanças: Dados que sofrem constantes mudanças estruturais são mais indicados para SGBDOO, pois a mudança do esquema de dados de um SGBDR é uma tarefa árdua que demanda muito tempo de trabalho. Já, nos SGBDOO, esta tarefa é amenizada em função de facilidades do modelo OO como o OID, a herança e o versionamento;

d) os dados da aplicação terão que ser acessados por mais de um usuário ao mesmo tempo para processos de gravação: Os únicos SGBD que suportam acesso concorrente de mais de um usuário ao mesmo dado simultaneamente para processos de gravação são os SGBDOO. Caso a aplicação não necessite de processos de gravação simultâneos sobre os mesmos dados, tanto SGBDR quanto SGBDOO podem ser utilizados;

e) os dados da aplicação são comportamentais: Dados comportamentais só podem ser gerenciados por SGBD que implementem o polimorfismo.

5.3.1.2 Natureza das Transações

A avaliação da natureza das transações tem por objetivo levantar as características das transações realizadas pela aplicação. Consiste em avaliar a natureza das transações verificar se:

a) as transações realizadas pela aplicação são curtas e direcionadas a alguns registros: Se a resposta a esta pergunta for não o SGBD mais indicado é o SGBDOO, pois transações longas não possuem bom desempenho quando aplicadas em SGBDR;

b) há necessidade de transações em cooperação de usuários: Somente os SGBDOO tem suporte a transações de cooperação, onde vários usuários acessam ao mesmo tempo um conjunto de dados para realização de uma tarefa comum.

5.3.1.3 Natureza da Concorrência

A avaliação da natureza da concorrência tem por objetivo levantar os controles de concorrência que serão necessários para o bom funcionamento da aplicação. Consiste em avaliar a natureza das transações verificar se:

a) O controle pessimista de concorrência (bloqueio exclusivo) é suficiente para atender a aplicação: O bloqueio exclusivo é o último nível de controle de concorrência dos SGBDR. Logo, se for necessário algum tipo de bloqueio que ultrapasse as características do bloqueio exclusivo, como por exemplo o bloqueio de hierarquia de classes ou o bloqueio de objetos complexos, é necessário a utilização de um SGBDOO. Se não houver esta necessidade, pode-se utilizar um SGBDR.

5.3.1.4 Processo de Desenvolvimento da Aplicação

A avaliação do processo de desenvolvimento da aplicação tem por objetivo levantar qual é o processo de desenvolvimento da aplicação. Consiste em avaliar o processo de desenvolvimento da aplicação verificar se:

a) o processo de desenvolvimento da aplicação é orientado a objeto: Aplicações com processos de desenvolvimento OO ganham em produtividade quando têm seus dados armazenados em SGBDOO, pois não precisam ter seu modelo de dados "transformado" num modelo de dados compatível com o SGBD. Em contrapartida, se o processo de desenvolvimento da aplicação for OO e o SGBD for relacional há a necessidade de "transformar" o modelo de dados do processo de desenvolvimento num modelo de dados relacional, na forma de tabelas, para adequá-lo ao SGBD. Esse processo, em geral, é moroso e complexo;

b) o processo de desenvolvimento da aplicação é estruturado: Aqui os papéis se invertem: processos de desenvolvimento estruturados tem seus dados mais facilmente modelados para SGBDR pois, a maioria dos modelos de dados resultantes da análise estruturada apresentam-se na forma de tabelas. Vale salientar que o trabalho de transformar o modelo de dados de um processo de desenvolvimento estruturado num modelo de dados para aplicação em um SGBDOO é bem menor que o trabalho de transformar o modelo de dados de um processo de desenvolvimento OO para aplicá-lo a um SGBDR.

5.3.2 Critérios de Avaliação do Software SGBD

Para avaliação do software de SGBD sugere-se a aplicação dos critérios de avaliação descritos a seguir.

5.3.2.1 Funcionalidade

A avaliação da funcionalidade consiste em verificar as características funcionais do SGBD. São atributos de análise da funcionalidade:

a) integridade: a avaliação das características funcionais de integridade do SGBD tem por objetivo verificar se o SGBD é integro. Consiste em avaliar a integridade do SGBD verificar se:

- o SGBD garante a integridade dos dados em atualizações simultâneas: O objetivo deste critério é verificar se o SGBD possui mecanismos para assegurar que atualizações simultâneas dos dados não ocasionem inconsistências na integridade física e lógica do modelo de dados;
- o SGBD garante a integridade dos dados durante os processos de recuperação: Este critério visa avaliar se o SGBD garante a integridade física e lógica dos dados durante processos de recuperação de problemas de aplicação, sistema operacional e/ou hardware;
- o SGBD apresenta suporte a mecanismos de *constraints* dos dados: *Constraints* são construções lógicas de condições específicas que sempre devem ser verdadeiras para garantir a integridade lógica do modelo de dados. O objetivo deste critério é avaliar se o SGBD possui mecanismos para a construção de *constraints*;

b) concorrência: a avaliação das características funcionais de concorrência do SGBD tem por objetivo verificar quais são os mecanismos de controle de concorrência que o SGBD oferece. Consiste em avaliar os mecanismos de controle de concorrência do SGBD verificar se:

- o(s) mecanismo(s) de controle de concorrência que o SGBD oferece atende(m) as necessidades da aplicação: O objetivo deste critério é avaliar se o(s) mecanismo(s) de controle de concorrência oferecido(s) pelo SGBD atende(m) as necessidades das situações de acesso concorrente que a aplicação apresentará;

- o SGBD provê o controle de concorrência baseado em múltiplas transações de leitura concorrentes mas só uma transação de escrita: O objetivo deste critério é avaliar se o SGBD provê mecanismos de controle de concorrência que permitam vários usuários lerem os dados ao mesmo tempo que um, e somente um, procede atualizações de gravação;
- o SGBD provê o mecanismo de controle baseado em múltiplas transações de escrita e leitura concorrentes: O objetivo deste critério é avaliar se o SGBD possui suporte a transações na forma de grupos de trabalho, onde mais de um usuário possui acesso simultâneo aos mesmos dados, tanto para leitura quanto para gravação;

c) recuperação: a avaliação das características funcionais de recuperação do SGBD tem por objetivo verificar quais são os mecanismos de suporte à recuperação que o SGBD oferece. Consiste em avaliar os mecanismos de recuperação do SGBD verificar se:

- o SGBD provê a recuperação de falhas de aplicação: O objetivo deste critério é avaliar se o SGBD possui mecanismos para recuperação de problemas de aplicação que venham, por exemplo, a abortar um transação;
- o SGBD provê a recuperação de falhas de sistema operacional: O objetivo deste critério é avaliar se o SGBD possui mecanismos para resolver problemas decorrentes de falhas no sistema operacional que venham, por exemplo, a tirar o SGBD do ar, deixando inúmeras transações incompletas e possíveis dados inconsistentes;
- o SGBD provê a recuperação de falhas de mídia: O objetivo deste critério é avaliar se o SGBD possui mecanismos para recuperação dos dados no caso de falhas de mídia como, por exemplo, *crash* de disco;

d) transações: a avaliação das características funcionais das transações do SGBD tem por objetivo verificar quais são os mecanismos de gerenciamento de transações que o SGBD oferece. Consiste em avaliar os mecanismos de gerência de transações do SGBD verificar se:

- as transações do SGBD podem acessar dados localizados em qualquer parte do SGBD, se este for distribuído: O objetivo deste critério é avaliar se o SGBD garante que qualquer transação acesse qualquer dado num SGBD distribuído;
- existe suporte a transações longas: O objetivo deste critério é avaliar se o SGBD possui mecanismos para suporte a transações longas;
- existe suporte a transações em grupo de usuários: O objetivo deste critério é avaliar se o SGBD possui mecanismos para suporte a transações em grupos de usuários;
- o SGBD permite a classificação das transações: O objetivo deste critério é avaliar se o SGBD permite classificar as transações com a intenção de antecipar a execução de determinadas transações;

e) *deadLocks:* a avaliação da característica funcional de *DeadLock* do SGBD consiste em verificar os seguintes pontos:

- o SGBD provê de mecanismos para descoberta de problemas de *DeadLock*: O objetivo deste critério é avaliar se SGBD possui mecanismos para descobrir possíveis problemas de *DeadLock*;
- se o SGBD possui mecanismos para descoberta de problemas de *DeadLock*, a forma como estes problemas são resolvidos atende as necessidades da aplicação: O objetivo deste critério é avaliar se a forma como o SGBD se propõem a resolver os problemas de *DeadLock* atende as necessidades da aplicação. Por exemplo: em uma aplicação que apresenta corriqueiramente transações longas, não é interessante que a forma como um SGBD resolva o problema do *DeadLock* seja abortar e reinicializar a última transação pois, se as transações envolvidas no problema forem longas, abortar uma das mesmas consiste numa considerável perda de tempo;

f) back-Up e *restore:* a avaliação da característica funcional de Back-Up e *Restore* do SGBD consiste em verificar os seguintes pontos:

- o SGBD apresenta mecanismos de Back-Up e *Restore*: O objetivo deste critério é avaliar se o SGBD apresenta mecanismos próprios para Back-Up e *Restore* dos dados;
- se o SGBD apresenta mecanismos próprios de Back-Up, o mesmo pode ser realizado com o SGBD On-Line: O objetivo deste critério é avaliar se é

possível realizar o processo de Back-Up dos dados com o SGBD On-Line ou se é necessário colocar o SGBD no modo *StandBy*;

g) indexação*:* a avaliação da característica funcional de indexação do SGBD consiste em verificar os seguintes pontos:

- o SGBD possui mecanismos de gerenciamento de índices para otimização de acessos: O objetivo deste critério é avaliar se o SGBD possui suporte a mecanismos para gerenciamento de índices;
- se o SGBD possui mecanismos de gerenciamento de índices, é permitido ao DBA selecionar qual tecnologia de gerenciamento de índices ele deseja utilizar no seu modelo de dados: O objetivo deste critério é avaliar se o DBA está habilitado a definir qual tecnologia de gerenciamento de índices (dentre as disponíveis no SGBD) ele quer que seja utilizada no seu modelo de dados;

h) segurança*:* a avaliação da característica funcional de segurança do SGBD consiste em verificar os seguintes pontos:

- o SGBD provê controle de acesso multi-nível para usuário: Este critério visa avaliar se o SGBD possui controle de segurança com base em níveis de acesso para usuários;
- o SGBD provê controle de acesso multi-nível para usuário/dados: Este critério visa avaliar se o SGBD possui controle de segurança com base em níveis de acesso para usuário e dados simultaneamente, onde define-se, além das características de acesso do usuário, que dados o usuário pode acessar.

5.3.2.2 Usabilidade

A avaliação da usabilidade do SGBD tem por objetivo levantar características ligadas aos processos de desenvolvimento de aplicações e manutenção do SGBD. São atributos de análise da usabilidade:

a) linguagem DML/DDL: este atributo tem por objetivo avaliar a usabilidade da linguagem DML/DDL do SGBD, verificando os seguintes pontos:

- quais são as linguagens de definição de dados (DDL) que o SGBD suporta: O objetivo deste critério é levantar quais são as linguagens DDL do SGBD;

− quais são as linguagens de manipulação de dados (DML) que o SGBD suporta: O objetivo deste critério é levantar quais são as linguagens DML do SGBD;

− existem linguagens DML/DDL no SGBD que são proprietárias (quais): O objetivo deste critério é verificar se existem linguagens, dentre as disponíveis no SGBD, que são proprietárias, a fim de se verificar pontos como a disponibilidade de documentação e treinamento sobre as mesmas;

− existem linguagens DML/DDL no SGBD que são complexas (quais): O objetivo deste critério é avaliar se existem linguagens DML/DDL no SGBD que são complexas a fim de, novamente, levantarem-se pontos como a disponibilidade de documentação e treinamento sobre as mesmas;

− as linguagens DML do SGBD garantem a completude computacional: Este critério tem por objetivo verificar se as linguagens DML do SGBD garantem o conceito de completude computacional;

− as linguagens DML/DDL do SGBD garantem a independência dos dados: O objetivo deste critério é avaliar se as linguagens DML/DDL do SGBD garantem a independência dos dados, mesmo quando ocorrem mudanças no esquema do modelo de dados do SGBD;

− as linguagens DML/DDL do SGBD possuem suporte à impedância desigual dos dados: O objetivo deste critério é avaliar se as linguagens DML/DDL do SGBD garantem que o problema da impedância desigual dos dados não existe, ou seja, tanto linguagem da aplicação quanto a linguagem DML/DDL do SGBD possuem a mesma forma de visualização dos dados;

− existem linguagens DML/DDL no SGBD que são OO ou apresentam características OO (quais): O objetivo deste critério é levantar quais são as características OO disponíveis tanto no SGBD quanto em suas linguagens de DML e DDL;

− existem linguagens DDL/DML no SGBD que estão baseadas numa versão padronizada (quais): O objetivo deste critério é levantar se existe alguma linguagem DML/DDL no SGBD que obedece às características de uma versão padrão da mesma;

b) linguagem de consulta: este atributo tem por objetivo avaliar a usabilidade da linguagem de consulta do SGBD, verificando os seguintes pontos:

- quais são as Linguagens de consulta que o SGBD suporta: O objetivo deste critério é levantar quais são as linguagens de consulta disponíveis no SGBD;
- existem linguagens de consulta no SGBD que são proprietárias (quais): O objetivo deste critério é verificar se existem linguagens, dentre as disponíveis no SGBD, que são proprietárias, afim de, novamente, verificar pontos como a disponibilidade de documentação e treinamento sobre as mesmas;
- existem linguagens de consulta no SGBD que são complexas (quais): O objetivo deste critério é avaliar se existem linguagens de consulta no SGBD que são complexas afim de, novamente, levantarem-se pontos como a disponibilidade de documentação e treinamento sobre as mesmas;
- existem linguagens de consulta no SGBD que estão baseadas numa versão padronizada (quais): O objetivo deste critério é levantar se existe alguma linguagem de consulta no SGBD que obedece às características de uma versão padrão da mesma;
- as linguagens de consulta do SGBD permitem consultas associativas: O objetivo deste critério é avaliar se as linguagens de consulta do SGBD permitem consultas associando mais de uma tabela ou classe através de expressões de união e interseção;

c) processo de desenvolvimento da aplicação: este atributo tem por objetivo avaliar a usabilidade do SGBD em relação ao desenvolvimento de aplicações, verificando os seguintes pontos:

- o SGBD possui ferramentas para otimizar o processo de definição dos dados: O objetivo deste critério é verificar se o SGBD possui ferramentas para auxiliar no processo de definição dos dados e quais são as características destas ferramentas (interface, grau de dificuldade e complexidade de conceitos implementados);
- o SGBD possui ferramentas para administração dos dados: O objetivo deste critério é verificar se o SGBD possui ferramentas que facilitem processos como a criação, reorganização e até mesmo eliminação de bancos de dados no SGBD;

- o SGBD possui ferramentas para geração de código fonte para aplicativos de manipulação de dados: O objetivo deste critério é verificar se o SGBD apresenta *wizards* para geração de código fonte para aplicativos de manipulação de dados com base no dicionário de dados existente no SGBD;
- o SGBD possui uma biblioteca padrão de classes ou procedimentos para maior produtividade do processo de desenvolvimento: O objetivo deste critério é verificar se o SGBD possui uma biblioteca padrão de classes ou procedimento que auxiliem o desenvolvedor no processo de desenvolvimento da aplicação.

5.3.2.3 Portabilidade

A análise das características de portabilidade do SGBD consiste na verificação dos seguintes pontos:

a) as mesmas versões do SGBD para diferentes plataformas são compatíveis: O objetivo deste critério é avaliar a compatibilidade das mesmas versões do SGBD em diferentes plataformas para levantarem-se pontos como a possibilidade de se obter um SGBD distribuído em diferentes plataformas ou migrar de plataforma o servidor de dados;

b) o SGBD garante o acesso de clientes em diferentes plataformas: O objetivo deste critério é avaliar se o SGBD disponibiliza *Drivers* para se acessar, por exemplo, um servidor em plataforma Risc/Unix diretamente de uma estação cliente em plataforma Intel/Windows 98 ou Intel/Windows NT.

5.3.2.4 Características Gerais

A análise das características gerais do SGBD tem por objetivo verificar os pontos que não são de cunho técnico mas que também devem ser levados em consideração quando da escolha de um SGBD. São atributos de análise das características gerais do SGBD:

a) maturidade do produto: consiste em avaliar a maturidade do produto verificar os seguintes pontos:

- quantos anos de desenvolvimento possui o produto: O objetivo deste critério é levantar, na forma de números, quantos são os anos de desenvolvimento do software SGBD;

- qual o número de usuários licenciados do produto: O objetivo deste critério é levantar, na forma de números, quantos são os usuários licenciados do produto;
- qual o número de usuários licenciados do produto que trabalham com o desenvolvimento de aplicações: O objetivo deste critério é levantar, na forma de números, quantos são os usuários licenciados do produto que trabalham com o desenvolvimento de aplicações;

b) documentação do produto: consiste em avaliar a documentação do SGBD verificar os seguintes pontos:

- a documentação do produto é clara: O objetivo deste critério é avaliar se o documentação do produto é compreensível;
- a documentação do produto é consistente: O objetivo deste critério é avaliar se não existem inconsistências na documentação do produto, como por exemplo, controvérsias no que diz respeito a explicações do mesmo tema;
- a documentação do produto é completa: O objetivo deste critério é avaliar se a documentação do produto é completa, apresentando a documentação do software como um todo, o guia de usuário das linguagens DML/DDL e Consulta do banco, exemplos completos das capacidades básicas de programação das linguagens de DML/DDL e Consulta, *Help On-line* nas ferramentas de administração e definição dos dados, etc;

c) treinamento do produto: consiste em avaliar as características de treinamento do produto verificar o seguinte ponto:

- qual a disponibilidade de cursos/treinamentos para o SGBD: O objetivo deste critério é avaliar qual é a disponibilidade de cursos/treinamentos para o SGBD. É fácil encontrar cursos/treinamentos para o SGBD? Os cursos/treinamentos do SGBD tem alto custo? Os cursos/treinamentos do SGBD são de boa qualidade?;

d) maturidade do implementador: consiste em avaliar a maturidade do implementador verificar os seguintes pontos:

- qual o porte do implementador: O objetivo deste critério é avaliar o tamanho do implementador com base no número de funcionários diretos e franquias que o mesmo possui;

– qual a situação financeira do implementador: O objetivo deste critério é avaliar qual a estabilidade financeira do implementador;

– quantos anos de experiência possui o implementador: O objetivo deste critério é levantar, na forma de números, quantos são os anos de experiência do implementador na área de SGBD;

– qual a experiência do pessoal técnico e administrativo do implementador no mercado de SGBD: O objetivo deste critério é avaliar se o pessoal técnico e administrativo do implementador possui conhecimento sobre a área de SGBD;

e) suporte do implementador: consiste em avaliar as características de suporte do implementador verificar os seguintes pontos:

– existe suporte por telefone: O objetivo deste critério é avaliar se o implementador disponibiliza suporte por telefone e em quanto tempo;

– existe suporte por correio eletrônico: O objetivo deste critério é avaliar se o implementador disponibiliza suporte através do correio eletrônico e em quanto tempo;

– existem peritos de suporte para as áreas de definição de dados, administração de dados e otimização de aplicações: O objetivo deste critério é avaliar se o suporte disponibilizado pelo implementador possui peritos para cada uma das áreas citadas acima ou se todo pessoal de suporte está habilitado a prestar suporte em todas as áreas.

Os critérios de avaliação descritos nos itens “d” e “e” também podem ser aplicados as revendas autorizadas do SGBD.

5.3.2.5 Desempenho

O desempenho é, sem dúvida, o critério de avaliação que apresenta maior complexidade para sua aplicação. Uma das melhores formas de se testar o desempenho do SGBD é submetê-lo a um processo de *BenchMark*. Um *BenchMark* efetivo deve levar em consideração os seguintes aspectos:

a) o número de usuários interativos do sistema aplicativo;

b) as taxas de atualização e acesso do SGBD;

c) o tamanho do SGBD;

d) as configurações de hardware de uma possível rede onde a aplicação irá rodar;

e) os acessos padrões da aplicação.

Dessa forma, para um *BenchMark* prover ao desenvolvedor informações úteis sobre o desempenho do SGBD, este deve ser modelado para imitar ao máximo o comportamento da aplicação que está sendo desenvolvida.

6 DESENVOLVIMENTO DO PROTÓTIPO

6.1 Metodologia de Desenvolvimento

Para especificação da modelagem relacional dos dados no Caché, utilizou-se a Análise Estruturada e o Modelo Conceitual de Dados. Segundo [ABR95], na análise estruturada as funções de um programa são organizadas de forma hierárquica, através da especificação dos processos de transformação de dados. O Modelo Conceitual de Dados ou Modelo Entidade-Relacionamento consiste de um diagrama que detalha as associações existentes entre as entidades de dados e utiliza componentes semânticos próprios.

Para especificação da modelagem orientada a objetos dos dados no Caché, utilizou-se a análise orientada a objetos definida por Peter Coad e Edward Yourdon em [COA91], que sugere a criação de um modelo lógico do sistema através de classes e objetos.

6.2 Descrição do Problema

O gerenciamento de Ordens de Serviço (OS), problema comum em toda e qualquer empresa que possui um setor próprio de manutenção, é parte integrante do processo de Planejamento, Programação e Controle da Manutenção. Cada vez mais evidencia-se, no cenário da industria nacional, a importância do setor de manutenção no contexto gerencial das empresas, afetando seus objetivos em produtividade, qualidade e competitividade [AGO97].

O atual cenário do processo de gerenciamento produtivo começa a perceber as conseqüências derivadas da indisponibilidade e demais ocorrências apresentadas pelas instalações e equipamentos, caracterizando a importância de um sistema informatizado que possibilite a implantação de modelos de planejamento de manutenção que venham a atender as reais necessidades da empresa.

No mesmo instante, o processo de gerenciamento administrativo também percebe que as conseqüências derivadas por problemas técnicos de equipamentos de uso diário, como um simples telefone ou um computador, acarretam sérios prejuízos para a empresa, na forma de “perda de tempo” em busca de soluções que poderiam ser agilizadas por sistema informatizado que agilizasse o processo de solicitação de conserto.

Nesse cenário propõem-se a trabalhar o processo de gerenciamento de OS. Não que o mesmo venha a solucionar como um todo os problemas citados acima, mas sim, servir de ferramenta para amenizar a ocorrência dos mesmos. O processo de gerenciamento de OS é formado pelas seguintes etapas:

a) abertura da OS: Processo realizado pela pessoa que necessita de um serviço. Consiste em relacionar informações, como: o serviço a ser executado, o local de execução do serviço (Centro de Custo), a pessoa que está solicitando, etc;

b) programação da OS: Processo realizado pelo programador do setor de manutenção. Este processo visa encaminhar a OS para a pessoa que realizará os serviço solicitados;

c) apontamentos da OS: Processo realizado pelo executante do serviço. Consiste em relacionar na OS dados como o tempo e os materiais gastos com o serviço;

d) fechamento da OS: Processo realizado pelo programador da manutenção. Consiste em encerrar a OS quando o serviço solicitado na mesma já foi realizado;

e) cancelamento da OS: Consiste em cancelar Ordens de Serviço sem executar o serviço nela solicitado. Inúmeros são os motivos que podem levar ao cancelamento de uma ordem de serviço, dentre os quais pode-se citar: OS abertas em duplicidade, OS com a descrição do serviço a ser executado errado, entre outros.

6.3 Diagrama de Contexto

O diagrama de contexto solução do problema encontra-se especificado na figura 6.1 e foi construído com o software PowerDesigner 6.1 ProcessAnalyst ™ da SyBase.

Existe, no contexto geral do sistema, a interação de duas entidades externas:

a) setor de manutenção: a entidade setor de manutenção do diagrama de contexto corresponde a todos os setores da empresa que prestam serviços de manutenção. Essa entidade é responsável pela programação da OS como um todo (programação, cancelamento, apontamentos de mão-de-obra e materiais, baixa e reabertura), bem como cadastrar as máquinas, prioridades de execução e tipos de serviço. O setor de manutenção também pode solicitar a execução de OS;

b) empresa: a entidade empresa representa todos os demais setores e departamentos da empresa. Em resumo, pode-se dizer que a entidade empresa representa os usuários dos serviços de manutenção. É a entidade empresa que solicita a execução de OS. A

entidade empresa também é responsável por manter os cadastros gerais utilizados pelo sistema, como o cadastro de estados, cidades, itens, grupos de itens, entre outros.

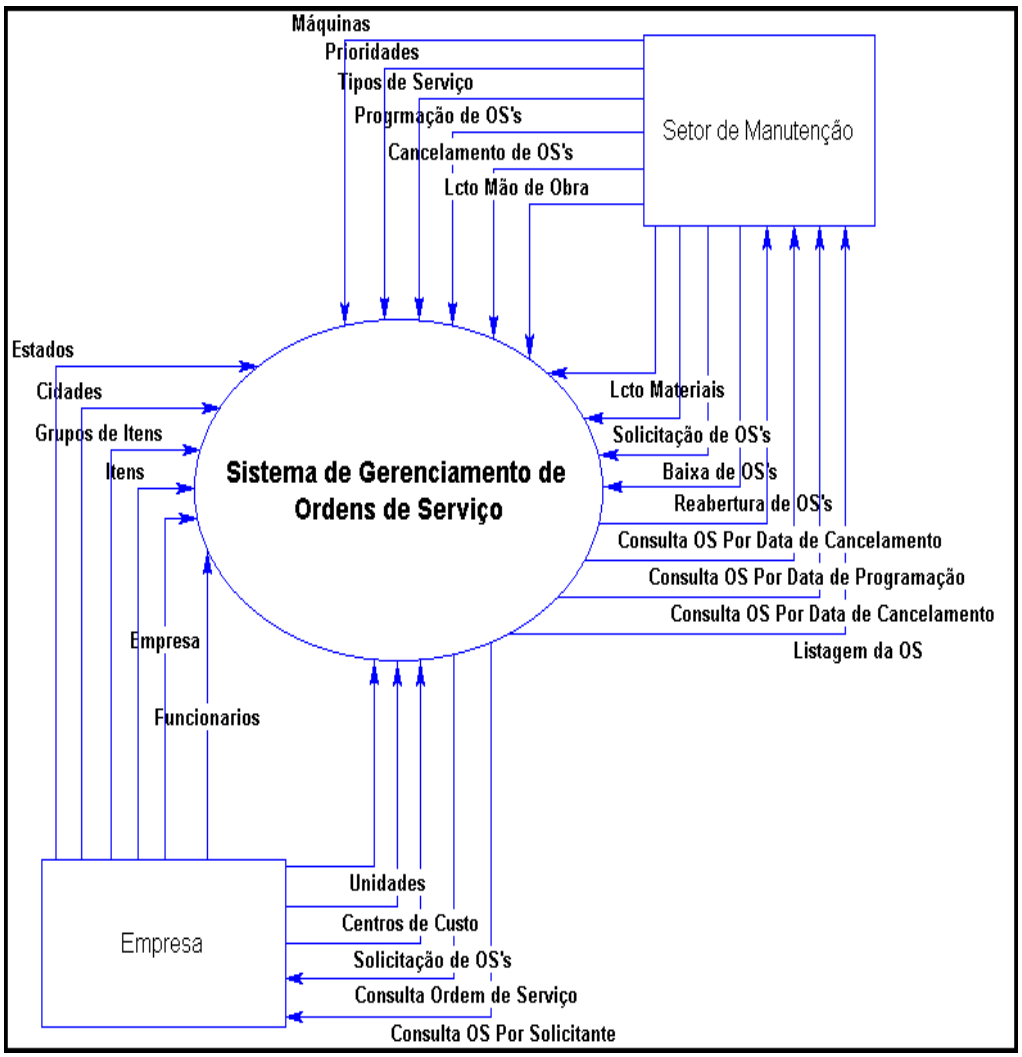

Figura 6.1 - Diagrama de Contexto

6.4 Modelo Entidade – Relacionamento

O modelo ER solução do problema encontra-se especificado na figura 6.2 e foi construído com a ferramenta PowerDesigner 6.1 DataArchitect ™ da SyBase. O dicionário de dados do referido modelo ER encontra-se descrito no Anexo 5.

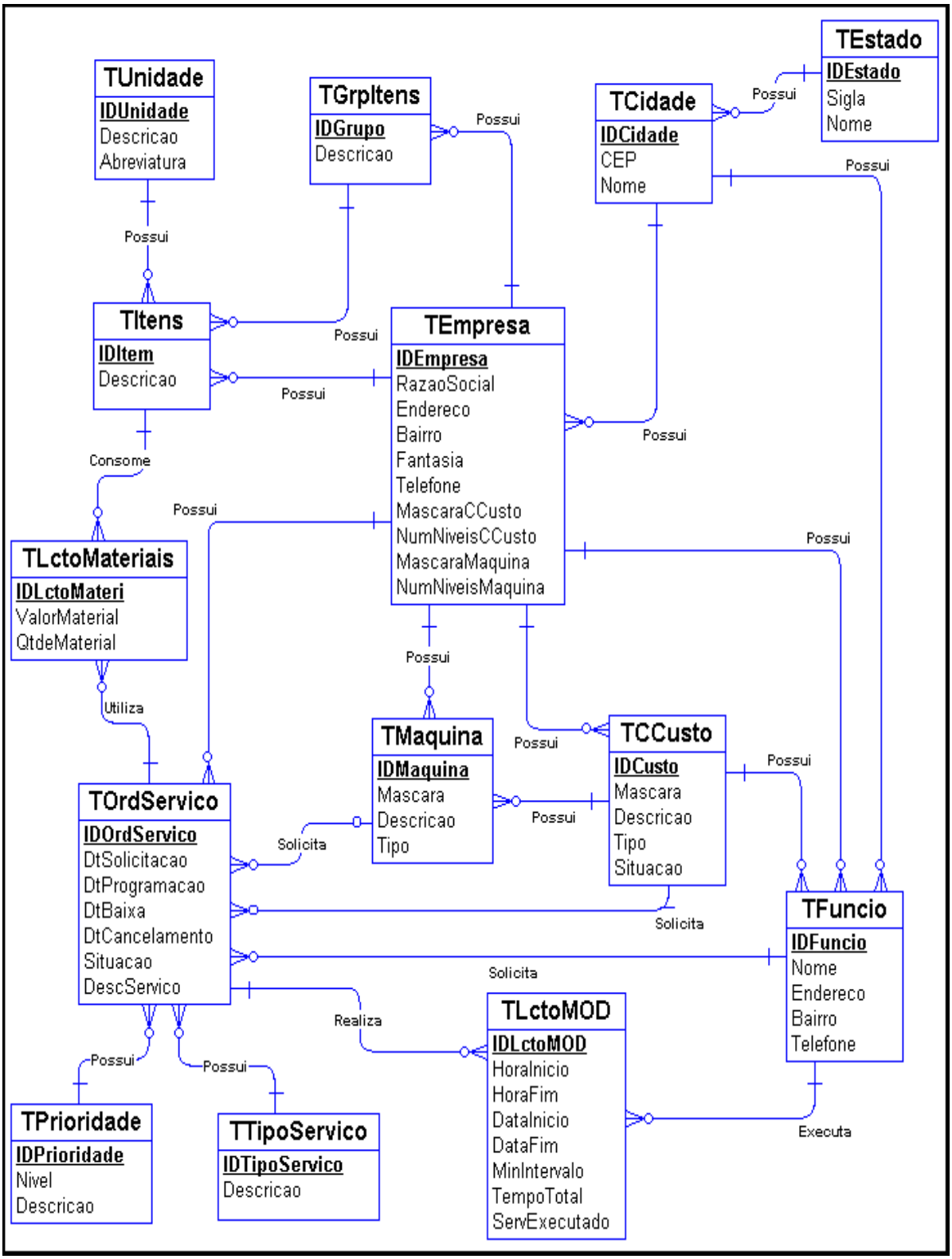

Figura 6.2 - Modelo Entidade - Relacionamento

6.5 Diagrama de Hierarquia de Funções

O diagrama de hierarquia de funções do protótipo desenvolvido neste trabalho está especificado na figura 6.3 e foi construído com o software Organograma Microsoft.

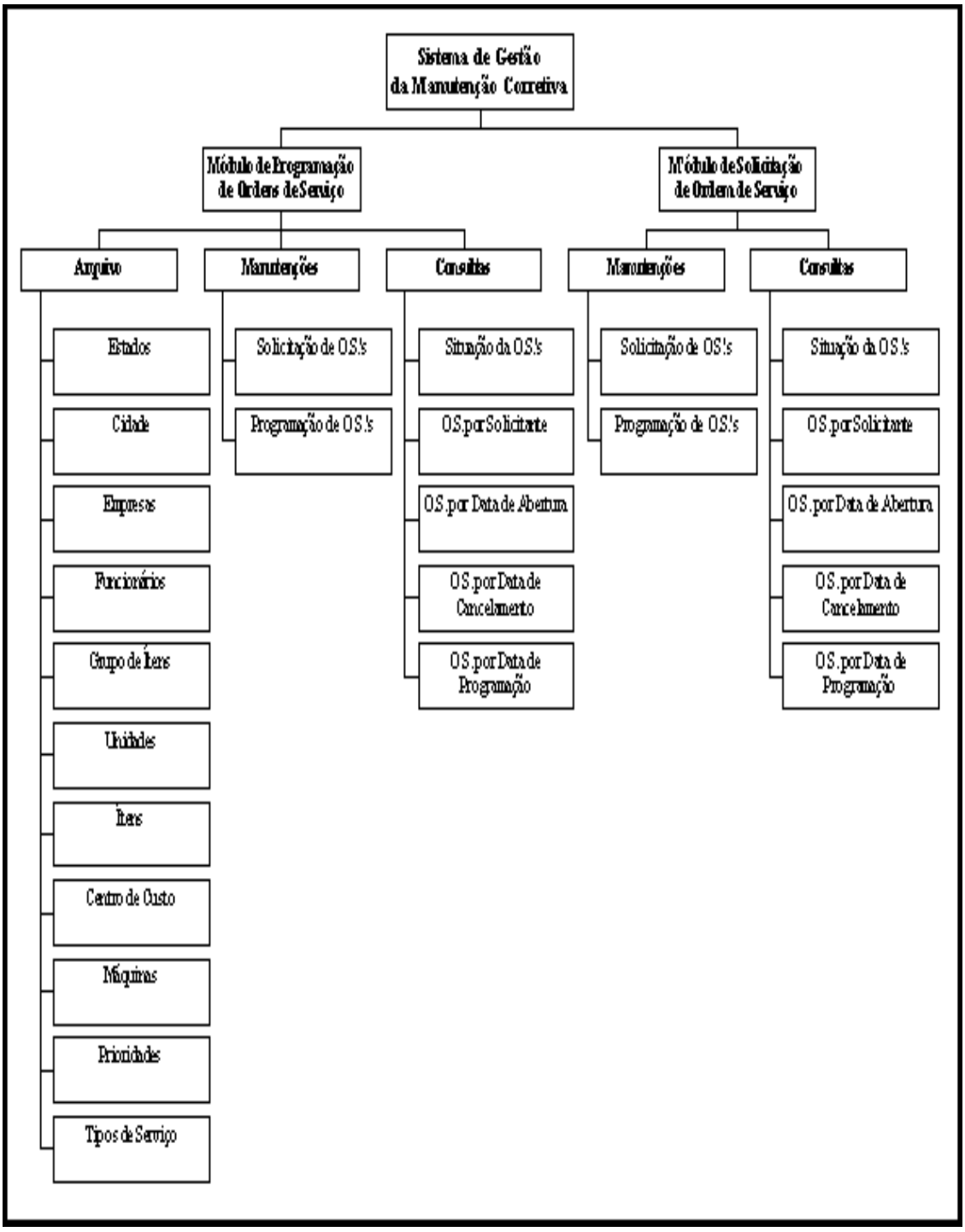

Figura 6.3 - Diagrama de Hierarquia de Funções

6.6 Diagrama de Hierarquia de Processos

O diagrama de hierarquia de processos do protótipo desenvolvido neste trabalho, especificado na figura 6.4, tem por objetivo determinar a lógica de ocorrência dos processos no sistema. Setas precedentes significam que estes processos devem ser executados antes. Este diagrama foi construído com o software Flow Charting 4, da Patton & Patton Software Corporation.

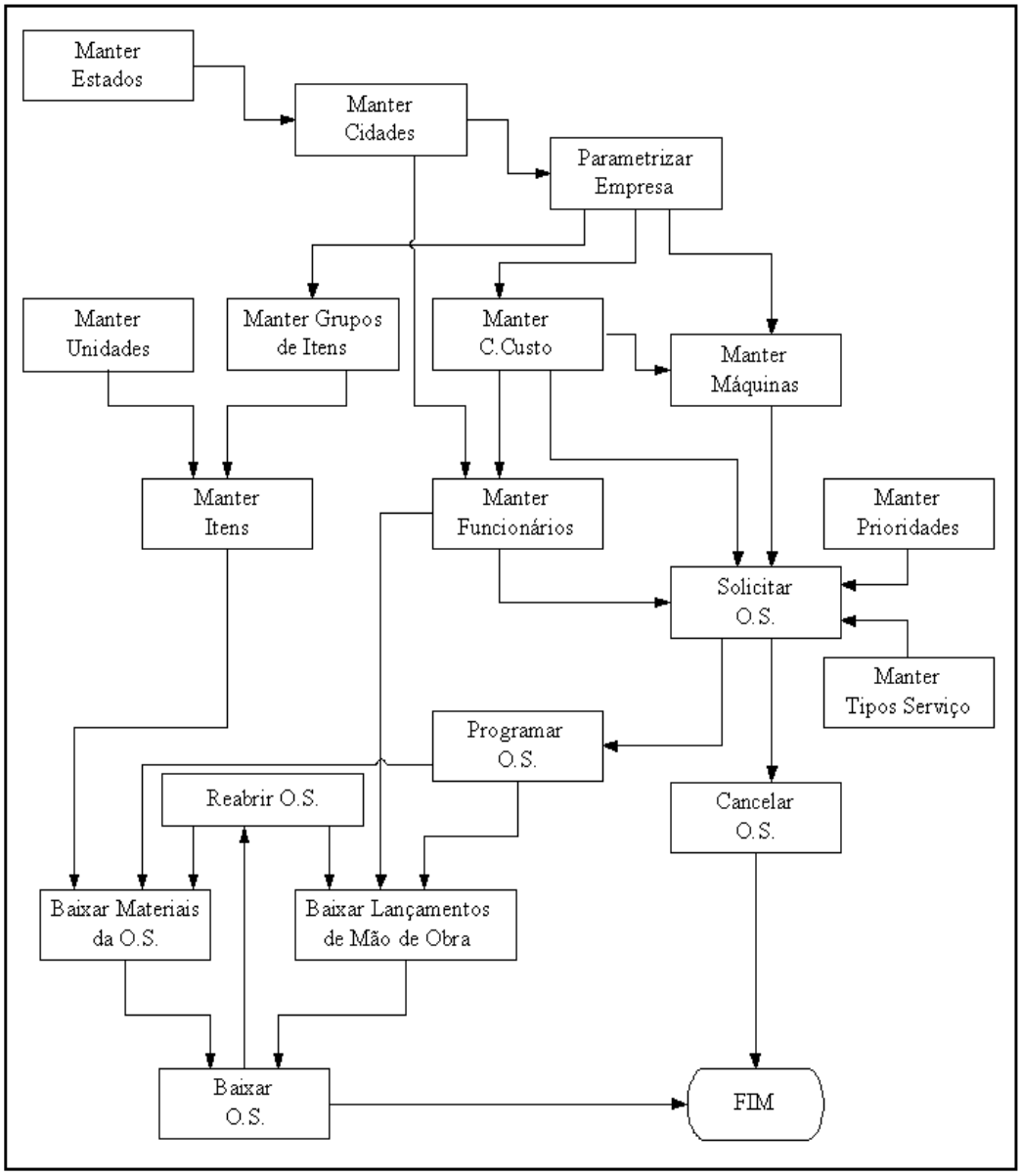

Figura 6.4 - Diagrama de Hierarquia de Processos

6.7 Diagrama de Classes-&-Objetos da Análise

O diagrama de classes-&-objetos da análise do protótipo encontra-se especificado na figura 6.5 e foi construído com a ferramenta Playground 1.0, da Object International. Esse diagrama segue os padrões de especificação de modelos de dados orientados a objetos descritos em [COA91]. O dicionário de dados do referido modelo, em virtude da similaridade dos dados, é o mesmo descrito no Anexo 5. A descrição dos métodos das classes do modelo encontra-se na tabela 6.1. Métodos comuns à todas as classes, como incluir, excluir, alterar e consultar não foram incorporados ao modelo

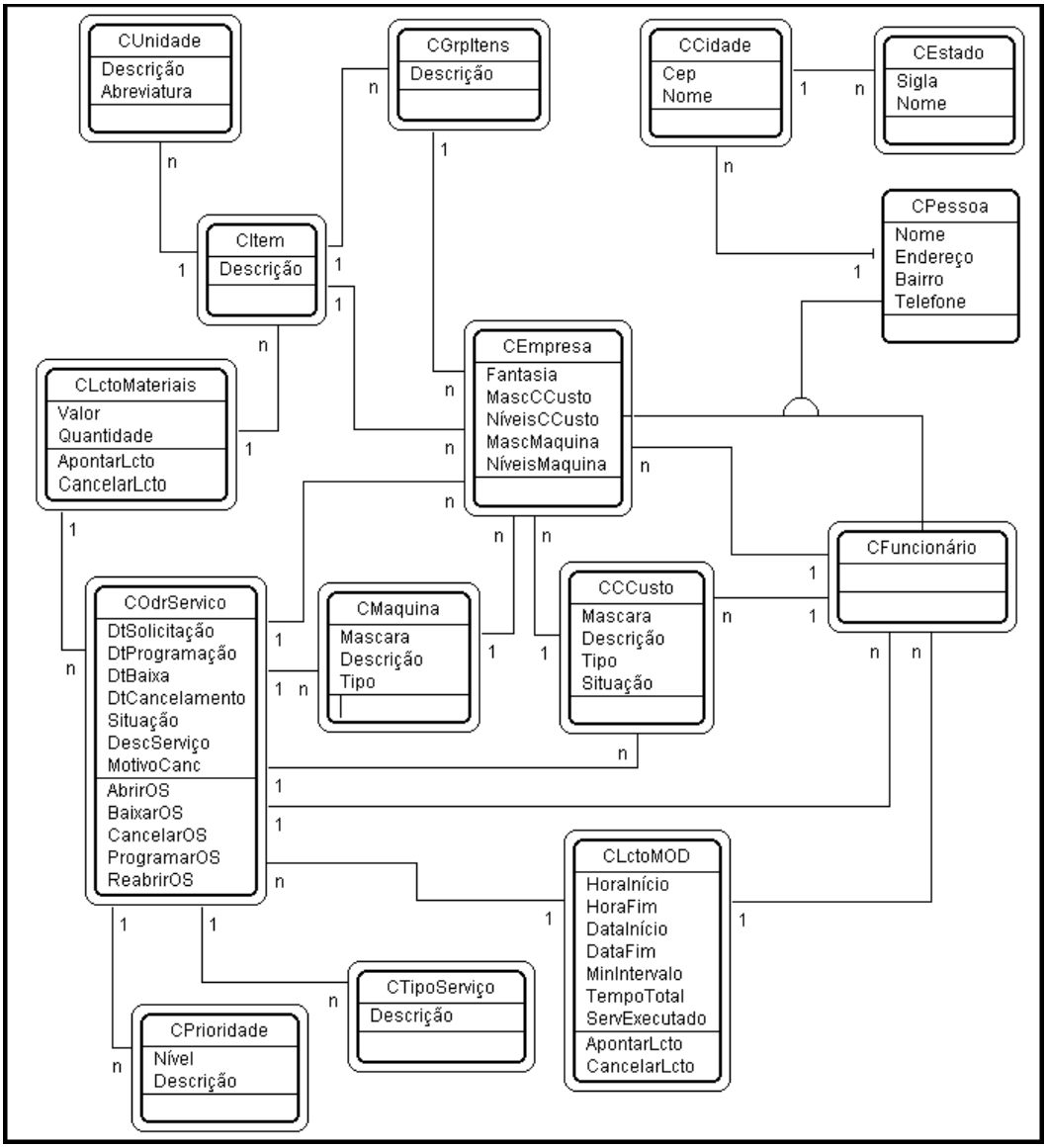

Figura 6.5 - Diagrama de Classes-&-Objetos da Análise

Classe	Método	Descrição
COrdServico	AbrirOS	Esse método é executado quando a OS é aberta. Em resumo, ele responsável por criar uma nova instância da classe COrdServico em disco.
COrdServico	BaixarOS	Esse método é executado quando a OS é encerrada. Em resumo, ele altera a situação da OS para encerrada e grava a data de baixa da mesma.
COrdServico	CancelarOS	Esse método é executado quando a Solicitação de OS é cancelada. Em resumo, ele altera a situação da OS para cancelada e grava a data de cancelamento da mesma.
COrdServico	ProgramarOS	Esse método é executado quando a Solicitação de OS é programada. Em resumo, ele altera a situação da OS para programada e grava a data de programação da mesma.
COrdServico	ReabrirOS	Esse método é executado quando deseja-se reabrir uma OS que já foi baixada. Em resumo, ele altera a situação da OS de baixada para programada e atribui nulo à data de baixa da OS.
CLctoMOD	ApontarLcto	Esse método é executado quando deseja-se informar ao sistema os apontamentos de Mão-de-Obra. Em resumo, esse método cria um nova instância da Classe CLctoMOD em disco.
CLctoMOD	CancelarLcto	Esse método é executado quando deseja-se cancelar do sistema determinado apontamento de Mão-de-Obra. Em resumo, esse método exclui fisicamente uma instância em disco da Classe CLctoMOD.
CLctoMateriais	ApontarLcto	Esse método é executado quando deseja-se informar ao sistema os apontamentos de Materiais. Em resumo, esse método cria uma nova instância da Classe CLctoMateriais em disco.
CLctoMateriais	CancelarLcto	Esse método é executado quando deseja-se cancelar do sistema determinado apontamento de Materiais. Em resumo, esse método exclui fisicamente uma instância em disco da Classe CLctoMateriais.

Tabela 6.1 - Descrição dos Métodos do Diagrama de Tipos de Objetos

6.8 Diagrama de Classes-&-Objetos do Projeto

O diagrama de classes-&-objetos do projeto do protótipo encontra-se especificado na figura 6.6 e também foi construído com a ferramenta Playground 1.0, da Object International. Esse diagrama segue os padrões de especificação de modelos de dados orientados a objetos descritos em [COA91]. O dicionário de dados do referido modelo, em virtude da similaridade dos dados, é o mesmo descrito no Anexo 5. A descrição dos métodos das classes do modelo encontra-se na tabela 6.2.

Classe	**Método**	**Descrição**
CPersistente	First	Esse método move a classe para sua primeira instância em disco.
CPersistente	Last	Esse método move a classe para sua última instância em disco.
CPersistente	Prev	Esse método move a classe uma instância para trás.
CPersistente	Next	Esse método move a classe uma instância à frente.

Tabela 6.2 - Descrição dos Métodos do Diagrama Classes

Observação: Os métodos First, Last, Prev e Next da classe CPersistente foram desenvolvidos para efetuarem a interface entre a ferramenta de implementação do protótipo (o Borland Delphi 4) e o modelo de objetos do Caché Objects pois não foram encontrados componentes para acesso nativo de objetos Caché com o ambiente Delphi. Em resumo, esses métodos permitem a navegabilidade entre as instâncias de determinada classe.

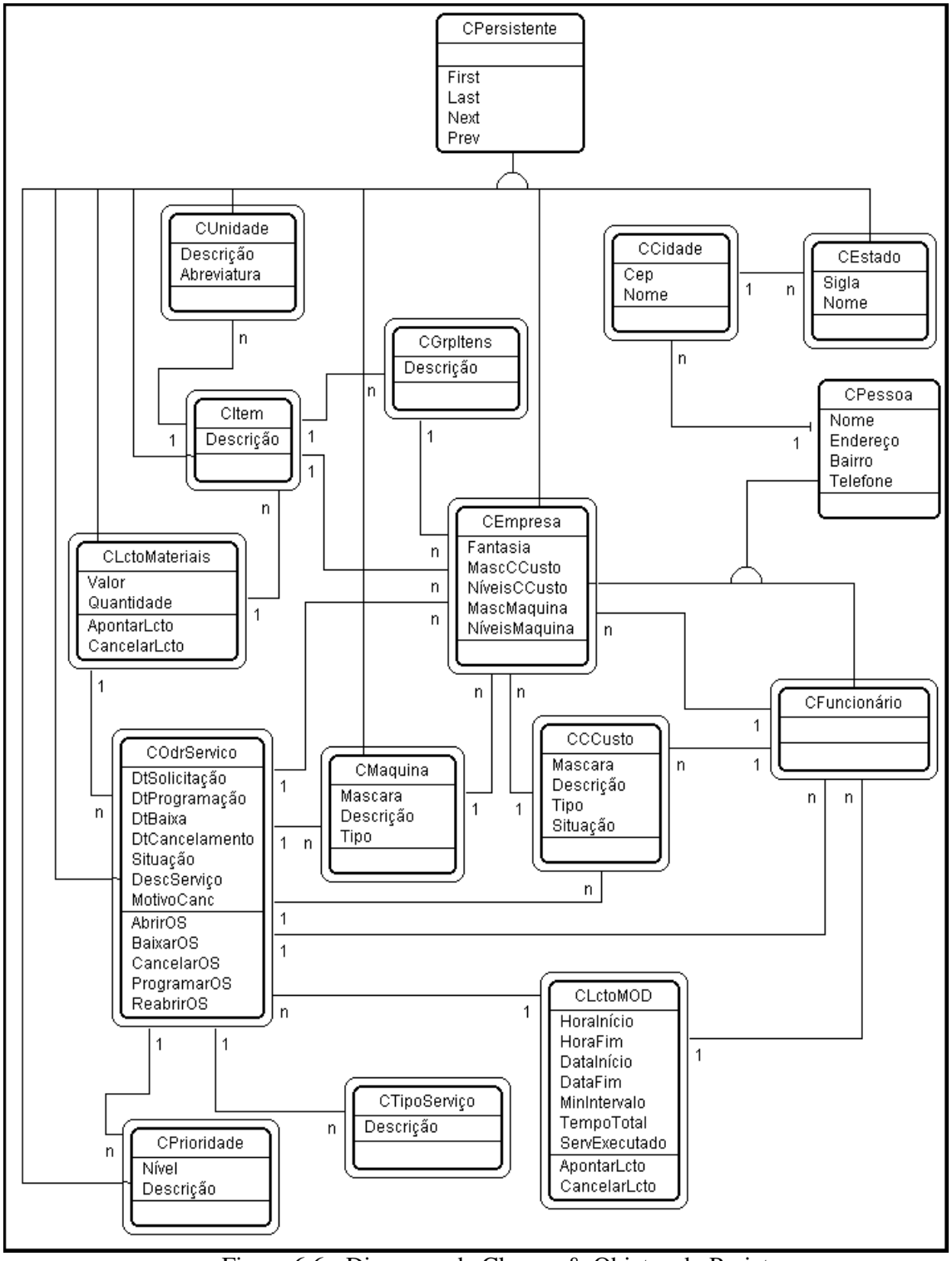

Figura 6.6 - Diagrama de Classes-&-Objetos do Projeto

6.9 Aplicação dos Critérios de Avaliação

A seguir é apresentado, na forma de questionário, o resultado da aplicação dos critérios de avaliação de SGBD definidos nesse trabalho no capítulo 5 sobre o contexto da aplicação e do SGBD do protótipo desenvolvido.

6.9.1 Avaliação do Software Aplicativo

Segue abaixo o questionário/avaliação do software aplicativo:

a) A aplicação possui manipulação de dados de multimídia (Som, Imagem, objetos BLOB – *Binary Large Objects*)?

R= Não, não foram incorporados objetos multimídia ao protótipo

b) É possível transcrever os dados do mundo real da aplicação para a forma de tabelas sem tornar o modelo de dados complexo?

R= Sim, o modelo de tabelas do referido protótipo não é complexo.

c) A estrutura dos dados da aplicação é estável ou sofre constantes mudanças?

R= O sistema de gerenciamento de Ordens de Serviço pode ser considerado um Sistema de estrutura de dados estável.

d) Os dados da aplicação terão que ser acessados por mais de um usuário ao mesmo tempo?

R= A necessidade do mesmo dado (no caso, a Ordem de Serviço) ter que ser acessada por mais de um usuário ao mesmo tempo não pode ser desconsiderada. Porém, como as transações a nível de SGBD são extremamente rápidas, esse acesso "simultâneo" é praticamente imperceptível.

e) Os dados da aplicação são comportamentais?

R= Não, o sistema não apresenta a existência de dados comportamentais.

f) As transações realizadas pela aplicação são curtas e direcionadas a alguns registros?

R= Sim, a aplicação não apresenta a ocorrência de transações de longa duração.

g) Há necessidade de transações em cooperação de usuários?

R= Embora a mesma Ordem de Serviço seja manipulada por mais de um usuário ao mesmo tempo, no que diz respeito a apontamentos de Mão-de-Obra e Materiais, estes à manipulam para fins específicos. Dessa forma, não existe na aplicação transações que devam ser realizadas em cooperação de usuários para um fim comum.

h) O controle pessimista de concorrência (bloqueio exclusivo) é suficiente para atender a aplicação?

R= Mesmo levando em consideração que mais de um usuário acesse a mesma O.S. ao mesmo tempo, as transações envolvidas na aplicação são extremamente rápidas e facilmente gerenciadas pelo mecanismo de bloqueio exclusivo.

i) Qual o processo de desenvolvimento da aplicação?

R= A aplicação englobou ambos os processos de desenvolvimento, tanto estruturado quanto orientado a objeto.

6.9.2 Avaliação do Software SGBD

A seguir é apresentado o questionário/avaliação do software de SGBD utilizado na implementação do protótipo.

6.9.2.1 Funcionalidade

Segue abaixo o questionário/avaliação da Funcionalidade do SGBD:

a) O SGBD garante a integridade dos dados em atualizações simultâneas?

R= Sim, o Caché apresenta inúmeros mecanismos para controle de concorrência.

b) O SGBD garante a integridade dos dados durante os processos de recuperação?

R= Sim, segundo os manuais técnicos do Caché.

c) O SGBD apresenta suporte a mecanismos de *constraints* dos dados?

R= Ambos os modelos de dados do Caché (Relacional e Orientado a Objetos) suportam mecanismos de *Constraints.*

d) O(s) mecanismo(s) de controle de concorrência que o SGBD oferece atende(m) as necessidades da aplicação?

R= Sim, os mecanismos de controle de concorrência são baseados em processos internos no próprio banco e contemplam as necessidades da aplicação.

e) O SGBD provê o controle de concorrência baseado em múltiplas transações de leitura concorrentes mas só uma transação de escrita?

R= Não foram encontradas referências técnicas com relação à essa característica, porém, testes efetuados com o SGBD demonstram a existência da mesma.

f) O SGBD provê o mecanismo de controle baseado em múltiplas transações de escrita e leitura concorrentes?

R= Não foram encontradas referências técnicas com relação à essa característica, porém, testes efetuados com o SGBD demonstram a não existência da mesma.

g) O SGBD provê a recuperação de falhas de aplicação?

R= Sim, o SGBD garante a recuperação de falhas de aplicação através dos mecanismos de *Commit* e *Rollback*, tanto para o modelo de objetos quanto para o modelo relacional de dados.

h) O SGBD provê a recuperação de falhas de sistema operacional?

R= Sim, esse tipo de recuperação também é realizado através dos processos de *Commit e Rollback* de transações.

i) O SGBD provê a recuperação de falhas de Mídia?

R= Para sanar o problema de possíveis falhas de mídia, o SGBD suporta o conceito de dualização de servidores.

j) As transações do SGBD podem acessar dados localizados em qualquer parte do SGBD, se este for distribuído?

R= Não foram encontradas referências técnicas com relação a essa característica.

k) Existe suporte a transações longas?

R= Não foram encontradas referências técnicas com relação a essa característica.

l) Existe suporte a transações em grupo de usuários?

R= Sim, o SGBD permite a realização de transações em grupo de usuários.

m) O SGBD permite a classificação das transações?

R= Sim, o SGBD possui um controle próprio de processos, e estes podem ser priorizados.

n) O SGBD provê de mecanismos para descoberta de problemas de *DeadLock*?

R= Não foram encontradas referências técnicas no que diz respeito a detecção de problemas de *DeadLock*. Sabe-se, no entanto, que tanto o modelo relacional quanto o modelo orientado a objetos do Caché apresenta mecanismos para gerência de *Lock* e conseqüente prevenção contra problemas de *DeadLock*.

o) A forma como os problemas de *DeadLock* são resolvidos atende as necessidades da aplicação?

R= Não foram encontradas referências técnicas no que diz respeito a solução de problemas de DeadLock, no entanto, conforme testes efetuados, percebeu-se que uma possível solução é “abortar” um dos processos em conflito.

p) O SGBD apresenta mecanismos de Back-Up e *Restore*?

R= Sim, o SGBD apresenta mecanismos próprios para Back-Up e *Restore* de dados.

q) O Back-Up pode ser realizado com o SGBD *On-line*?

R= Sim, pode-se realizar o processo de Back-Up com o banco *on-line*. O processo de Back-Up também pode ser pré-programado.

r) O SGBD possui mecanismos de gerenciamento de índices para otimização de acessos?

R= Sim, é possível criar índices nas estruturas dos objetos e tabelas.

s) O usuário pode selecionar qual tecnologia de gerenciamento de índices que ele deseja utilizar no seu modelo de dados?

R= Não, o Caché implementa uma tecnologia de gerência de índices própria, baseada no modelo de dados multidimensional.

t) O SGBD provê controle de acesso multi-nível para o usuário?

R= Sim, o Caché permite definição de níveis de acesso para os usuários.

u) O SGBD provê controle de acesso multi-nível para usuário/dados?

R= Sim, quando se cria um usuário no Caché concedem-se privilégios de acesso a dados para o mesmo.

6.9.2.2 Usabilidade

Segue abaixo o questionário/avaliação da Usabilidade do software SGBD:

a) Quais são as linguagens de definição de dados (DDL) que o SGBD suporta?

R= SQL Padrão e Caché CDL.

b) Quais são as linguagens de manipulação de dados (DML) que o SGBD suporta?

R= SQL Padrão e COS.

c) Existem linguagens DML/DDL no SGBD que são proprietárias? Quais?

R= Sim. O COS e o Caché CDL são proprietárias.

d) Existem linguagens DML/DDL no SGBD que são complexas? Quais?

R= Não. O SQL do Caché segue as características funcionais do SQL padrão, que são simples e já estão consolidadas. Com relação ao COS, a linguagem não é complexa e é facilmente absorvida, principalmente por pessoas que já tiveram algum tipo de contato com a linguagem Mumps Padrão. O Caché CDL também é facilmente absorvido, principalmente por pessoas que possuem algum conhecimento teórico sobre OO.

e) As linguagens DML do SGBD garantem a completude computacional?

R= Sim, tanto o SQL quanto o COS do Caché suportam o conceito de completude computacional.

f) As linguagens DML/DDL do SGBD garantem a independência dos dados?

R= Sim, tanto o SQL quanto o COS suportam a independência entre dados e programas. O SQL implementa esse conceito através da interface ODBC. Já o COS implementa tal conceito implicitamente em sua estrutura de linguagem. Com relação ao Caché CDL, o mesmo não apresenta comandos de interação com os dados, apenas comandos de definição de dados.

g) As linguagens DML/DDL do SGBD possuem suporte à impedância desigual dos dados?

R= Sim, principalmente o COS, que implementa tal conceito em sua própria estrutura de linguagem. Já o SQL, suporta tal conceito através de sua interface ODBC.

h) Existem linguagens DML/DDL no SGBD que são OO ou apresentam características OO? Quais?

R= Sim, tanto o COS quanto o Caché CDL apresentam características de OO, como herança (inclusive herança múltipla), classes e OID.

i) Existem linguagens DDL/DML no SGBD que estão baseadas numa versão padronizada? Quais?

R= Sim , o SQL do Caché é baseado no SQL padrão.

j) Quais são as Linguagens de consulta que o SGBD suporta?

R= Tanto o modelo de objetos quanto o modelo Relacional do Caché suportam, como linguagem de consulta, o SQL padrão. No caso do modelo de objetos também é possível otimizar as consultas SQL através de expressões em COS.

k) Existem linguagens de consulta no SGBD que são proprietárias? Quais?

R= Apenas o COS, que serve para otimizar as expressões de consulta, é proprietário.

l) Existem linguagens de consulta no SGBD que são complexas? Quais?

R= Não, o SQL do Caché, como já mencionado, é baseado no SQL padrão, que já está consolidado no mercado. Já as expressões de otimização do SQL em COS são facilmente absorvidas.

m) Existem linguagens de consulta no SGBD que estão baseadas numa versão padronizada? Quais?

R= O SQL do Caché é baseado no SQL padrão.

n) As linguagens de consulta do SGBD permitem consultas associativas?

R= Sim, o SQL do Caché permite consultas associativas tanto de tabelas quanto de classes.

o) O SGBD possui ferramentas para otimizar o processo de definição dos dados?

R= Sim, tanto na definição de um modelo de dados relacional quanto na definição de um modelo de dados orientado a objetos existem ferramentas para otimizar o processo. O detalhe é que a ferramenta de definição de classes do Caché, como já mencionado anteriormente, é melhor que a ferramenta de definição de tabelas.

p) O SGBD possui ferramentas para administração dos dados?

R= Sim, existem ferramentas tanto para administração dos dados quanto para administração do banco como um todo, contemplando características como o gerenciamento de processos, acessos e controle de *Lock*.

q) O SGBD possui ferramentas para geração de código fonte para aplicativos de manipulação de dados?

R= O Caché disponibiliza um componente para Visual Basic que permite, a partir do dicionário de classes, a criação de programas para manipulação de dados na forma de objetos. Com relação ao Caché Relacional, podem ser utilizados os *wizards* padrões para geração de código fonte para manipulação de banco de dados que acompanham ferramentas como o Visual Basic e o Delphi.

r) O SGBD possui uma biblioteca padrão de classes ou procedimentos para maior produtividade do processo de desenvolvimento?

R= Sim, o Caché apresenta uma série de classes de sistema que servem para realizar diversos processos, como por exemplo, o acesso à instâncias de classe em disco.

6.9.2.3 Portabilidade

Segue abaixo o questionário/avaliação da Portabilidade do software SGBD:

a) As mesmas versões do SGBD para diferentes plataformas são compatíveis?

R= Sim, as versões do Caché para todo tipo de plataforma que o mesmo suporta são totalmente compatíveis. Para por exemplo, migrar uma base de dados Caché instalada em uma plataforma Intel/Windows9* para uma plataforma Risc/Unix, basta apenas transferir a base de dados

b) O SGBD garante o acesso de clientes em diferentes plataformas?

R= Sim, clientes em qualquer plataforma podem acessar um servidor Caché em outra plataforma. Por exemplo, uma base Caché instalada em um servidor Risc/Unix pode receber acessos concorrentes tanto de plataformas Intel/Windows 9* quanto de terminais burros

6.9.2.4 Características Gerais

Segue abaixo o questionário/avaliação das Características Gerais do software SGBD:

a) Quantos anos de desenvolvimento possui o produto?

R= O InterSystems Caché está em desenvolvimento desde 1994. Entretanto, vale salientar que o Caché consiste na evolução de outro produto da InterSystems, denominado de Open M Server, que já é desenvolvido há mais tempo.

b) Qual o número de usuários licenciados do produto?

R= Segundo dados da própria InterSystems, são mais de 2.000.000.

c) Qual o número de usuários licenciados do produto que trabalham com o desenvolvimento de aplicações?

R= Infelizmente esse dado não foi encontrado.

d) A documentação do produto é clara?

R= Sim, toda parte documentacional do Caché, tanto em seus manuais quanto em seu sistema de *help on-line* são claros.

e) A documentação do produto é consistente?

R= Sim, a documentação do Caché, dentro do que foi pesquisado para elaboração deste protótipo, não apresentou nenhuma espécie de inconsistência.

f) A documentação do produto é completa?

R= No ponto de documentação completa, só está faltando o sistema de *help on-line* do Caché Object Architect e melhorar o *help on-line* do %msql.

g) Qual a disponibilidade de cursos/treinamentos para o SGBD?

R= Hoje, apenas a empresa representante direta do Caché no Brasil (LEME Informática) e suas subsidiárias apresentam planos de cursos/treinamentos sobre o Caché.

h) Qual o tamanho do implementador?

R= A InterSystems conta hoje com mais de 130 funcionários diretos, distribuidores em mais de 20 países e clientes em 88 países.

i) Qual a situação financeira do implementador?

R= O único dado financeiro da empresa disponível (inclusive em material publicitário) garante que a empresa mantém lucratividade desde sua fundação, em 1978.

j) Quantos anos de experiência possui o implementador?

R= A InterSystems apresenta hoje uma experiência de 21 anos no mercado de software.

k) Qual a experiência do pessoal técnico e administrativo do implementador no mercado de SGBD?

R= Infelizmente esse dado não foi encontrado.

l) Existe suporte por telefone?

R= Sim, tanto a própria InterSystems quanto suas distribuidoras oferecem o suporte por telefone. O suporte à InterSystems deve acontecer primeiro nas franquias de suas distribuidoras. Se estas não conseguirem resolver o problema, elas encaminham o mesmo para a distribuidora da InterSystems do determinado país. Caso a distribuidora não consiga resolver o problema, o mesmo é encaminhado para a própria InterSystems.

m) Existe suporte por correio eletrônico?

R= Sim, a InterSystems, suas distribuidoras e franquias oferecem o mecanismo de suporte através de correio eletrônico.

n) Existem peritos de suporte para as áreas de definição de dados, administração de dados e otimização de aplicações?

R= Na InterSystems esses peritos existem. Em relação à suas distribuidoras e franquias, tal detalhe depende da estrutura das mesmas.

6.10 Implementação do Protótipo

A ferramenta utilizada para implementar o protótipo deste trabalho foi o Borland Delphi 4.0. Segundo [LOY97] , o Delphi não é apenas uma linguagem de programação. O Delphi é um ambiente integrado de desenvolvimento (IDE do inglês *Integrated Development Environment*) que permite a edição, compilação e execução de um programa, além de também oferecer uma série de utilitários para facilitar processos de criação de banco de dados e suas respectivas tabelas. A linguagem de programação que está por trás do ambiente Delphi é o Object Pascal, uma LPOO.

A versão do Banco de Dados Caché utilizada para armazenar o modelo de dados do protótipo foi a 2.1.6-F.12 (*build* 028). Nesta versão do Caché, o modelo de dados orientado a objetos disponível apresenta-se em versão *beta*. Somente a partir da versão 3.1 (*build* 46) do Caché, lançada em Maio de 1999, que o suporte ao modelo de objetos passou a ser disponibilizado oficialmente. O modelo de dados relacional disponível na versão 2.1.6-F.12 (*build* 028) é oficial e suporta o SQL padrão.

O modelo de dados do protótipo engloba os conceitos de armazenamento orientado a objetos e relacional do Caché. Para trabalhar com ambos os modelos, os programas do protótipo foram divididos da seguinte forma:

a) O cadastro de estados, empresas, unidades, itens, máquinas e prioridades foram desenvolvidos com seus dados sendo manipulados via objetos no Caché;

b) O cadastro de cidades, funcionários, grupos de itens, centros de custo e tipos de serviço foram desenvolvidos com seus dados sendo manipulados via tabelas no Caché;

c) Os programas de solicitação de ordens de serviço e de apontamentos de mão-de-obra foram desenvolvidos trabalhando seus dados como tabelas no Caché;

d) Os programas de programação de ordens de serviço e de apontamentos de materiais foram desenvolvidos trabalhando seus dados como objetos no Caché;

e) Como a linguagem de consulta do Caché tanto para o modelo relacional quanto para o modelo orientado a objetos é o SQL, todas as consultas do sistema foram desenvolvidas com base em construções SQL. Vale salientar, que apesar do SQL do Caché ser baseado na linguagem SQL padrão, ele é otimizado por comandos COS quando da necessidade de construções mais complexas para manipulação de bases de dados de objetos ou tabelas.

Na figura 6.7 encontra-se a tela de *Login* do protótipo. Nela, são identificados a empresa na qual deseja-se trabalhar bem como o usuário que utilizará o sistema e sua respectiva senha.

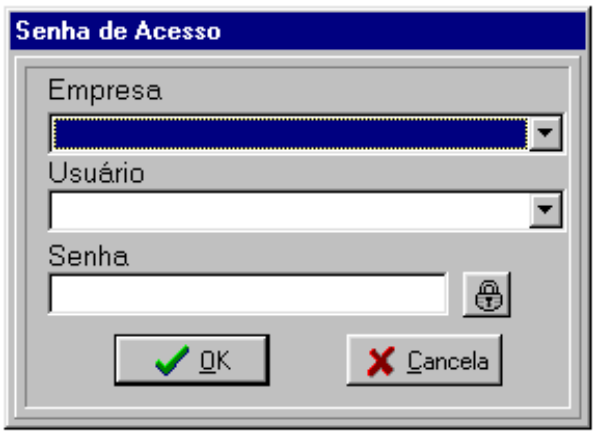

Figura 6.7 – Tela de *Login*

Na figura 6.8 encontra-se o menu geral do protótipo. No menu Arquivo, encontram-se os cadastros gerais do sistema, como o cadastro de estados, cidades, empresa, etc. No menu Manutenção, encontram-se os formulários de Solicitação de OS e Programação de OS. No menu Consultas, encontram-se as consultas do sistema.

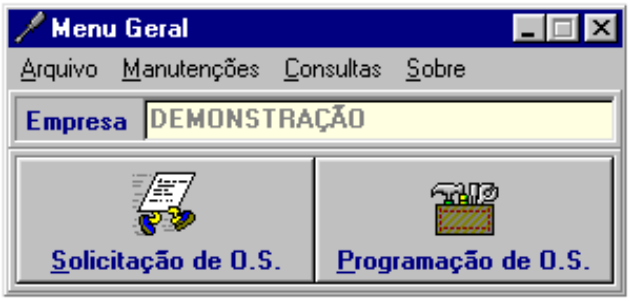

Figura 6.8 – Menu Geral

A figura 6.9 corresponde ao cadastro de empresas do protótipo. Todos os cadastros do protótipo, inclusive o formulário de solicitação de OS, seguem este padrão de tela. A barra de ferramentas, encontrada na parte superior da figura 6.9, encontra-se detalhada na figura 6.10.

Figura 6.9 – Cadastro de Empresas

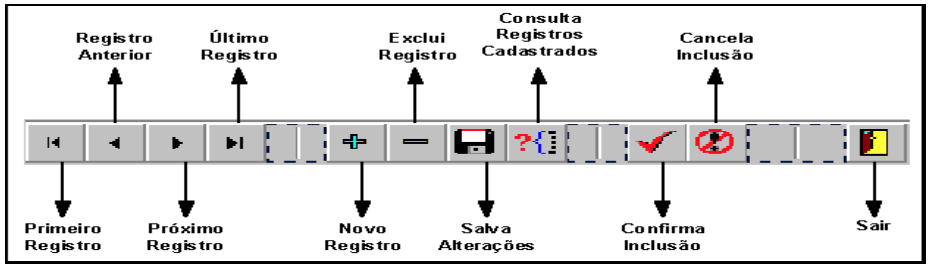

Figura 6.10 – Barra de Ferramentas dos Formulários do Protótipo

Na figura 6.11 encontra-se demonstrado o formulário de Solicitação de Serviço. Os usuários dos serviços do setor de manutenção utilizam este formulário para solicitar a execução de suas OS.

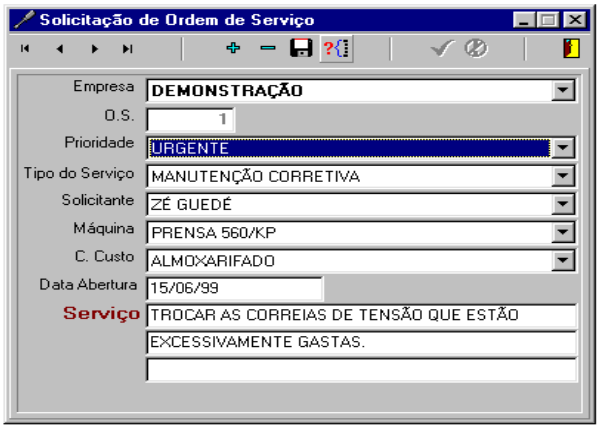

Figura 6.11 – Formulário de Solicitação de Ordens de Serviço

Uma vez solicitada a OS, automaticamente a mesma aparece no formulário de Programação de OS, demonstrada na figura 6.12. Através deste módulo, o responsável pelo setor de manutenção da empresa procede o encaminhamento da OS, programando-a ou cancelando-a, conforme sua avaliação. Este módulo também permite a listagem da OS ou a reabertura de OS baixadas ou canceladas indevidamente.

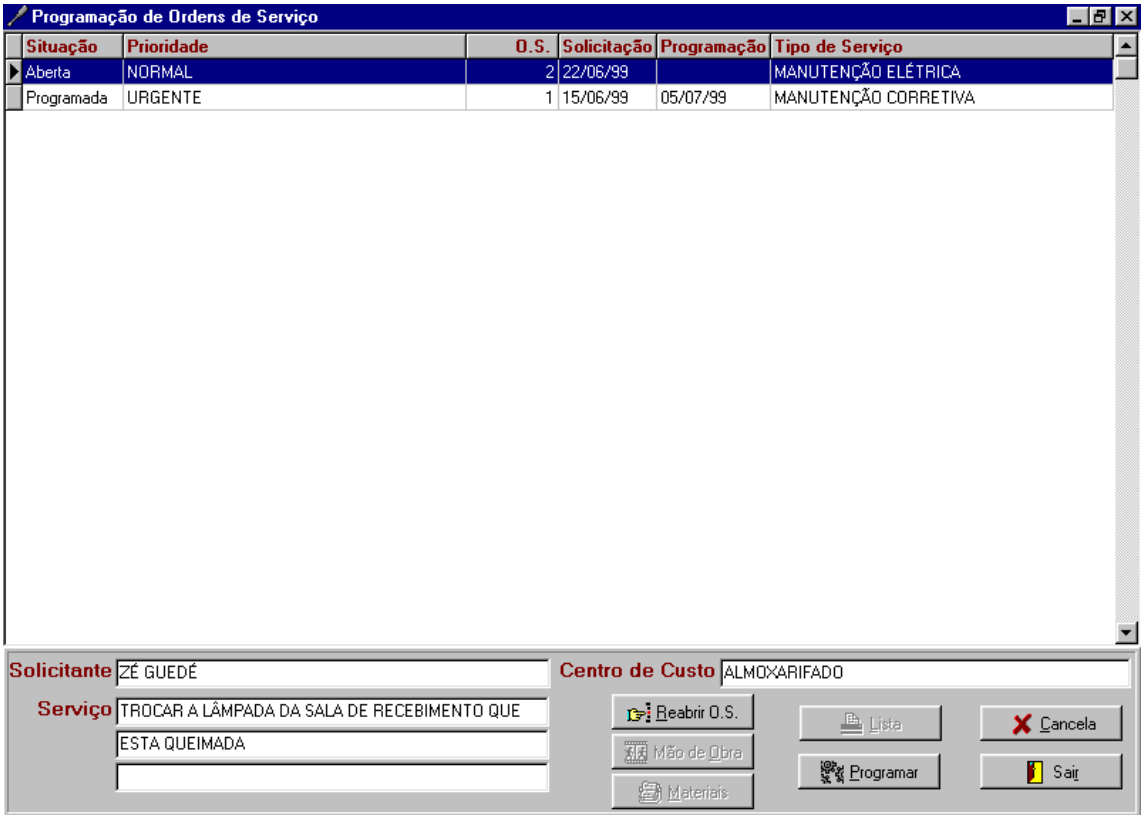

Figura 6.12 – Programação de Ordens de Serviço

Também é através deste módulo que se procedem os apontamentos de Mão-de-Obra (através do formulário demonstrado na figura 6.13) e os apontamentos de Materiais consumidos (através do formulário demonstrado na figura 6.14) da OS.

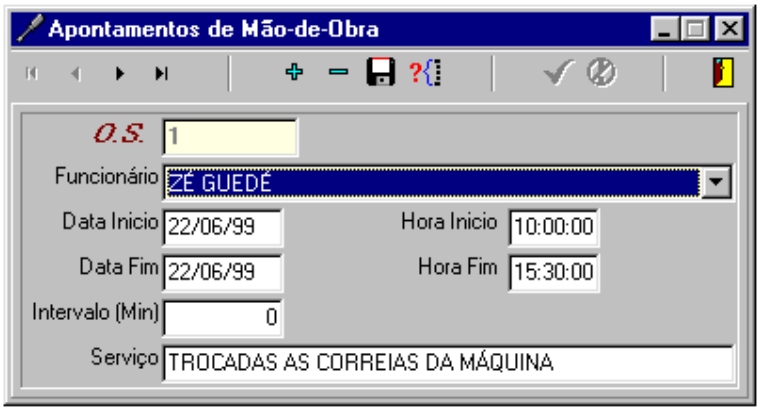

Figura 6.13 – Apontamentos de Mão-de-Obra da Ordem de Serviço

Figura 6.14 – Apontamentos de Materiais Consumidos da Ordem de Serviço

Uma das consultas disponíveis no protótipo, a de "OS por Solicitante", pode ser visualizada na figura 6.15.

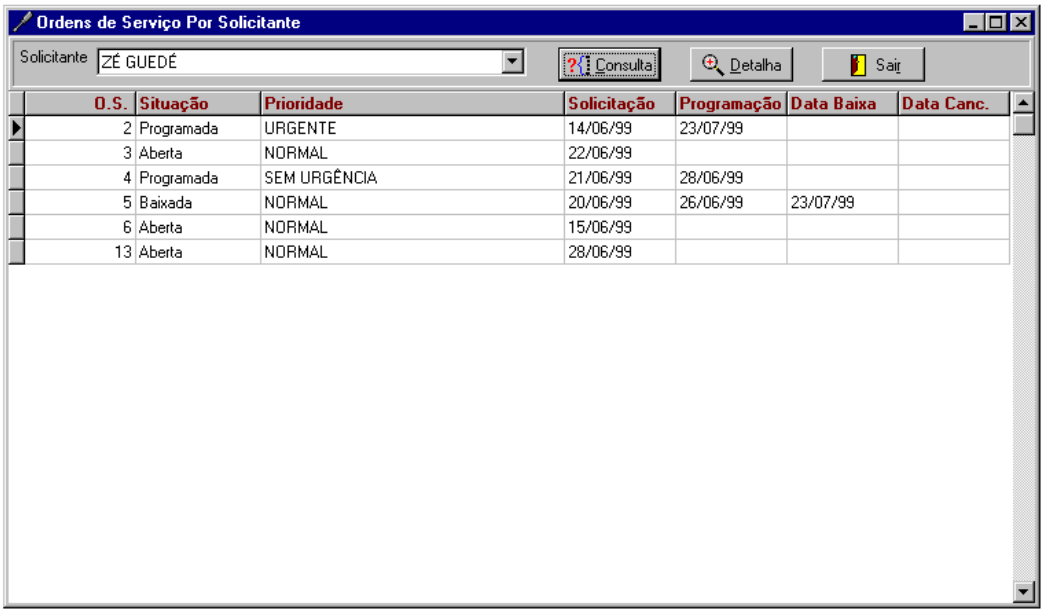

Figura 6.15 – Consulta Ordens de Serviço por Solicitante

7 CONCLUSÕES

Neste capítulo serão apresentadas as conclusões finais sobre o trabalho e sugestões para aperfeiçoamento do mesmo.

7.1 Considerações sobre o Modelo de Dados Relacional do Caché

O modelo de dados relacional do Caché consiste basicamente em mapear as estruturas de armazenamento em árvore na forma de tabelas. Esse mapeamento fica armazenado em um dicionário de dados que se responsabiliza por interpretar as solicitações de acesso relacional ao Caché.

Um problema sério encontrado no modelo relacional do Caché diz respeito a ferramenta de definição de tabelas do banco. A ferramenta de definição de tabelas, denominada de %msql, possui uma interface caracter desagradável, complexidade conceitual e uma estrutura de *help* que deixa muito a desejar.

Outro problema encontrado diz respeito à interface ODBC do Caché com o Delphi. Vale salientar que este não é um problema do Caché propriamente dito, mas sim do Delphi, que não reconhece algumas construções relacionais do Caché bem como certos *constraints* de dados e chaves.

7.2 Considerações sobre o Modelo 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.

Uma grande vantagem do modelo de objetos do Caché consiste que, ao se definir um dicionário de Classes, o sistema Caché gera implicitamente um dicionário de dados relacional na forma de tabelas. Dessa forma, de um único processo de definição de dados obtêm-se ambos os modelos de dados, relacional e orientado a objeto. As mesmas estruturas de dados podem ser acessadas por ambos os modelos, concorrentemente, sem problema algum.

Outra vantagem do modelo de objetos do Caché diz respeito à ferramenta de definição de classes: o Caché Object Architect. Além de possuir uma interface GUI agradável e ser conceitualmente simples, o Caché Object Architect possui *Wizards* para agilizar os processos de criação de Classes e *Queries*. Infelizmente, um problema encontrado no Caché Object Architect diz respeito à falta de uma estrutura de *help On-Line*.

O único problema encontrado com relação ao modelo de objetos do Caché foi a falta de componentes para acesso nativo dos objetos Caché através do Delphi. Existe um componente com essas características, mas ele só pode ser utilizado com Microsoft Visual Basic. A InterSystems já está trabalhando em um componente desse porte para o Delphi, que deve ser disponibilizado nos próximos *builds* da versão 3.1 do Caché.

7.3 Considerações sobre a Implementação do Protótipo

Os programas do protótipo implementados acessando os dados de maneira relacional foram mais rapidamente desenvolvidos em função da gama de componentes para interface de banco de dados relacionais que o Delphi oferece. Apesar da interface ODBC entre o Delphi e o Caché apresentar alguns problemas, como já citado no item 6.8.1, todo o acesso relacional ao Caché foi realizado através dos componentes de acesso a banco de dados do Delphi.

Sem dúvida, os programas do protótipo que acessavam os dados de maneira orientada a objetos tiveram um processo de implementação mais "trabalhoso". Vale salientar que este trabalho excessivo ocorreu em virtude da falta de componentes para acesso direto dos objetos Caché através do Delphi, o que fez com que esse processo de acesso tivesse que ser totalmente programado na forma de interface com rotinas COS armazenadas no banco. Essa interface foi realizada através do Visual M, um subsistema do sistema Caché que permite a interação entre ferramentas de desenvolvimento (como o Delphi e o Visual Basic) com os dados e rotinas do Caché, possibilitando a criação de um ambiente Cliente/Servidor.

O Visual M é composto de duas partes:

a) Visual M Server: O Visual M Server é um processo no Caché que habilita e gerencia as conexões requisitadas pelos clientes, tanto remotos quanto locais;

b) Visual M Client: O Visual M Client é um componente Active X que deve ser adicionado a uma ferramenta gráfica de desenvolvimento. Ele permite a troca de dados e a execução de comandos COS em um servidor local ou remoto.

Em resumo, o acesso aos dados na forma de objetos no protótipo foi realizado através de rotinas escritas em COS, armazenadas no próprio Caché e executadas através do **Visual M**.

Outro ponto que merece destaque na implementação do protótipo diz respeito ao processo de manutenção do mesmo. Apesar do protótipo ter apresentado baixos índices de manutenção, em virtude de não ter sido posto em utilização oficialmente, o processo de manutenção dos programas implementados com os dados sendo acessados de maneira orientada a objetos foi mais fácil. Isso deve-se ao fato dos processos de manipulação dos dados no modelo orientado a objetos estarem concentrados nas próprias classes do modelo. Dessa forma, sabe-se de antemão o local onde provavelmente deverá ser feita a manutenção: no dicionário de classes.

7.4 Considerações sobre os Critérios de Avaliação

Com relação à avaliação da aplicação desenvolvida nesse trabalho pode-se dizer que ambos os modelos de dados (relacional e orientado a objetos) contemplam e atendem as necessidades da aplicação. Dessa forma, a maneira como a aplicação foi implementada, englobando ambas as tecnologias de modelagem de dados, atende às premissas dos critérios propostos.

Sobre o SGBD utilizado na implementação do protótipo, o Caché, também pode-se dizer que o mesmo atendeu de maneira satisfatória o questionário de avaliação proposto, atingindo uma gama considerável de respostas positivas.

7.5 Considerações Finais sobre o Trabalho Desenvolvido

A evolução dos SGBD surgiu em vínculo à própria evolução da Ciência da Computação. Computadores cada vez mais poderosos aliados à crescente necessidade do atendimento maciço de aplicações complexas e da interoperabilidade entre os vários setores da organização-empresa fez surgir as premissas e garantiu a evolução dos SGBD.

Nesse contexto, não obstante, todos os SGBD tiveram seus momentos de ascensão e glória. Estabeleceram-se no mercado agradando a analistas e desenvolvedores de software em geral. Tornaram-se ferramentas de desenvolvimento importantes e ocupam um lugar privilegiado até hoje.

Os SGBDR, apesar do contestado problema de desempenho, firmaram-se como os modelos de dados predominantes no mercado. Dominam grande massa das aplicações e armazenam considerável gama de dados em seus produtos. No atual momento, os BDR estão incorporando características de OO sobre seus modelos a fim de ampliar o leque de aplicações que os mesmos atingem atualmente.

Em contrapartida, os SGBDOO vem promovendo um processo de evolução constante, na forma de produtos e ferramentas de desenvolvimento. A tecnologia da OO firma-se com o aproximar do milênio. A indiscutível ascensão da Internet e a propagação sadia da modelagem OO impulsionam os SGBDOO.

A união dos conceitos de OO e de SGBD torna os SGBDOO poderosas ferramentas no processo de desenvolvimento de sistemas. Os ganhos em produtividade do modelo OO unem-se à gerência de persistência dos SGBD, contemplando qualquer tipo de necessidade dos desenvolvedores de software.

No entanto, a escolha do melhor modelo de dados para determinado software aplicativo, apesar de parecer uma tarefa simples e sem importância, pode trazer consideráveis problemas de implementação e de modelagem de dados durante o processo de desenvolvimento da aplicação; se o modelo de dados escolhido não atender às premissas e necessidades do software aplicativo.

A aplicação de critérios de avaliação, na forma de um roteiro de perguntas ou questionário formal, viabiliza e facilita o processo de escolha de qual o melhor modelo de dados para determinada aplicação, levando em consideração tanto os aspectos da aplicação envolvida quanto as características do software de SGBD propriamente dito.

Uma limitação dos critérios de avaliação apresentados nesse trabalho consiste no fato dos mesmos não auxiliarem diretamente o processo de escolha de qual o melhor produto de software de SGBD, dentre os produtos disponíveis no modelo de dados escolhido, que deve ser utilizado no processo de desenvolvimento da aplicação. Os critérios de avaliação aqui apresentados visam a auxiliar a escolha de qual modelo de dados atenderá melhor as necessidades da aplicação.

O software de gerenciamento de banco de dados Caché apresenta características interessantes e que estão garantindo sua popularização em meio a comunidade computacional. Seu modelo de dados multidimensional, aliado as suas diversas opções de acesso aos dados e a seu poderoso modelo de objetos garantem performance e escalabilidade para o banco, mesmo quando submetido a situações críticas em transações.

No protótipo desenvolvido, as maiores dificuldades encontradas foram com relação à interface entre o modelo de dados do Caché, tanto relacional quanto orientado a objetos, e à ferramenta de desenvolvimento Delphi. A interface ODBC entre o Delphi e o modelo relacional do Caché apresenta alguns problemas consideráveis. Já, para o modelo orientado a objetos do Caché, faltam componentes para acesso nativo aos objetos instanciados em disco que, sem dúvida, viriam a facilitar e otimizar o processo de implementação.

Algo considerável no protótipo desenvolvido diz respeito à coesão harmoniosa entre o modelo de dados relacional e o modelo de dados orientado a objetos na mesma aplicação. Isso leva a crer que um SGBD que apresente o poder e facilidade de ambas as tecnologias em um único sistema arquitetônico pode em muito auxiliar e facilitar o processo de desenvolvimento de sistemas. Consultas podem facilmente ser construídas na forma de instruções SQL assim como estruturas de dados complexas podem ser modeladas com o auxílio da tecnologia OO do banco. Os SGBD que incorporarem tanto as características relacionais quanto as orientadas a objeto em um único sistema de gerenciamento terão, com certeza, importante papel num futuro não muito distante.

7.6 Sugestões para Futuros Trabalhos

A título de sugestão, relacionou-se abaixo alguns tópicos que poderiam ser melhor desenvolvidos, a fim de completar o exposto nesse trabalho:

a) completar os critérios de avaliação apresentados nesse trabalho com critérios de avaliação que auxiliem no processo de escolha do melhor produto de software de SGBD, dentre os disponíveis no referido modelo de dados escolhido;

b) completar os critérios de avaliação apresentados nesse trabalho adicionando características de avaliação para banco de dados inteligentes;

c) aplicar os critérios de avaliação aqui apresentados a uma gama considerável de aplicações e SGBD, a fim de apresentar uma demonstração prática do funcionamento desses critérios;

d) apresentar um estudo sobre o funcionamento dos SGBD que contemplam tanto o modelo relacional quanto o modelo orientado a objetos de dados em um único sistema de gerenciamento de dados.

ANEXO 1 - A API do *ClassDictionary*

A API do ClassDictionary tem por objetivo compilar e manipular objetos e definições de classe. Esse utilitário está implementado em COS na forma de uma série de programas armazenados na biblioteca **%apiOBJ**.

A1.1 Alguns Programas da API do *ClassDictionary*

Nos tópicos a seguir estão descritos alguns dos programas da API do *ClassDictionary*.

A1.1.1 Object Command Shell

Sintaxe: Do Shell^%apiOBJ

É executado junto aos terminais de emulação Caché (como o **Caché *Programmer*** e o **NetTERM**¸ por exemplo) e permite a utilização de código COS como linhas de comando.

A1.1.2 LoadFile

Sintaxe: Do LoadFile^%apiOBJ(**file**, **flag**, **.erro**)

Importa um arquivo CDL para o dicionário de Classes. **File** é o nome do arquivo CDL a ser carregado e opcionalmente compilado, dependendo do valor de **flag**. O conteúdo de **flag** é explicado na seção A1.2 e o conteúdo de **erro** na seção A1.3.

A1.1.3 LoadDir

Sintaxe: Do LoadDir^%apiOBJ(**dir**, **flag**, **.erro**)

Importa os arquivos CDL de um diretório para o dicionário de Classes. **Dir** é o nome do diretório onde se encontram os arquivos CDL a serem carregados e opcionalmente compilados, dependendo do valor de **flag**. O conteúdo de **flag** é explicado na seção A1.2 e o conteúdo de **erro** na seção A1.3.

A1.1.4 Compile

Sintaxe: Do Compile^%apiOBJ(**classname**, **flag, .erro**)

Compila a classe **classname**. Assume-se que **classname** é uma classe definida no dicionário de classes. O conteúdo de **flag** é explicado na seção A1.2 e o conteúdo de **erro** na seção A1.3.

A1.1.5 CompileSystem

Sintaxe: Do CompileSystem^%apiOBJ(**flag, .erro**)

Compila as classes de sistema do Caché. O conteúdo de **flag** é explicado na seção A1.2 e o conteúdo de **erro** na seção A1.3.

A1.1.6 CompileAll

Sintaxe: Do CompileAll^%apiOBJ(**flag, .erro**)

Compila todas as classes (de usuário e de sistema) definidas no dicionário de classes do NameSpace atual. O conteúdo de **flag** é explicado na seção A1.2 e o conteúdo de **erro** na seção A1.3.

A1.1.7 ExportCDLList

Sintaxe: Do ExportCDLList^%apiOBJ(**classlist, file, flag, .erro**)

Exporta todas as classes descritas em **classlist** para o arquivo CDL **file** . O nome das classes em **classlist** devem estar separados por vírgula. O conteúdo de **flag** é explicado na seção A1.2 e o conteúdo de **erro** na seção A1.3.

A1.1.8 ExportJavaList

Sintaxe: Do ExportJavaList^%apiOBJ(**classlist, Dir, flag, .erro**)

Exporta todas as classes descritas em **classlist** para o diretório **Dir**, gerando um arquivo para cada classe exporta com a extensão ".java". O nome das classes em **classlist** devem estar separados por vírgula. O conteúdo de **flag** é explicado na seção A1.2 e o conteúdo de **erro** na seção A1.3.

A1.1.9 Delete

Sintaxe: Do Delete^%apiOBJ(**classname**)

Apaga a classe **classname** do dicionário de classes do NameSpace atual.

A1.1.10 DeleteAll

Sintaxe: Do DeleteAll^%apiOBJ()

Apaga todas as classes do dicionário de classes do NameSpace atual.

A1.1.11 GetVersion

Sintaxe: Do GetVersion^%apiOBJ()

Retorna a versão instalada do Caché Objects.

A1.1.12 Method

Sintaxe 1: Do Method^%apiOBJ(**class**,**oref**,**method**,**parm1**,...,**parm9**)

Sintaxe 2: Set X=$$Method^%apiOBJ(**class**,**oref**,**method**,**parm1**,...,**parm9**)

Este programa é utilizado para executar indiretamente um método de um objeto em particular. Ele executa o método **method** do objeto da classe **class** instanciado em **oref** passando **parmN** como parâmetros. Na sintaxe 2, o resultado da execução do método é retornado para a variável **X.**

A1.2 O Argumento "flag"

Flag é uma cadeia de caracteres que tem objetivo definir parâmetros de execução para os programas do ClassDictionary. Os parâmetros permitidos para **flag** podem ser visto na tabela A1.1.

Parâmetro	Descrição	Objetivo
a	*Application*	Específico para compilar classes de usuário
c	*Compile*	Nos programas de importação de classes este parâmetro especifica que as classes também devem ser compiladas. Este parâmetro é ignora pelo programa *Compile*.
d	*Display*	Determina que o progresso dos programas de compilação e importação devem ser mostrados na tela.
r	*Recursive*	Determina que todas as superclasses especificadas para a classe em questão também devem ser compiladas. Durante a compilação de classes de sistema, este parâmetro deve fazer parte do conteúdo de **flag** .

s	*System*	Compila as classes de sistema.
u	*Update*	Compila somente as classes alteradas desde de o último processo de compilação.

Tabela A1.1 - Os Parâmetros de Flag

Exemplos:

Do LoadFile^%apiOBJ(“C:\MyClasses\Car.cdl”, “cr”)

Do LoadDir^%apiOBJ(“C:\MyClasses”, “c”)

A1.3 O Argumento “erro”

O argumento erro retorna de **1** à **n** mensagens de erro no formato:

erro = n
erro(1) = ”Mensagem de Erro 1”
erro(2) = ”Mensagem de Erro 2”
...
erro(n) = ”Mensagem de Erro n”

ANEXO 2 - O Macro Pré-Processador

O Macro Pré-Processador é responsável pela "tradução" do código COS programado em seu código de execução *Run-time*. O código COS descrito na tabela A2.1 instancia um objeto da classe "CEstado" e executa o método "Set". Na coluna da esquerda observa-se o código fonte COS programado. Na Coluna da direita tem-se o código fonte COS "Macro-Processado", ou seja, traduzido para execução em *Run-time*. A principal diferença existente é que o código "Macro-Processado" é um pouco mais complexo que o código COS fonte padrão. O código "Macro-Processado" é muito similar, em sintaxe, à linguagem M padrão.

Código Fonte COS	Código COS "Macro-Processado"
New Instancia	**New** Instancia
Set Instancia=##class(CEstado).%New()	**Set** Instancia=$$%New^ooCEstadoR1()
Do Instancia.Set(P0)	**Do** InvokeMethod^%occRun("",Instancia,"Set",P0)
Quit	**Quit**

Tabela A2.1 - Código COS

ANEXO 3 - O Caché Object Architect

O Caché Object Architect (figura A3.1) é um ambiente com interface GUI que permite a manipulação completa das classes do Caché Objects. O Caché Object Architect possui *Wizards* para facilitar a criação de Classes, Atributos, Métodos e *Queries*. Também é possível exportar, através do Caché Object Architect, definições de classes Caché como definições de classes em C ou Java.

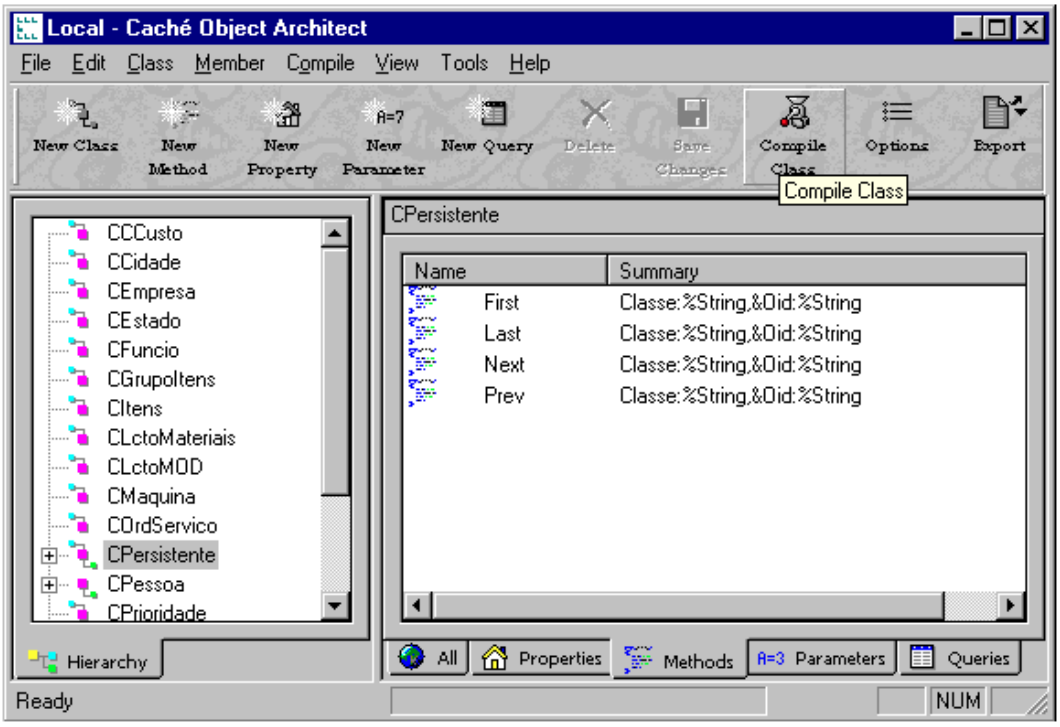

Figura A3.1 - O Caché Object Architect

ANEXO 4 - *Caché Class Definition Language*

O *Caché Class Definition Language* (CDL) permite a manipulação de classes Caché através de um arquivo texto no formato ASCII. Esse arquivo deve ser carregado e compilado pelos programas do API *ClassDictionary*. O CDL também permite copiar ou mover classes de um NameSpace para outro no Caché.

Os arquivos CDL são muito úteis quando se deseja mover ou copiar classes de um sistema Caché para outro. As classes são simplesmente exportadas para arquivos CDL, importadas para o novo sistema e compiladas. Um arquivo CDL pode conter várias definições de classe.

A4.1 Guia de Referência

Nos tópicos a seguir estão descritas as principais características de sintaxe da linguagem CDL.

A4.1.1 Comentários

Comentários não podem aparecer dentro de alguma parte do arquivo CDL que é formado por código COS, como as implementações de método, por exemplo.

Comentário de Linha: **/ /**

Comentários em Geral: abertura com /* fechamento com *\

A4.1.2 Delimitador de bloco de comandos

Abertura com { fechamento com }

A4.1.3 Comando CREATE CLASS

Sintaxe: CREATE CLASS **classname** { **classkeyword1; classkeyword2; ...**}

O comando CREATE CLASS cria uma nova definição de classe no ClassDictionary. Se a classe que esta sendo definida já existe, um erro ocorre. **Classname** é uma palavra alfanumérica, que começa com uma letra ou o sinal de percentual (%), e define o nome da classe que esta sendo criada. **Classkeywordn** correspondem a palavras reservadas para

definição de classes e são explicadas na seção A4.2. No quadro A4.1 apresenta-se um programa exemplo da criação de uma classe através do comando CREATE CLASS.

```
Create Class Pessoa
{
        super = %RegisteredObject;
        persistent;

        attribute Nome              { type = %String}
        attribute DtNascimento      { type = %Date}
}
```

Quadro A4.1 - Sintaxe do comando CREATE CLASS

A4.1.4 Comando DROP CLASS

Sintaxe: DROP CLASS **classname**

O comando DROP CLASS apaga uma definição de classe no ClassDictionary. **Classname** define o nome da classe à ser apagada.

Exemplo: Drop Class Pessoa

A4.2 Palavras Reservadas para Definição de Classes

Nos tópicos a seguir estão descritas as palavras reservadas utilizadas para definições de classe através da linguagem CDL. Em geral, as palavras reservadas da linguagem CDL definem as características das classes.

A4.2.1 ABSTRACT

Sintaxe: ABSTRACT;

Define que a classe é abstrata.

A4.2.2 FINAL

Sintaxe: FINAL;

Define que a classe é final. Não pode ter subclasses.

A4.2.3 PERSISTENT

Sintaxe: PERSISTENT;

Define que a classe é persistente.

A4.2.4 SUPER

Sintaxe: SUPER = **classname1**, **classname2**, **. . .** , **classnamen**;

Define a(s) superclasse(s) da classe.

A4.2.5 SYSTEM

Sintaxe: SYSTEM;

Define que a classe é uma classe de sistema.

A4.2.6 DESCRIPTION

Sintaxe: DESCRIPTION = **descrição**;

Define uma descrição para a classe.

A4.2.7 PARAMETER

Sintaxe: PARAMETER **ParameterName** { **ParameterDefinition** }

Define os parâmetros da classe.

ParameterName define o nome do parâmetro.

ParameterDefinition possui uma sintaxe similar às definições de classe e possui apenas uma palavra reservada:

a) DEFAULT: Define o nome do atributo cujo o valor é padrão para esse parâmetro ou o valor do parâmetro propriamente dito.

Sintaxe: DEFAULT = **AttributeName** onde:

- **AttributeName:** é o nome do atributo de valor padrão ou um valor propriamente dito (um String ou Integer, por exemplo).

A4.2.8 ATTRIBUTE

Sintaxe: ATTRIBUTE **AtributeName** { **AtributeDefinition** }

Define as propriedades/atributos da classe.

AttributeName define o nome do atributo/propriedade.

AtributeDefinition define as características do atributo. Também possui uma sintaxe muito similar às definições de classe e pode receber (entre várias) as seguintes palavras reservadas:

a) CALCULATED: Define que este é um atributo calculado.

Sintaxe: CALCULATED;

b) FINAL: Define que o atributo é final e não pode ser sobreposto nas subclasses.

Sintaxe: FINAL;

c) PRIVATE: Define que o atributo é privado.

Sintaxe: PRIVATE;

d) PUBLIC: Define que o atributo é público.

Sintaxe: PUBLIC;

e) REQUIRED: Define que o atributo é requerido.

Sintaxe: REQUIRED;

f) TYPE: Define o tipo de dado do atributo. Entre os tipos suportados pelo Caché estão:

- **%Boolean;**
- **%Date;**
- **%Float;**
- **%Integer;**
- **%Status;**
- **%Time;**
- **%String;**
- **%Name**.

Sintaxe: TYPE = **TypeDefinition**;

g) INITIAL: Define uma expressão em COS ou um valor literal que é assumido como valor inicial do atributo.

Sintaxe: INITIAL = **Expression**;

A4.2.9 METHOD

Sintaxe: METHOD **MethodName** (**Pars**) { **MedthodDefinition** }ou

METHOD **MethodName FormalSpec** (**Pars**) { **MedthodDefinition** }onde

Define os métodos da classe.

MethodName define o nome do método.

FormalSpec define o separador padrão para os parâmetros formais desse método. O separador default é a “,” (vírgula).

Pars define os parâmetros formais do método. Esses parâmetros são passados ao método quando de sua chamada. **Pars** consiste de uma cadeia de caracteres no formato: **FormalParameter : FormalType = DefaultValue**¸ separadas pelo **FormalSpec**.

MethodDefinition consiste de uma série de palavras reservadas para definição das características do método e também possui uma sintaxe muito similar as de definições de classe. Dentre suas palavras reservadas estão:

a) CALL: Define o nome do programa que deve ser executado quando da chamada deste método.

Sintaxe: CALL = **ProgramName**;

b) CLASSMETHOD: Define que o método é um método de classe.

Sintaxe: CLASSMETHOD;

c) CODE: Define o código COS do método.

Sintaxe: Code = { **Bloco de Comandos** }

d) FINAL: Define que o método é final e não pode ser sobreposto nas subclasses.

Sintaxe: FINAL;

e) PRIVATE: Define que o método é privado.

Sintaxe: PRIVATE;

f) PUBLIC: Define que o método é publico.

Sintaxe: PUBLIC;

g) RETURNTYPE: Define o tipo de dado do valor retornado pelo método. Quando RETURNTYPE não está presente na declaração do método ou recebe o valor nulo (“”) o método não retorna nenhum valor.

Sintaxe: RETURNTYPE = **DataType**;

h) DESCRIPTION: Define uma descrição para o método

Sintaxe: DESCRIPTION = “**Descrição do Método**”;

A4.3 Uma Definição de Classe em CDL

O quadro A4.2 apresenta a sintaxe de criação da classe **Pessoa** em CDL.

```
Drop Class Pessoa;
Create Class Pessoa
{
        description = " Uma Simples Classe de Exemplo ";
        final;
        super = %RegisteredObject, %Persistent;
        persistent;

        parameter KEYS          {default = "CPF"; }
        parameter INDEX         {default = "Nome"; }

        attribute Nome          { type = %String(MAXLEN=35); required; }
        attribute CPF           { type = %String(MAXLEN=12); required; }
        attribute Telefone      { type = %String(MAXLEN=12); required;
                                        initial= "(000) 000-0000";}

        method editCPF() {
            returntype=%string;
            description = " Um Exemplo de Método";
            final;
            public;
            code = {
           Quit $E(..CPF,1,3)_"."_$E(..CPF,4,6)_"."_$E(..CPF,7,9)_"-"_$E(..CPF,10,11)
                        }
        }

        method print() {
            description = "Outro Exemplo de Método";
            final;
            public;
            code = {
            Write ..editCPF()
            Write ..Nome
            Quit
                        }
        }
}
```

Quadro A4.2 – Uma Definição de Classe em CDL

ANEXO 5 – O Dicionário de Dados do Protótipo

Encontra-se descrito a seguir o dicionário de dados do protótipo, contendo o nome da tabela, uma descrição sobre a mesma e a estrutura de seus atributos (tipo, tamanho e descrição).

Tabela: **TEstado** Descrição: **Tabela de Estados**

Campo	Tipo	Tamanho	Descrição
IDEstado	Integer	5,0	Identificador (ID do Objeto)
Sigla	String	2,0	Sigla do Estado
Nome	String	35,0	Nome do Estado

Tabela: **TCidade** Descrição: **Tabela de Cidades**

Campo	Tipo	Tamanho	Descrição
IDCidade	Integer	5,0	Identificador (ID do Objeto)
IDEstado	Integer	5,0	Identificador do Estado da Cidade
Nome	String	35,0	Nome da Cidade
CEP	String	5,0	Cep da Cidade

Tabela: **TGrpItens** Descrição: **Tabela de Grupos de Itens**

Campo	Tipo	Tamanho	Descrição
IDGrupo	Integer	5,0	Identificador (ID do Objeto)
IDEmpresa	Integer	5,0	Identificador da Empresa do Grupo
Descricao	String	35,0	Descrição do Grupo

Tabela: **TUnidade** Descrição: **Tabela de Unidades dos Itens**

Campo	Tipo	Tamanho	Descrição
IDUnidade	Integer	5,0	Identificador (ID do Objeto)
Descricao	String	35,0	Descrição da Unidade
Abreviatura	String	5,0	Abreviatura da Unidade

Tabela: **TTipoServico** Descrição: **Tabela de Tipos de Serviços**

Campo	Tipo	Tamanho	Descrição
IDTipoServico	Integer	5,0	Identificador (ID do Objeto)
Descricao	String	35,0	Descrição do Tipo de Serviço

Tabela: **TPrioridade** Descrição: **Tabela de Prioridades dos Serviços**

Campo	Tipo	Tamanho	Descrição
IDPriori	Integer	5,0	Identificador (ID do Objeto)
Descricao	String	35,0	Descrição da Prioridade
Nível	Integer	5,0	Nível da Prioridade

Tabela: **TItens** Descrição: **Tabela de Itens**

Campo	Tipo	Tamanho	Descrição
IDItem	Integer	5,0	Identificador (ID do Objeto)
IDEmpresa	Integer	5,0	Identificador da Empresa do Item
Descricao	String	35,0	Descrição do Item
IDUnidade	Integer	5,0	Identificador da Unidade do Item
IDGrupo	Integer	5,0	Identificador do Grupo do Item

Tabela: **TEmpresa** Descrição: **Tabela de Empresas**

Campo	Tipo	Tamanho	Descrição
IDEmpresa	Integer	5,0	Identificador (ID do Objeto)
IDCidade	Integer	5,0	Identificador da Cidade da Empresa
RazaoSocial	String	35,0	Nome da Empresa
Endereco	String	35,0	Endereço da Empresa
Bairro	String	35,0	Bairro da Empresa
Telefone	String	10,0	Telefone da Empresa
NomeFantasia	String	20,0	Nome Fantasia da Empresa
MascCCusto	String	20,0	Máscara dos C. Custos da Empresa
NivCCusto	String	2,0	Número de Níveis da Máscara dos C. Custo
MascMaquina	String	20,0	Máscara das Máquinas da Empresa
NivMaquina	String	2,0	Número de Níveis da Máscara das Máquinas

Tabela: **TCCusto** Descrição: **Tabela de Centros de Custo**

Campo	Tipo	Tamanho	Descrição
IDCCusto	Integer	5,0	Identificador (ID do Objeto)
IDEmpresa	Integer	5,0	Identificador da Empresa do C. Custo
Mascara	String	20,0	Máscara do C. Custo
Descricao	String	35,0	Descrição do C. Custo
Tipo	String	1,0	Tipo do Centro de Custo. Pode ser: 1 – Comum 2 – Auxiliar 3 – Produtivo 4 – Administrativo 5 – Diferido
Situacao	String	1,0	Situação do C. Custo. Pode Ser: 0 – Inativo 1 – Ativo

Tabela: **TMaquina** Descrição: **Tabela de Maquinas/Equipamentos**

Campo	Tipo	Tamanho	Descrição
IDMaquina	Integer	5,0	Identificador (ID do Objeto)
IDEmpresa	Integer	5,0	Identificador da Empresa da Máquina
IDCCusto	Integer	5,0	Identificador do C. Custo da Máquina
Mascara	String	20,0	Máscara da Máquina
Descricao	String	35,0	Descrição da Máquina
Tipo	String	1,0	Tipo da Máquina. Pode ser: 1 – Fixa 2 – Móvel

Tabela: **TFuncio** Descrição: **Tabela de Funcionários**

Campo	Tipo	Tamanho	Descrição
IDFuncio	Integer	5,0	Identificador (ID do Objeto)
IDEmpresa	Integer	5,0	Identificador Empresa do Funcionário
IDCCusto	Integer	5,0	Identificador C. Custo do Funcionário
IDCidade	Integer	5,0	Identificador Cidade do Funcionário
Nome	String	35,0	Nome do Funcionário
Endereco	String	35,0	Endereço do Funcionário
Bairro	String	35,0	Bairro do Funcionário
Telefone	String	10,0	Telefone do Funcionário

Tabela: **TOrdServico** Descrição: **Tabela de Ordem de Serviço**

Campo	Tipo	Tamanho	Descrição
IDOrdServico	Integer	5,0	Identificador (ID do Objeto)
IDEmpresa	Integer	5,0	Empresa da Ordem de Serviço
IDPriori	Integer	5,0	Prioridade da Ordem de Serviço
IDTipoServico	Integer	5,0	Tipo de Serviço da Ordem de Serviço
IDFuncio	Integer	5,0	Funcionário que solicitou o Serviço
IDMaquina	Integer	5,0	Máquina p/qual será feito o Serviço
IDCCusto	Integer	5,0	C. Custo p/qual será feito o Serviço
Situacao	String	1,0	Situação da O.S. (Ordem de Serviço) 0 – Aberta/Solicitada 1 – Programada 2 – Baixada 3 – Cancelada
DtSolicitacao	Date	5,0	Data de solicitação da O.S.
DtProgramacao	Date	5,0	Data de Programação da O.S.
DtBaixa	Date	5,0	Data de Baixa da O.S.
DtCancelamento	Date	5,0	Data de Cancelamento da O.S.
DescServico	String	150,0	Descrição Serviço a Ser Executado. (Como o Caché não possuí um tipo de dado "Memo", foram criados 3 campos físicos do tipo "string", com 50 caracteres cada.)

Tabela: **TLctoMOD** Descrição: **Tabela de apontamentos de Mão - de - Obra da Ordem de Serviço**

Campo	Tipo	Tamanho	Descrição
IDLctoMOD	Integer	5,0	Identificador (ID do Objeto)
IDOrdServico	Integer	5,0	Ordem de Serviço desse Apontamento
IDFuncio	Integer	5,0	Funcionário que executou esse Serviço
HoraInicio	Time	5,0	Hora de Início do Serviço
HoraFim	Time	5,0	Hora de Término do Serviço
DataInicio	Date	5,0	Data de Início do Serviço
DataFim	Date	5,0	Data de Término do Serviço
MinIntervalo	Integer	5,0	Minutos de Intervalo do Serviço
TempoTotal	Integer	5,0	Tempo Total do Serviço (Em segundos)
ServExecutado	String	50,0	Serviço Executado

Tabela: **TLctoMateriais** Descrição: **Tabela de apontamentos de Materiais da Ordem de Serviço**

Campo	Tipo	Tamanho	Descrição
IDLctoMOD	Integer	5,0	Identificador (ID do Objeto)
IDOrdServico	Integer	5,0	Ordem de Serviço desse Apontamento
IDItem	Integer	5,0	Funcionário que executou esse Serviço
ValorMaterial	Float	5,3	Valor Unitário do Material
QtdeMaterial	Float	5,3	Quantidade de Material Utilizada

GLOSSÁRIO

API	- (*Application Program Interface*) um formato de linguagem e mensagem utilizado por um programa aplicativo para comunicar-se com outro programa que lhe fornece serviços.
Desenvolvedores	- Pessoas que trabalham com o desenvolvimento de software.
Interface	- Conexão e interação entre o hardware e o software e o usuário.
Internet	- Rede mundial de computadores conectados através de linha telefônica.
Mumps	- Linguagem de Programação, desenvolvida no laboratório de ciências da computação do Massachusetts General Hospital, para processar dados orientados por cadeia de caracteres e suportar bases de dados de acesso aleatório. A sigla Mumps corresponde a abreviatura de **M**assachusetts General Hospital **U**tility **M**ulti-**P**rogramming **S**ystem.
Padrão de Mercado	- Produto ou serviço que se tornou padrão por ser utilizado pela maioria do mercado disponível.
Rotina	- Nome geralmente utilizado para referenciar um programa escrito em Caché Object Script.
Swizzled	- Processo que converte a representação das relações em disco dos objetos em ponteiros de memória para todos os objetos relacionados que estão em memória;

REFERÊNCIAS BIBLIOGRÁFICAS

[ABR95] ABREU, Mauricio; MACHADO, Felipe Nery R.. **Projeto de banco de dados: uma visão prática.** São Paulo : Érica, 1995.

[AGO97] D'AGOSTINI, Júlio César. **Seminário catarinense de manutenção – apostila de curso.** Joinville, 1997.

[ABN96] ABNT. **NBR 13596/1996 – Qualidade de software produto.** São Paulo : ABNT, 1996.

[BAR96] BARRETO Jr, José. **Qualidade de software.** Endereço Eletrônico : http://www.BARRETO.com.br/qualidade/index.htm, 1999.

[CEL97] CELKO, Joe; CELKO, Jackie. **Verdades e mentiras sobre banco de dados de objetos.** Byte Brasil, São Paulo, n. 10, p.86-89, out.1997.

[COA91] COAD, Peter; YOURDON, Edward. **Análise baseada em objetos.** Rio de Janeiro : Campus, 1991.

[DAT90] DATE, C.J. **Introdução a sistemas de banco de dados, 4ª ed.** Rio de Janeiro : Campus, 1990.

[DIA96] DIAS, Márcio de Sousa. **Conceitos básicos.** Endereço Eletrônico : http://www.nce.ufrj.br/~marciosd/pfinal/cap2b.html, 1999.

[GRE98] GREIN, Dirceu; TOMIFA, Katia Francelino. **Sistemas de gerenciamento de banco de dados orientado a objetos.** Endereço Eletrônico : http://www.pr.gov/celepar/batebyte/bb78/colunado.htm, 1999

[GRO96] GROHS, Hans J. **Disciplina de Programação III – SQL. Curso de Ciências da Computação – Bacharelado.** FURB : Anotações em Sala de Aula, 1996.

[HUM95] HUMPHREY, Watts S. **A Discipline for software engineering.** Addison Wesley, 1995.

[INT97a] INTERSYSTEMS Corporation. **Caché programming guide.** Estados Unidos : InterSystems Corporation, 1997.

[INT97b] INTERSYSTEMS Corporation. **Object script language reference.** Estados Unidos : InterSystems Corporation, 1997.

[INT98a] INTERSYSTEMS Corporation. **Class reference.** Estados Unidos : InterSystems Corporation, 1998.

[INT98b] INTERSYSTEMS Corporation. **User's guide.** Estados Unidos : InterSystems Corporation, 1998.

[INT99] INTERSYSTEMS Corporation. **Caché.** Endereço Eletrônico : http://www.intersystems.com, 1999.

[IPS99a] IPSUM. **A tecnologia M.** Endereço Eletrônico : http://www.ipsum.com.br/tecm.htm, 1999.

[IPS99b] IPSUM. **Caché – a nova geração de bancos de dados.** Endereço Eletrônico : http://www.ipsum.com.br/cache.htm, 1999.

[LOY97] LOYOLA, Afonso Celso Martins. **Delphi 3 - método rápido.** Rio de Janeiro : Infobook, 1997.

[KER94] KERN, Vinícius. **Banco de dados relacionais.** São Paulo : Érica, 1994.

[KHO94] KHOSHAFIAN, Setrag. **Banco de dados orientado a objeto.** Rio de Janeiro : Infobook, 1994.

[KOR97] KORTH, Henry F., SILBERSCHATZ, Abraham. **Sistema de banco de dados, 2ª ed.** São Paulo : Makron Books, 1997.

[MAC95] MACIASZEK, Leszek A. **Relational versus object databases: contention or coexistence?** Endereço Eletrônico : http://www.comp.mq.edu.au/courses/comp866/oovsrel.html, 1999.

[MAR95] MARTIN, James; ODELL, James J.. **Análise e projeto orientados a objeto.** São Paulo : Makron Books, 1995.

[OBJ99] OBJECTSTORE. **Object-oriented database systems.** Endereço Eletrônico : http://www.rs6000.ibm.com/resource/aix_resource/Pubs/redbooks/htmlbooks/gg244128.00/4128ch4.html#chap4, 1999.

[RAO94] RAO, Bindu Rama. **Object-oriented databases.** Estados Unidos : McGraw-Hill, 1994.

[RUD93] RUDMIK Andres; McFARLAND Gergory. **Object-oriented database management systems.** Endereço Eletrônico : http://www.dacs.com./techs/OODBMS, 1999.

[RUM94] RUMBAUGH, James; BLAHA, Michael; et al. **Modelagem e projeto baseados em objetos, 5ª ed.** São Paulo : Campus, 1994.

[SAL92] SALGADO, Ana Carolina; FONSECA, Décio; ALBUQUERQUE, Eduardo Simões de; MEIRA, Sílvio Romero de Lemos. **Sistemas hipermídia : hipertexto e banco de dados.** Instituto de Informática da UFRGS : Porto Alegre, 1992.