UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA DA COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Este exemplar corresponde à redação final da tese defendida por Mamede A. M. da Silveira e aprovada pela Comissão Julgadora em 05 / 04 / 91.

Orientador

EXTENSÃO DO NÚCLEO UNICOSMOS PARA SUPORTE À ORIENTAÇÃO POR OBJETO

autor: Mamede Augusto Machado da Silveira
orientadores: Prof. Dr. Léo Pini Magalhães
Prof. Dr. Fernando A.Campos Gomide

Dissertação apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas como parte dos requisitos para título de Mestre em Engenharia Elétrica.

abril de 1991

Dedico este trabalho à minha família -
Denise, Marcela, Lina e aos meus pais.

AGRADECIMENTOS

À minha grande companheira Denise pelo carinho, pela compreensão e pela amor a mim dedicados durante todos os momentos difíceis deste trabalho.

Ao Professor Dr. Léo Pini pela orientação, paciência e companherismo tão fundamental no desenvolvimento deste trabalho.

Ao Professor Dr. Fenando Gomide pela orientação e pelo apoio no desenrolar deste trabalho.

Ao caríssimo Mestre Armando Delgado pelas discussões filosóficas e de implementações, sem as quais seria impossível a efetivação deste trabalho, reintero meu agradecimento especial.

À Universidade Federal do Ceará pela oportunidade e pelo apoio financeiro a mim oferecidos.

À CAPES pelo apoio financeiro complementar, sem o qual seria impossível fazer este trabalho.

Ao Projeto Ethos, em especial ao Prof. Mestre Eduardo Tadao Takahashi, pelas ferramentas de suporte oferecidas.

À todos os amigos e amigas que participaram direta ou indiretamente neste trabalho.

RESUMO

Nos últimos anos, o uso de Sistemas de Banco de Dados (SBD) em aplicações comerciais tem se estendido a outras áreas do conhecimento, como por exemplo a área de Projeto. A utilização do computador no desenvolvimento de Projeto tem exigido não somente as facilidades normais oferecidas pelo SBD's, mas outras facilidades mais complexas que não são suportadas por estes. Isto tem provocado o surgimento de linhas de pesquisas para solucionar estas limitações nos atuais SBD's. Existem diversas propostas para o tratamento de informaçoes complexas na área de Projeto. Uma delas chama-se Sistema de Banco de Dados Orientado por Objeto (SBDOO). Dentro desta linha existem várias abordagens, onde uma delas é a adoção de um Núcleo para SBDOO's que dê suporte a Orientação por Objeto nas funções de baixo nível. O presente trabalho tem como objetivo estender um Núcleo denominado UNICOSMOS (UNICAMP/FEE Object Storage Management System) para fornecer tal suporte.

ÍNDICE GERAL

LISTA DE FIGURAS

Capítulo 1

1- Introdução

A área de Banco de Dados emergiu de necessidades de gerência de grandes volumes de dados ligados a aplicações comerciais. Com o passar do tempo a utilização de Sistemas de Banco de Dados estendeu-se também a outras aplicações. Uma destas aplicações tem sido a Engenharia. Diversamente de outras aplicações, em Engenharia existe uma necessidade muito grande de manipulações de dados complexos. Sistemas de Banco de Dados (SBD's) aplicados à Engenharia além das exigências normais (por exemplo : dados persistentes, dados concorrentes, recuperação dos dados, independência dos dados, etc) colocam novas exigências (conceito de abstração, mecanismo de herança, tratamento com objetos complexos, etc), que os tornam diferentes dos SBD's de aplicações comerciais. Tal fato tem contribuído para o surgimento de novas tecnologias em Banco de Dados, que normalmente são denominadas de Sistemas de Banco de Bados não-convencionais. Atualmente uma destas tecnologias que tem sobressaído bastante é a que se denomina de Sistema de Banco de Dados Orientado por Objeto (SBDOO).

Desde 1986 desenvolve-se no Departamento de Engenharia da Computação e Automação Industrial da Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas (DCA/FEE/UNICAMP) um núcleo de Sistema de Gerência de Banco de Dados (SGBD) denominado UNICOSMOS (UNICAMP/FEE Object Storage Management System). Este núcleo trata de toda manipulação de dados a nível mais baixo. Assim, em sua versão inicial o UNICOSMOS oferece facilidades para as tarefas de armazenamento e remoção de seus objetos e atributos, relacionamento entre objetos, etc. Sistemas de Banco de Dados Orientados por Objeto (SBDOO) necessitam no entanto apoio também ao conceito de abstração, mecanismo de herança e Tipos Abstratos de Dados (TAD's), havendo, em geral duas opções para o oferecimento destas facilidades, uma a nível do próprio gerenciador de Banco de Dados e outra a nível de um núcleo de SBD, tema e solução aqui discutidos.

O presente trabalho tem assim o objetivo de estender o UNICOSMOS, para oferecer estas facililidades, conservando a filosofia de um núcleo que dê suporte aos gerenciadores de Banco de Dados (por exemplo: GERPAC [RICA/87], IRIS [WILK/90], ORION [KIM/90] ou O2 [DEUX/90]) nas funções de baixo nível, mantendo assim a uniformidade neste suporte.

Esta dissertação é composta de oito capítulos. O primeiro é esta Introdução onde se descreve, de maneira sucinta, o escopo do trabalho a ser discutido nos próximos capítulos.

O segundo capítulo trata da definição de um Sistema de Banco de Dados convencional e de suas características principais.

Já no terceiro capítulo, são discutidas as características fundamentais de um Sistema de Banco de Dados Orientado por Objeto e também alguns conceitos de abstração utilizados pelo mesmo.

No quarto capítulo é descrita a versão original do UNICOSMOS, que é um Sistema utilizado como um núcleo básico nos gerenciadores de Banco de Dados não-convencional.

No quinto capítulo é proposta uma nova filosofia para o UNICOSMOS, a fim de que este possa dar suporte aos gerenciadores na implementação dos conceitos dos Sistemas discutidos no terceiro capítulo.

O sexto capítulo cita alguns exemplos práticos de como o UNICOSMOS auxiliaria os gerenciadores na implementação de alguns conceitos de abstrações.

No sétimo capítulo é feita uma síntese do trabalho e são apresentadas as conclusões extraídas do presente trabalho.

O oitavo capítulo contém todas as referências bibliográficas utilizadas no desenvolvimento deste trabalho.

Capítulo 2

Sistemas de Banco de Dados

2.1- Introdução

Tem-se observado, com o passar do tempo, que programas em geral guardam uma dependência muito intensa com relação a estruturação de seus dados. Mesmo quando eles utilizam dados semelhantes, requerem arquivos distintos. Isso não caracteriza necessariamente um problema crítico quando se trata o computador como uma simples máquina de calcular. Porém, quando a complexidade e o volume de dados dos programas de aplicação aumentam, isto pode causar problemas e assim a tendência é estabelecer-se, por exemplo uma ou mais Bases de Dados gerenciadas por um Sistema de Software.

As técnicas de gerência de dados têm sofrido várias modificações, conforme os avanços tecnológicos obtidos nas áreas de hardware e software. Pode-se distinguir três fases, que são :

1- Programas totalmente dependentes da estrutura física dos dados;

2- Programas dependentes da organização lógica dos dados;

3- Programas independentes da organização lógica dos dados.

Os programas da primeira fase requeriam que o próprio usuário desenvolvesse não somente o software de aplicação como também o software de estruturação de dados e operações de E/S, sendo essa estruturação feita especificamente para uma determinada aplicação. Caso ocorressem modificações na organização interna dos dados e/ou nos dispositivos de armazenamento, os programas de aplicação teriam que ser, devidamente alterados, recompilados e testados.

Já nos programas da segunda fase existe um isolamento entre

o programa de aplicação e a estrutura física dos dados. Este isolamento é obtido através de um pacote de software normalmente embutido no próprio Sistema Operacional, onde é realizado o interfaceamento Base de Dados/Programa de Aplicação. Esta interface mascara todos os detalhes da organização física dos dados, tornando assim os programas independentes dos dispositivos de armazenamento.

O crescente uso de computadores em várias áreas de aplicação, exige um isolamento ainda maior entre programas e sua estruturação lógica de dados (por exemplo: tabelas de acesso, listas ligadas para armazenar informações, etc). Assim, havendo modificações nos endereços dos dados e/ou nas estratégias de acesso lógico aos mesmos, os programas de aplicação permanecem inalterados, pois toda complexidade da estruturação lógica dos dados requeridas por diferentes usuários tornar-se-ía transparente aos mesmos. Para os usuários finais, os dados estariam representados e estruturados de uma forma simples e bem mais compreensível. Toda esta estruturação é definida através de uma ferramenta formal e precisa denominada de **Modelo de Dados**. Daí então, surgiu o conceito de **Sistemas de Banco de Dados**, que provê recursos para mapear a estrutura definida pelo Modelo de Dados, na estrutura de dados lógicos e físicos implementada no Sistema Computacional.

2.2- Sistemas de Banco de Dados - Definição

Sistema de Banco de Dados (SBD) é definido como uma coleção de dados interrelacionados, armazenados e controlados por um conjunto de programas do Sistema. Estes programas têm como função tornar a utilização da Base de Dados uniforme à aplicação.

A coleção de dados é comumente referenciada como **Base de Dados** e é ela que funciona como fonte ou depósito de informações para os programas de aplicação. Os programas que fazem todo o controle de trânsito de informações são agrupados como o **Sistema de Gerência de Banco de Dados (SGBD)**. Sendo assim, o SGBD tem como função proporcionar um ambiente, conveniente e eficiente, para armazenar e

retirar informações na Base de Dados [KORT/89]. Existe também a figura do **Administrador de Banco de Dados (ABD)**, que é a pessoa ou o grupo de pessoas responsável pelo mapeamento das informações definidas pelos usuários de aplicação, para as primitivas do modelo de dados suportado pelo SGBD em questão. Geralmente, o ABD é considerado parte integrante do SBD, pelo fato do desempenho deste depender das decisões sobre as categorias de acesso e estruturação física/lógica tomadas pelo ABD [DATE/86].

O SGBD é um conjunto de procedimentos que isola os programas de aplicação dos detalhes internos referentes à criação, armazenamento, busca, atualização, segurança e estruturação física da Base de Dados [TSIC/77]. Sendo assim, o usuário terá acesso aos dados que lhes são de direito, sem estar, contudo, preocupado com aspectos internos dos mesmos, ou seja, sua organização e estruturação interna. Desta forma, o SGBD deve incluir funções que facilitem não somente na definição das estratégias dos dados, mas também nos acessos que serão feitos pelos usuários, conforme mostra a figura 2.1.

2.3- Características de um SBD

Pode-se considerar que os principais objetivos a serem atendidos por um SBD são resumidos basicamente em proporcionar Independência e Compartilhamento de Dados, objetivando uma melhor utilização destes e integrando-os como um conjunto global. A seguir serão definidas as principais características que devem estar presentes em um SBD.

2.3.1- Independência dos Dados

Entende-se por Independência dos Dados, o isolamento existente entre a forma de utilização dos dados pelo usuário, a estrutura de armazenamento e também a estratégia de acesso dos mesmos. Com isso, objetiva-se a diminuição da interdependência entre os programas de aplicação e a forma de tratamento dos dados requisitados pelos mesmos programas. Este isolamento é definido por uma interface

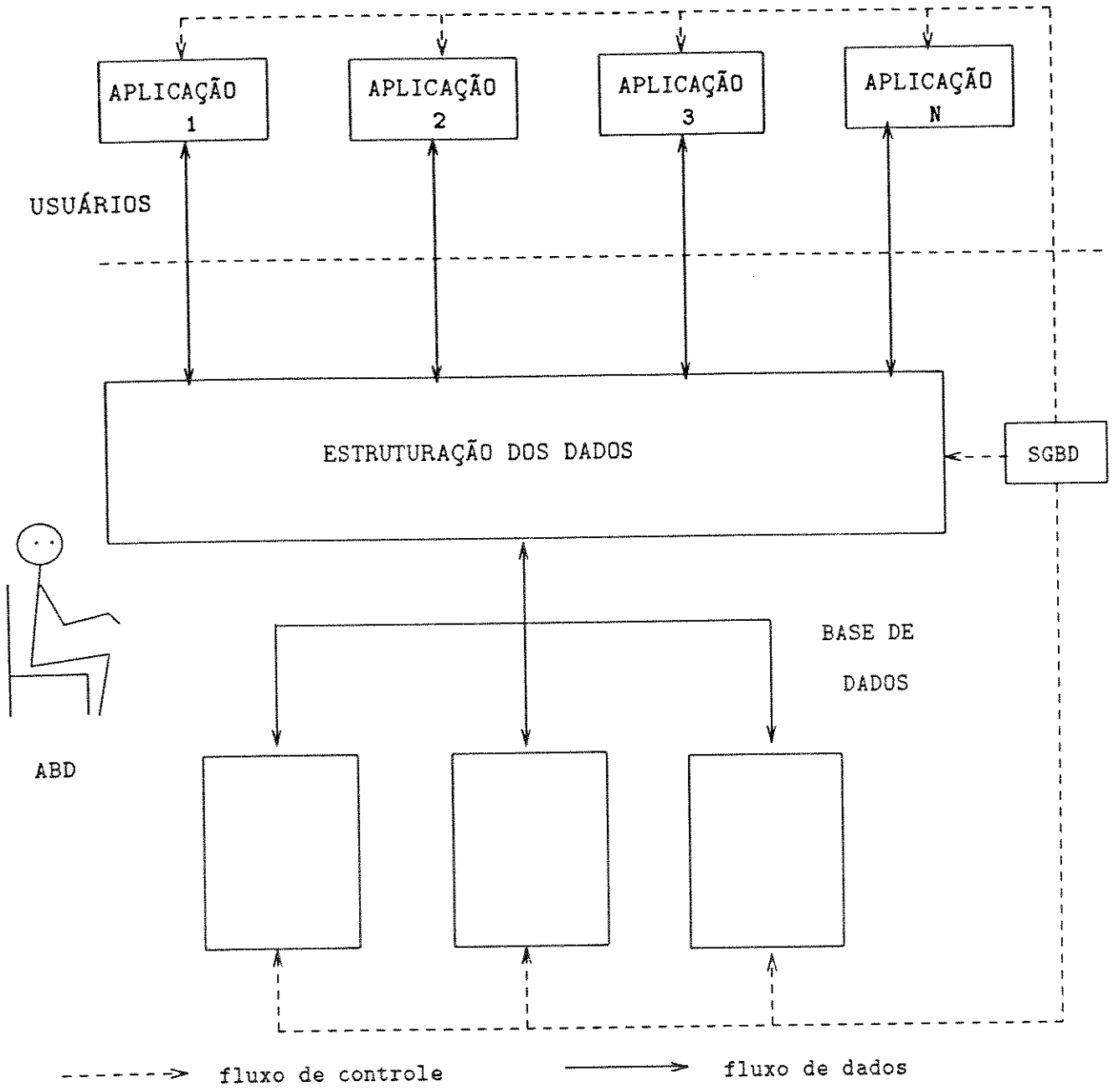


FIGURA 2.1 SISTEMAS DE BANCOS DE DADOS

que mapeia a representação lógica dos dados na sua representação física e vice-versa.

De acordo com a complexidade e a potencialidade das funções embutidas no SGBD, são distinguidos diferentes graus de independência de dados que podem ter a seguinte classificação: INDEPENDÊNCIA SIMPLESMENTE FÍSICA, INDEPENDÊNCIA FÍSICA E PRINCIPALMENTE LÓGICA e INDEPENDÊNCIA FÍSICA E LÓGICA. Pode-se concluir que, de uma forma geral, quanto maior for o grau de independência dos dados, menor será o custo de adaptação dos programas já existentes, causado por possíveis modificações de representação internas dos dados no computador.

2.3.2.- Compartilhamento de Dados

Um SBD deve permitir que dados sejam compartilhados. Num nível mais simples, ele deve permitir que dados criados por um usuário possam ser utilizados por outro usuário. O compartilhamento de dados tem como função permitir o uso compartilhado de Bases de Dados por vários usuários, embora não necessariamente no mesmo instante. O compartilhamento é considerado como um fator muito importante nas aplicações, onde se têm um grande volume de dados e uma intensa comunicação entre os programas que representam as aplicações (figura 2.2). Como consequência deste compartilhamento, têm-se um decréscimo da necessidade de redundância física dos dados, facilitando assim o controle de consistência. Contudo, geram-se alguns problemas, tais como: a quebra de privacidade dos dados particulares dos usuários; possibilidade de ocorrência de duas ou mais operações simultâneas e incompatíveis, causando um intertravamento (*deadlock*) caso ocorra falha na coordenação do Sistema; etc.

Vale acrescentar ainda que, pelo fato de programas de aplicação compartilharem os mesmos dados e não apresentarem finalidades necessariamente semelhantes, o SGBD deve também prover recursos que abstraiam diferentes visões do mundo real, a partir da mesma Base de Dados. É como se os dados fossem estruturados para o uso

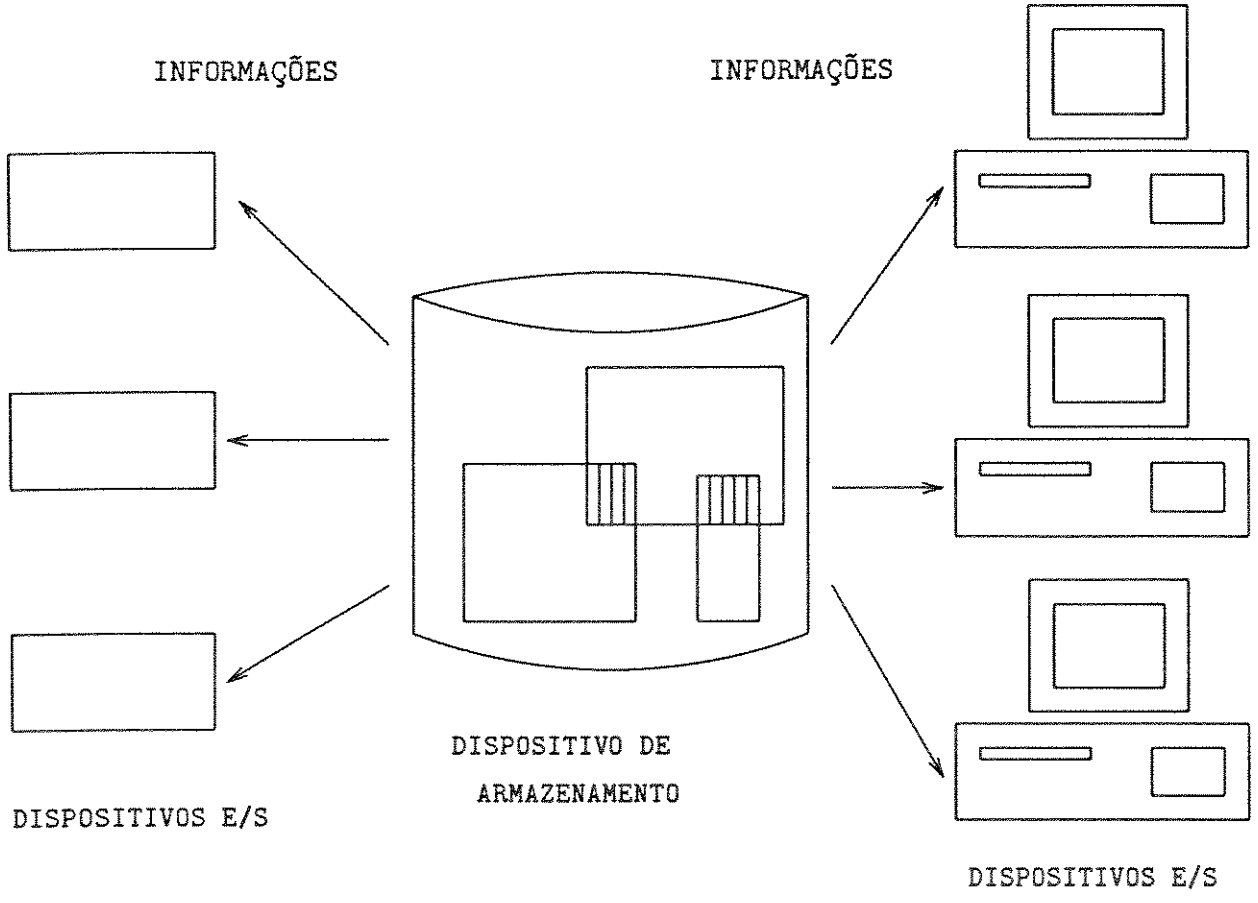


FIGURA 2.2 COMPARTILHAMENTO DE DADOS

exclusivo de cada usuário. Sendo assim, alguns aspectos relacionados com compartilhamento de dados devem ser ressaltados. A seguir serão detalhados esses aspectos.

2.3.2.1- Controle de Concorrência e de Consistência

SBD's têm um mecanismo de controle de concorrência que evita usuários executarem ações inconsistentes na Base de Dados. Isso é feito bloqueando-se o acesso aos dados que estão sendo escritos ou lidos por um determinado usuário. O controle de concorrência é geralmente acoplado com o mecanismo de transação que provê atomicidade: um grupo de operações solicitados por um usuário é executada totalmente ou não é executado. Assim, a Base de Dados nunca é deixada parcialmete atualizada, e mudanças parciais não são visíveis a outros usuários [ZDON/90].

Diz-se que uma Base de Dados é consistente num determinado instante, se quaisquer grupos de dados idênticos geram resultados de valores iguais para inferências idênticas, satisfazendo ao mesmo tempo as restrições impostas. Vale observar, que nos casos onde são permitidas redundâncias, o SBD deve garantir a consistência nas redundâncias . A sequência de operações que leva a Base de Dados de um estado consistente para um novo estado também consistente é chamada de **transação.** No período de transformação da Base de Dados de um estado para outro, a mesma fica temporariamente inconsistente [NEUM/82]. O SBD deve ter controles que distingam as inconsistências temporárias das permanentes, para neste ultimo caso tomar as devidas providências corretivas.

Para um SBD assegurar a correção e consistência na Base de Dados, tem-se que observar as restrições de integridade. Restrições de integridade são declarações que devem ser sempre verdadeiras para os dados da Base de Dados. Existem vários tipos de restrições que são:

- Restrição de domínio para campos de registro, que especifica o intervalo legal dos valores que um campo de um determinado registro

pode assumir;

- Restrição de chave, que diz quais valores de dados servem para identificar univocamente ítens numa coleção (por exemplo: que nº da identidade duma pessoa, identifica uma única pessoa numa cidade);

- Restrição referencial, que declara que uma referência num dado conduz a um outro dado.

Alguns SBD's provêm mecanismos de disparo (*triggers*) para auxiliar na imposição de restrições: ações podem ser iniciadas para acessar a um determinado dado, para verificar se as restrições são mantidas; ou executar atualizações adicionais para conservar a Base de Dados em um estado consistente.

2.3.2.2- Redundância

Dados redundantes podem causar disperdício de espaços físicos, dificultando as operações de manipulação dos dados, isto significa que a atualização de um dado redundante requer a mesma atualização nos outros dados redundantes para evitar inconsistências. Contudo, nem sempre a redundância é indesejável. Em alguns casos, ela é recomendada para melhorar o tempo de acesso e simplificar os métodos de endereçamento; ou ainda para assegurar a recuperação de dados no caso de falhas acidentais. Sendo assim, um SBD tem que eliminar os dados redundantes desnecessários, para fazer a otimização entre o desempenho do Sistema e a quantidade de dados redundantes; e também garantir a consistência da redundância.

2.3.3- Linguagem

Um SBD oferece uma linguagem de definição e manipulação de dados. A Linguagem de Definição de Dados (LDD) é uma linguagem para definição da estrutura do SBD. A estrutura de armazenamento e os métodos de acesso usados pelo SBD são especificados por um conjunto de definições em um tipo especial de LDD denominado Linguagem de Definição de Dados e Memória. Já Linguagem de Manipulação de Dados

(LMD) é uma linguagem que permite aos usuários acessar ou manipular dados organizados por um modelo de dados apropriado. A parte de uma LMD que envolve consulta a informação é denominada Linguagem de Consulta [KORT/89]. É interessante que se tenha uma Linguagem de Consulta declarativa e um suporte de acesso associativo. Principalmente nos SBD relacionais, existe a facilidade de expressar consultas ou atualizações desejadas na Base de Dados. Esta facilidade não exige que o usário dê os detalhes de como a resposta será obtida. Ele fornece "o que", e o processador de consultas determina "como". Várias consultas expressas nesta linguagem são associativas; ou seja, elas requisitam recuperação de dados baseada nas propriedades dos mesmos. Esta recuperação também pode ser baseada nas propriedades de outro dados que estão relacionados. Assim vários SBD's suportam criação e manutenção de estruturas auxiliares para acesso de dados (índices, arquivos invertidos), que dão caminhos alternativos para localizar dados de maneira associativa. Assim, o processador de consultas escolhe que estrutura ele deve utilizar para responder a uma consulta.

2.3.4.- Persistência

Um SBD provê um armazenamento persistente e estável. Entende-se por persistência o fato dos dados serem acessíveis após todo o processo de criação. Armazenamento estável significa que os dados têm alguma proteção em caso de alguma falha no processo, no Sistema ou na comunicação dos dados. No caso de falha de processo, um programa usando o SBD termina anormalmente. Na falha de Sistema, o computador onde o SGBD reside sai de operação devido a um erro de software do Sistema Operacional ou do SGBD; ou ainda por um erro de hardware. Na falha do meio de comunicação, o meio de armazenamento (usualmente um disco rígido) é danificado. Muitos SGBD's solucionam falhas de processo e de Sistema com um mecanismo de recuperação, que escreve informações sobre as mudanças da Base de Dados em memória secundária; e utiliza essas informações para fazer correções dos dados após uma falha. Para solucionar as falhas de comunicação, o Sistema provê: a duplicação de dispositivos de memória; escreve informações de

estado na Base de Dados; ou atualiza a fita magnética.

2.3.5- Desempenho

O desempenho de um SBD está intimamente relacionado com a estruturação física e lógica dos dados, e considera como fatores de avaliação: o volume do espaço físico ocupado; o acesso aos dados; e os tipos de interação usuário/Sistema que estão disponíveis. Por exemplo, para os usuários que estão *on-line*, o tempo de diálogo homem-máquina (intervalo de tempo entre a entrada do comando e o retorno da mensagem do mesmo) é um fator essencial na avaliação do desempenho do SBD.

Com o intuíto de aumentar o desempenho do Sistema,recursos tais como Migração de Dados e Sintonização podem ser incorporados as facilidades oferecidas pelo SBD.

2.3.5.1- Migração de Dados

Dependendo do Sistema, é interessante fazer agrupamento de dados conforme sua frequência de utilização. Pode-se tomar como estratégia, por exemplo, agrupar os dados menos utilizados em memórias lentas (fitas) e os que são frequentemente utilizados armazenados em memórias rápidas (disco rígido). Periodicamente deve-se fazer uma reavaliação de todos os dados da Base de Dados e fazer, se for o caso, uma reorganização de suas localidades. Este mecanismo é denominado de Migração de Dados. Vale salientar, que o decréscimo do tempo de acesso aos dados não é obtido somente com Migração de Dados, mas também através da definição ou predefinição da estratégia de acesso a ser utilizada (tabelas hash, B-trees, etc).

2.3.5.2- Sintonização

Dependendo das alterações que os dados sofram, suas estruturas podem ter também modificações (por exemplo, caminhos de acesso, quantidade de registros, índices, apontadores, etc). É sabido que o desempenho dos programas de aplicação depende da estrutura de

dados - sempre existe uma estrutura que é mais adequada para uma determinada aplicação. Assim, deseja-se que o SBD tenha condições de organizar os dados dos programas de aplicação, em uma de suas estruturas disponíveis que mais se adequada ao programa, assegurando desta forma sempre um bom desempenho.

2.3.6- Autorização

Muitos SBD suportam posse de dados a um usuário particular e dão ao proprietário um mecanismo para permitir acesso seletivo aos seus dados por outros usuários.

2.4- Abstração de Dados

Um conceito que surgiu paralelamente ao conceito de independência dos dados foi o de **Abstração de Dados**, que considera não somente a questão da representação dos dados, mas também a sua semântica [HAMM/76] . Uma coleção de dados numa Base de Dados pode ser representada em níveis de abstração semanticamente distintos e que apresentam diferentes graus de independência lógica e/ou física. Considera-se que quanto maior o nível de abstração dos dados, maior será a aproximação semântica destes dados com o mundo real visto pelo usuário.

A fim de que todas as infomações relevantes de um programa de aplicação sejam armazenadas num SBD, é necessário que estas informações sejam identificadas e modeladas por meio de primitivas aceitáveis pelo SGBD, e a partir daí convertidas em dados internos compatíveis com a estrutura física de armazenamento.

O processo de identificação e modelagem das informações é consequência das diferentes percepções das mesmas. Estas percepções podem ser classificadas em diferentes Níveis de Abstração de Dados. Apesar de não existir nenhum padrão para classificá-los, neste trabalho serão considerados quatro níveis, conforme os que são

especificados em [CHEN/76], [SETZ/86] e [WGIO/77]. Eles são:

- Nível Conceitual;
- Nível Descritivo;
- Nível Organizacional;
- Nível Físico.

Para melhor compreensão da hierarquia dos níveis, ver figura 2.3. Conforme mostra a figura, acima de todos os níveis está o conjunto de informações que de forma abstrata, descreve o ambiente onde uma determinada aplicação está inserida. Estas informações encontram-se a níveis de idéias, não existindo nenhum tratamento formal. Elas constituem o que se chama de Mundo Real. Neste Mundo Real encontram-se os agentes que atuam na aplicação (os seres, os fatos, as coisas e organismos sociais). Cabe ao usuário, ou ao projetista da Base de Dados, definir que parte do Mundo Real é interessante à aplicação para fins de tratamento de informações [SETZ/86]. Quando se está fazendo esta definição, automaticamente se está definindo o que chamamos de Domínio de Aplicação.

Os Níveis Conceitual e Descritivo referem-se a uma percepção mais próxima do Mundo Real, enquanto que os níveis Organizacional e Físico aproximam-se mais da concepção do Mundo Computacional. O processo de mapeamento das informações, vai desde a modelagem pelo usuário de aplicação (Nível Conceitual) até a estrutura física dos dados (nível Físico), onde realmente se dá o armazenamento dos dados na memória física do computador.

2.4.1- Nível Conceitual

No Nível Conceitual o usuário trata das informações do Mundo Real relativas a um determinado problema. Melhor dizendo, somente a parte do Mundo Real de interesse do usuário é modelada. Utiliza-se para isso uma forma de comunicação que seja mais próxima e familiar ao usuário; e de preferência sem, ou com um mínimo possível, uso de formalismos matemáticos.

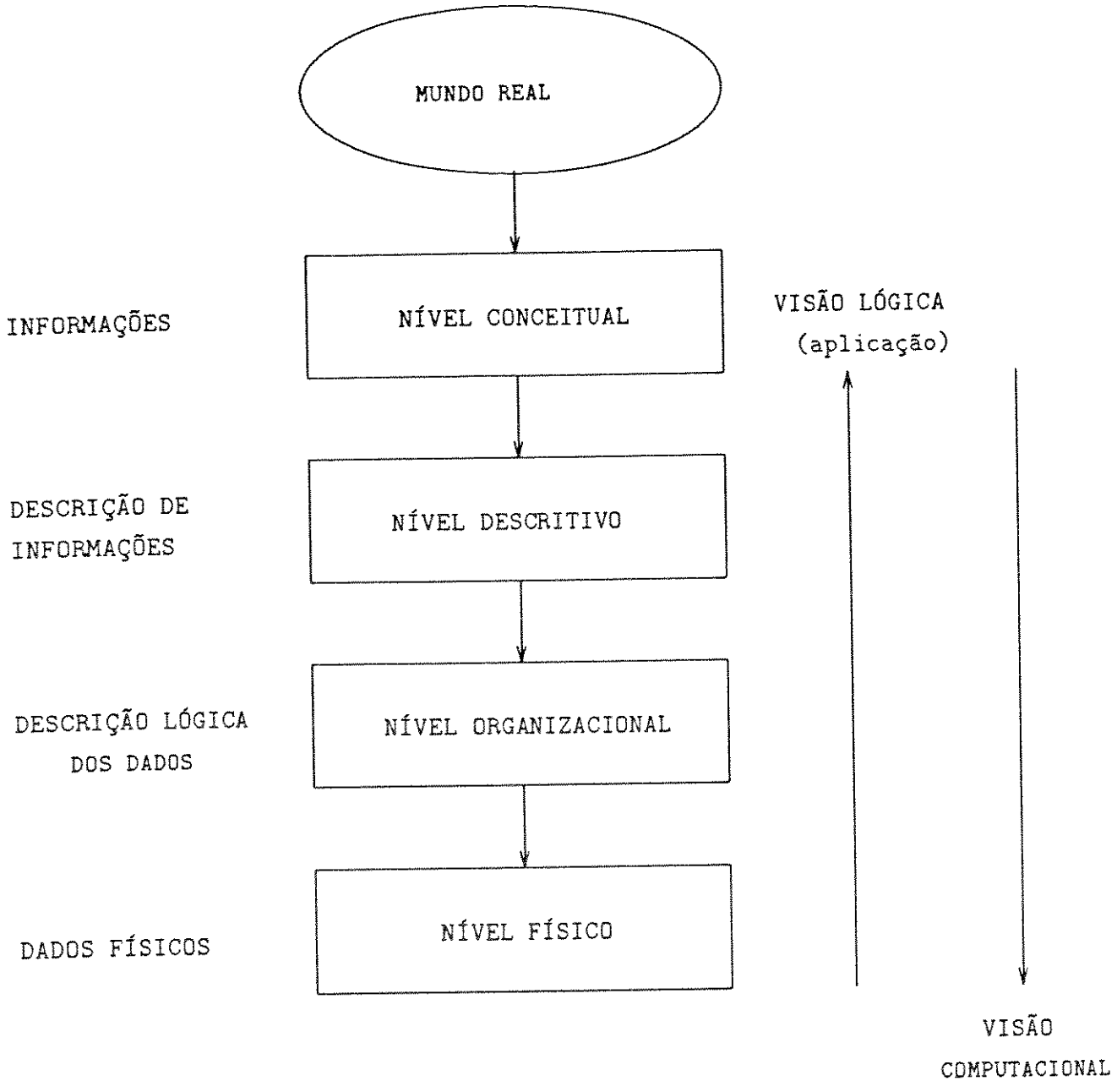


FIGURA 2.3 - NÍVEIS DE ABSTRAÇÃO DE DADOS

Desta maneira, tem-se no Nível Conceitual a descrição de um universo (ou partes dele), que deve ser de forma clara e inteligível para as pessoas que interagem com ele. Para isso não são exigidos maiores conhecimentos adicionais, além daqueles normalmente empregados nessa interação. Esta descrição deve ser bem organizada e representar o máximo possível um modelo da realidade, que será chamado de **Modelo Conceitual**.

Há várias formas de comunicação que representam o modelo conceitual. Uma delas é através de sentenças obedecendo as regras gramaticais e semânticas de uma determinada linguagem. Para efeito de exemplo, apresenta-se a seguir uma descrição textual simplificada da organização de uma indústria que projeta e fabrica roupas:

1) A **indústria** possui vários **empregados**;
2) Cada **empregado** está lotado numa **divisão** da **indústria**;
3) Cada **divisão** é responsável por um **projeto**;
4) Cada **projeto** utiliza (na fabricação do produto projetado) vários **tecidos** que são oferecidos por diversos fornecedores e são utilizados na confecção de **roupas**;
5) Os **empregados** são caracterizados na **indústria** pelo **nome**, **número da matrícula do INAMPS** e **salário**.
6) Os **projetos** são caracterizados dentro da **indústria** pelo **nome**, **número do projeto**, **data de início** e **tempo previsto de execução**.

Com o crescimento do uso do computador como ferramenta de apoio às atividades humanas, surgiu a necessidade de meios mais adequados para comunicação entre os seres humanos e os computadores. Preferivelmente, esta comunicação deveria se processar em linguagem natural dos usuários (por exemplo português, francês, etc). No entanto, no estágio atual do desenvolvimento de Software, os computadores não absorveram ainda a capacidade de lidar com vocabulários volumosos e estruturas gramaticais sofisticadas, havendo assim a necessidade de limitar a semântica de sua linguagem e estabelecer regras gramaticais mais simplificadas.

2.4.2- Nível Descritivo

Desenvolve-se no nível Descritivo um modelo da informação denominado Modelo Descritivo. A linguagem utilizada durante o estabelecimento do modelo descritivo deve ser estritamente formal, pois o objetivo final é atingir, através de níveis de abstração posteriores, um modelo computacional que possa ser fornecido e processado pelo computador.

Um modelo descritivo inicial, derivado do modelo conceitual, pode ser formalizado, por exemplo, através das seguintes primitivas: Entidades (substantivos), Relacionamentos (verbos ou ações) e Atributo (adjetivos ou advérbios).

Define-se como Entidades os objetos e/ou eventos do modelo conceitual. É muito comum que as entidades semelhantes sejam agrupadas em conjuntos ou classes de entidades, chamadas Tipos de Entidades. Cada um destes é definido por um conjunto de Atributos, que corresponde ao conjunto das características e propriedades dos objetos.

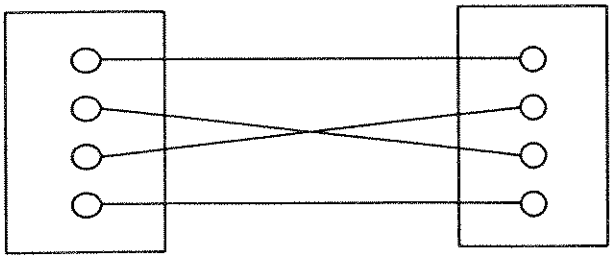
Na realidade existem vários tipos de entidades de relevância para o usuário, que associados podem compor outras informações. A associação estabelecida entre entidades chama-se Relacionamento. Basicamente existem três tipos de relacionamentos - 1:1, 1:N e M:N . Admitindo que a associação entre duas entidades A e B tal como correspondências fa: A -> B e fb: B -> a, pode-se definir formalmente três tipos de relacionamento, conforme a figura 2.4 [TSIC/77] :

1:1 - Quando ambas as correspondências fa e fb são funcionais (total ou parcialmente);

1:N - Quando somente uma das correspondências fa ou fb é funcional;

M:N - Quando nenhuma das duas correspondências é funcional, isto é, cada A pode ter muitos B associados e vice-versa.

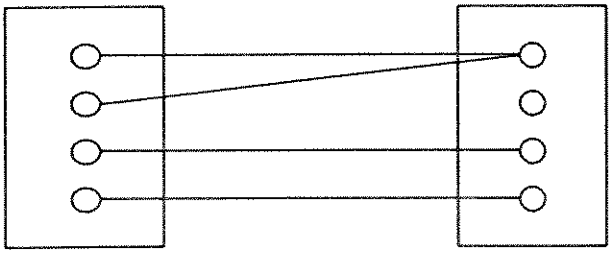
Assim o conceito de funcionalidade das associações explicita



ASSOCIAÇÃO 1 : 1

fx é totalmente funcional

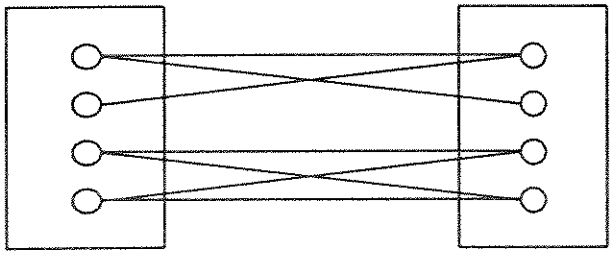
fy é parcialmente funcional



ASSOCIAÇÃO 1 : N

fx é totalmente funcional

fy não é funcional



ASSOCIAÇÃO M : N

fx não é funcional

fy não é funcional

fx

fy

FIGURA 2.4 TIPOS DE RELACIONAMENTO

o tipo de relacionamento existente entre os componentes identificados num contexto de aplicações.

Considerando como passo inicial a formalização acima obtida, o usuário já está habilitado a representar suas informações de uma maneira formal e precisa. Com isto, estas informações já podem ser emitidas ao responsável pelo SGBD, (por exemplo, o ABD) para que este,a partir da representção formal do usuário, projete a estrutura de dados da aplicação em questão. Para tal, utilizar-se-á dos recursos oferecidos pelo SGBD disponível.

Para projetar a estrutura de dados, o ABD precisa conhecer os tipos e a finalidade das informações que serão manipulados pelo SGBD. Como estes tipos são específicos e vinculados a aplicação, eles são usualmente definidos por usuários especializados na área de aplicação. Tal fato pode provocar inconsistência na comunicação entre o ABD e as informações representadas pelo usuário (modelo conceitual ou modelo descritivo inicial), comprometendo a precisão das informações. Consequentemente, o ADB pode interpretar as informações sob ângulos diferentes e optar, em níveis subsequentes de abstração, por uma organização ineficiente às necessidades do usuário.

Para resolver o possível problema da má interpretação das informações do Mundo Real definidas no Nível Conceitual, são utilizados Modelos de Dados com os quais se especifica formalmente os requisitos de uma aplicação específica e se define detalhadamente as primitivas identificadas no modelo Decritivo. Resumindo, os Modelos de Dados conduzem o usuário a especificar suas necessidades de forma precisa e concisa. Desta forma, os ABD's terão uma noção geral e correta sob a natureza das informações que devem constar na Base de Dados, a fim de suportar os possíveis programas que as utilizarão, e terão também condições de definir uma estrutura propícia aos recursos oferecidos pelo SGBD escolhido.

Com a utilização de Modelo de Dados tem-se, por um lado, o usuário utilizando-se de um modelo que é adequado para definição do

Modelo Descritivo de sua aplicação. Por outro, o ABD também utiliza um modelo que seja suportado pelo SGBD, e que sirva para passar ao computador o modelo das informações em questão. Com relação ao mapeamento do modelo descritivo do usuário para o modelo suportado pelo SGBD, ter-se-ía dois casos a serem analisados.

O primeiro caso, seria em que os Modelos de Dados utilizados pelos usuários (seus Modelos Descritivos), fossem diretamente mapeados com os Modelos de Dados utilizados pelo ABD (suportados pelo SGBD). Neste caso, bastaria somente que o ABD definisse a organização da estrutura dos dados na Base de Dados diretamente dos esquemas dos Modelos de Dados dos usuários. Estes esquemas seriam compilados e utilizados pelo SGBD, tanto para codificar e decodificar corretamente os dados, quanto para fazer controle de integridade e consistência dos mesmos.

O segundo caso, seria em que os modelos de dados utilizados pelos usuários e ABD não fossem mapeados diretamente. Neste caso, as informações deveriam ser transformadas e estruturadas de uma forma que fossem suportadas pelos recursos oferecidos pelo SGBD. A documentação dos esquemas, constituiria uma importatnte fonte de referência para o usuário ter idéia de como os dados armazenados deveriam ser manipulados (figura 2.5). Estes esquemas corresponderiam ao Nível Organizacional.

2.4.3- Nível Organizacional

O Nível Organizacional reflete a capacidade que o SGBD dispõe para gerar, automaticamente, um mapeamento correto e preciso entre o nível Físico e as representações lógicas dos dados. O ABD utiliza estas representações lógicas para organizar uma estrutura de dados que seja compatível com a estrutura oferecida pelo SGBD. Esta organização é feita a partir das informações especificadas pelos usuários, informações estas que são correspondentes ao Nível Descritivo.

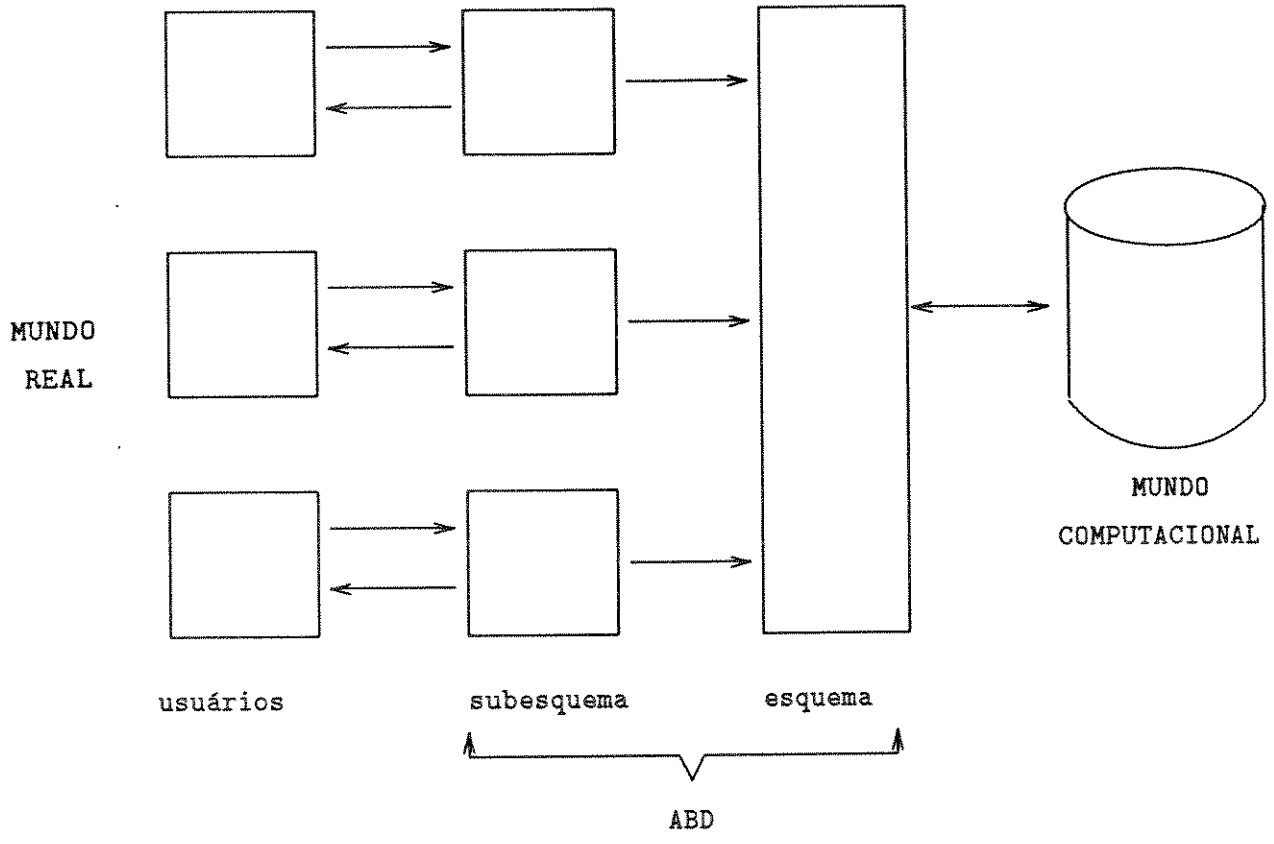


FIGURA 2.5 INTERFACE USUÁRIOS (Mundo Real) /
ABD (Mundo Computacional)

Quanto ao grau de abstração, o Nível Organizacional pode ser classificado em duas categorias:

- Dependente dos caminhos lógicos de acesso,
- Independente dos caminhos lógicos de acesso.

Modelos independentes dos caminhos lógicos de acesso não oferecem recursos explícitos para que os usários possam interferir na organização física do dados. Geralmente, um SGBD que suporte estes modelos tem uma estrutura mais complexa, devendo conter algoritmos para mapear as descrições formais em estruturas físicas mais adequadas aos programas de aplicação.

Independente do grau de abstração, uma Base de Dados pode ser acessada por vários usuários que têm diferentes visões da informação armazenada. Assim, para facilitar a interação usuário/Sistema e também proteger os dados de acessos inadequados, é desejável que as informações de interesse de um determinado usuário permaneçam à parte das informações consideradas globais. Desta forma, considerando este enfoque, são destacados ainda dois subníveis de abstração, como mostra a figura 2.6:

1- SUBESQUEMAS: estão relacionados com as descrições formais dos dados utilizados pelos programas de aplicação. Trata-se da visão que cada usuário tem sobre a Base de Dados;

2- ESQUEMAS: estão relacionadas com as descrições formais do contexto global da Base de Dados. Trata-se da visão que um ABD deve ter sobre a Base de Dados, para que o mesmo possa analisar e supervisionar o desempenho do Sistema como um todo.

Os esquemas e os subesquemas formam um conjunto de **Máscaras**, que são utilizadas pelo SGBD para fazer o mapeamento dos dados físicos para as abstrações associadas aos diferentes níveis/subníveis e vice-versa, como mostra a figura 2.6.

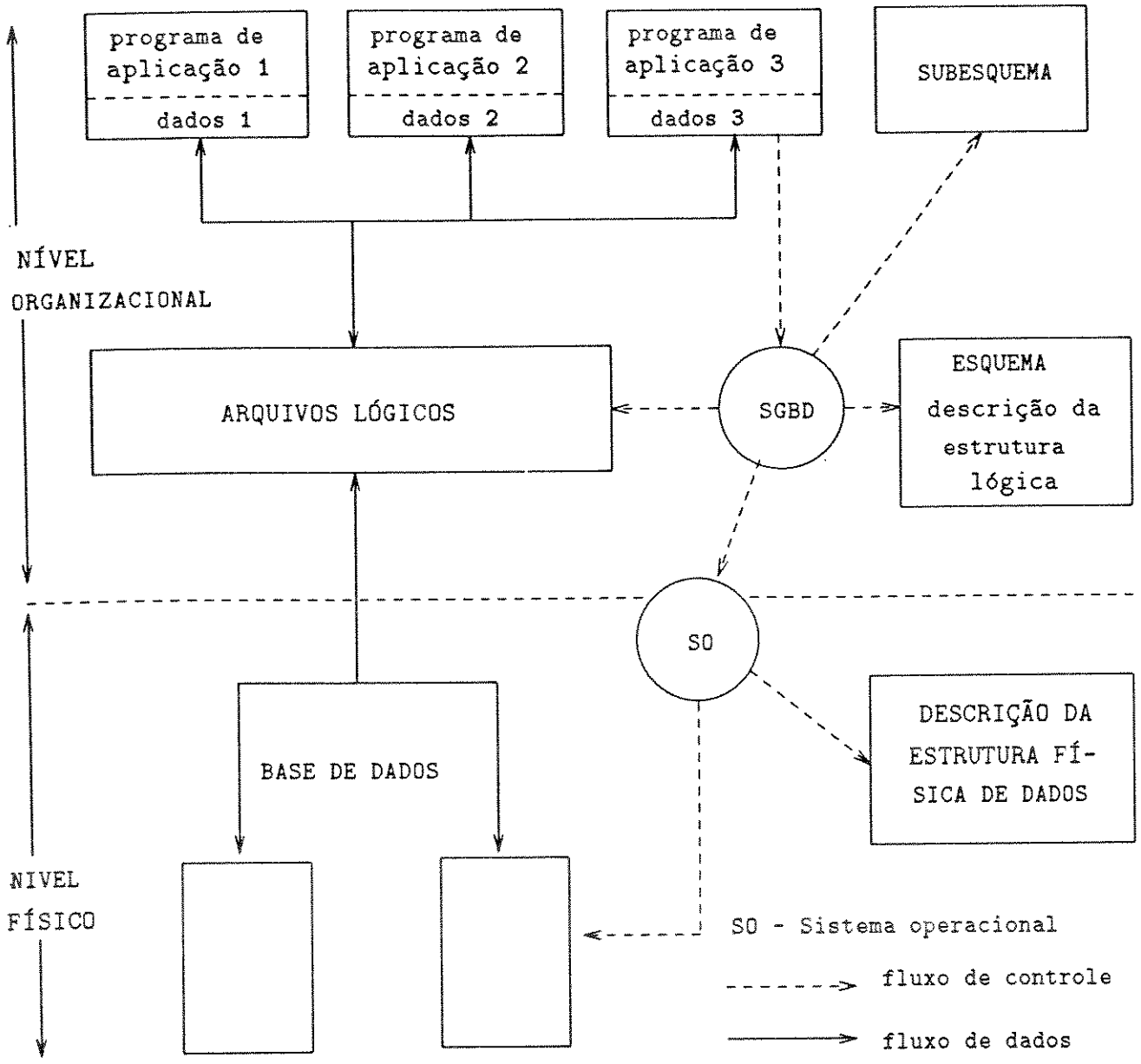


FIGURA 2.6 DISTINÇÃO NOS DOIS SUBNÍVEIS DE ABSTRAÇÕES NO NÍVEL ORGANIZACIONAL (ESQUEMAS E SUBESQUEMAS)

2.4.4- Nível Físico

Para definir a organização física dos dados, os critérios usados são diferentes daqueles utilizados para a organização lógica dos dados. Geralmente, na maior parte dos Sistemas, a implementação da estrutura física dos dados fica sob a responsabilidade do Sistema Operacional no qual o SGBD está apoiado. Desta forma, citam-se somente as características físicas da Base de Dados que devem ser analisadas pelo ABD a fim de averiguar a compatibilidade entre as exigências das aplicações e o SGBD. Existem estudos para propiciar este suporte tal como o trabalho de [CARV/82].

A organização física dos dados está intimamente ligada à eficiência operacional: tempo de operação (caminhos de acesso, localidade física dos dados, etc) e custo (volume de espaço físico ocupado). A seguir são relacionadas algumas características que devem ser observadas quando da escolha de uma estrutura física dos dados [MART/77]:

1- Espaço de memória;
2- Redundância física dos dados;
3- Tempos de operação desejáveis (inserção, remoção, etc);
4- Tempos de resposta desejáveis;
5- Volatilidade dos dados armazenados: tempo de permanência dos dados no Sistema;
6- Tipos de dados armazenados: tamanho dos registros, bloco, etc;
7- Estrutura dos dados lógicos que ela suporta;
8- Grau de confiabilidade e eficiência necessário;
9- Custo.

Observe-se que várias dessas características são incompatíveis entre si. Por exemplo, Sistemas com alta densidade de armazenamento (ocupa pouco espaço físico) requerem maiores tempos de busca (tempo de operação maior) conforme mostra figura 2.7.

FIGURA 2.7 CARACTERÍSTICAS ANTAGÔNICAS DE UMA ESTRUTURA FÍSICA

2.5- Resumo

Um SBD é definido como uma coleção de dados interrelacionados, que é armazenada e controlada por um conjunto de programas do Sistema. Estes programas têm como objetivo tornar a utilização da Base de Dados uniforme à aplicação. Esta coleção de dados é referenciada como Base de Dados e funciona como uma espécie de fonte ou depósito de informações. Os programas que fazem o controle de trânsito de informações na Base de Dados são agrupados como o SGBD.

Existem algumas características fundamentais que devem estar presentes num SBD, que são:

1- Independência de Dados;
2- Compartilhamento de Dados;
3- Abstração de Dados.

Independência de Dados é o isolamento que o SBD oferece entre a forma de utilização dos dados pelo usuário, a estrutura de armazenamento e também a estratégia de acesso dos mesmos. Conforme a complexidade do SGBD, existem diferentes graus de independência que podem ter a seguinte classificação: Simplesmente Física; Física e particularmente Lógica; e Física e Lógica.

Compartilhamento de Dados é uma propriedade que um SBD deve oferecer para permitir que os dados de uma Base de Dados possam ter o uso simultâneo por vários usuários. É um fator muito importante em aplicações com grande volume de dados e uma intensa comunicação entre os programas que representam as aplicações.

Abstração de Dados é um conceito que considera não somente a questão da representação dos dados, mas também a sua semântica [HAMM/76]. Uma coleção de dados num Banco de Dados pode ser representada em níveis de abstração semanticamente distintos e que apresentam diferentes graus de independência lógica e/ou física. Existem diferentes níveis de abstração de dados que podem ser

classificados segundo critérios de [CHEN/76], [SETZ/86] e [WGIO/77], a saber:

* Nível Conceitual;
* Nível Descritivo;
* Nível Organizacional;
* Nível Físico.

Nível Conceitual e Descritivo referem-se a uma percepção mais próxima do Mundo Real, enquanto que os Níveis Organizacional e Físico aproximam-se mais da concepção do Mundo Computacional. O processo de mapeamento das informações, vai desde a modelagem do usuário (Nível Conceitual) até a estrutura física dos dados (Nível Físico), onde se dá o armazenamento das informações na memória física do computador.

No próximo capítulo serão discutidos alguns aspectos de Sistemas Orientados por Objetos, fazendo uma análise da necessidade destes Sistemas absorverem algumas facilidades oferecidas pelos SBD's, para que os Sistemas Orientados por Objetos atendam aos requisitos de novas aplicações.

CAPÍTULO 3

Sistemas de Banco de Dados Orientados por Objetos (SBDOO)

3.1- Introdução

É preocupação das pessoas que pesquisam Sistemas de Banco de Dados, a gerência de grande quantidade de dados de forma persistente, confiável e compartilhada. Grande, significa muito espaço para que os dados permaneçam na memória principal; persistente significa que dados persistem de uma sessão para outra; confiável significa recuperável em caso de falha do *hardware* ou *software* e compartilhada significa que vários usuários podem acessar os dados de uma maneira ordenada. Estes quatro adjetivos, como já vistos no capítulo 2, caracterizam os principais problemas de SBD e definem a especificidade da área.

É possível, por exemplo, encontrar novas soluções fora do mundo dos sistemas relacionais, para o problema básico de acesso compartilhado de grande quantidade de dados persistentes e confiáveis.

No entanto, para pessoas fora da área de SBD, a comunidade desta área deveria estar preocupada com o projeto e implementação de sistemas relacionais voltados para aplicações comerciais (ditas convencionais). A ênfase nessa aplicação é obviamente histórica: a bordagem dos problemas em SBD foi primeiramente experimentada pelos implementadores das referidas aplicações. Daí a tendência específica aos requerimentos desta área. A ênfase de sistemas relacionais veio do fato que estes proveram boas respostas a essas aplicações [BANC/88].

Três novos fenômenos surgiram, no entanto:

1- Novas aplicações, fora das áreas convencionais, estão sentindo a necessidade de gerenciar grandes quantidades de dados de forma confiável, compartilhada e persistente. Por exemplo , aplicações em CAD (*Computer Aided Design*), CASE (*Computer Aided Software Engineering*), automação de escritórios, inteligência artificial, etc.

Estes novos usuários da tecnologia de SBD trouxeram novas exigências.

2- A taxa de custo, memória principal/memória em disco tem mudado, modificando as suposições em torno dos projetos atuais de SBD.

3- A dificuldade de compatibilização entre linguagens de programação e linguagens de consultas relacionais. Sistemas Relacionais são convenientes para meios de consulta, mas não para desenvolvimento de aplicações. Desenvolvimento de aplicações requer a comunicação entre uma linguagem de consulta relacional e uma linguagem de programação. Estes dois tipos de linguagens não são em geral compatíveis. Elas têm tipos diferentes, modelos computacionais diferentes, etc. Para resolver esta dificuldade, tem-se visto uma tendência em integrar tecnologia de SBD e de linguagem de programação.

3.2- Natureza da área

Três fatores principais contribuíram para o interesse em Sistemas de Banco de Dados Orientados por Objeto (SDBOO), que são:

1- A comunidade de SBD percebeu a dificuldade de compatibilidade, referida na seção 3.1. Com isto iniciou-se a busca de tecnologias para solucionar este problema. Orientação por objeto tornou-se promissor pelo fato de prover um suporte para representar e gerenciar programas e dados.

2- Pessoas que constroem Sistemas Orientados por Objeto necessitam da funcionalidade de SBD. Adicionar algumas destas funcionalidades a um Sistema Orientado por Objeto, o torna um SBDOO.

3- A tecnologia orientada por objeto provocou interesse na comunidade de SBD. Alguns aspectos desta tecnologia (essencialmente a abordagem do poder de modelar e o aspecto do modelo semântico de dados) foram responsáveis por tal interesse. Isto fez com que se incluísse alguns destes aspectos em novos protótipos de SBD's. O enfoque dado para resolver o problema do projeto de SBD, foi prover

ferramentas mais poderosas para modelar o mundo real.

Um dos objetivos da comunidade de SBD é integrar tecnologia de SBD e a abordagem orientada por objeto em um único Sistema.

3.3- Orientação por Objeto

Nesta seção descrevem-se as principais características de um **Sistema Orientado por Objeto** (SOO). Estas características deverão ser introduzidas em um SGBD, de forma que o torne um SBDOO. Talvez estas características representem as idéias mais importantes e aquelas que terão maior impacto na produtividade do programador de aplicação.

3.3.1- Encapsulação

Encapsulação é o princípio que permite modelar ao mesmo tempo, dados e operações. Um objeto tem uma parte de **interface** e uma parte de **implementação**. A parte de **interface** é a especificação do conjunto de operações que podem ser executadas no objeto. É a única parte visível do objeto. A parte de **implementação** tem uma parte de **dados** e uma parte de **operações**. A parte de **dados** é a memória do objeto e a parte de **operações** descreve, em alguma linguagem de programação, a implementação de cada operação.

A título de ilustração, considere o objeto **empregado**. Este objeto poderia ser representado por alguma relação em um Sistema Relacional. Este Sistema requereria o uso de uma linguagem de consulta relacional, com a qual se consultaria a relação, e posteriormente um programador de aplicação escreveria programas para atualizar os registros da referida relação. Ter-se-íam assim, programas para aumentar salário de **empregado** e para demitir **empregado**. Estes programas seriam escritos em alguma linguagem com declarações do tipo **SQL** (*Structed Query Linguage*) embutidas [KORT/89].

Os programas seriam armazenados em sistemas de arquivos tradicionais, separadamente da Base de Dados. Desta forma fica bem

claro que existiriam distinções entre programa e dados, e entre linguagem de consulta (para consultas) e linguagem de programação (para programas de aplicação).

Em SOO's, definir-se-ía empregado como um objeto que tem uma parte valor, que poderia ser similar ao registro definido no sistema relacional, e uma parte de operações que consistiria das operações **aumentar** e **demitir**, e talvez alguma operação extra para consultar os dados de **empregado**.

Assim, poder-se-ía ter um único modelo para representar dados e operações, e ter também encapsulação (ocultação) de informações. Uma vez definida a interface do objeto, nenhuma outra operação que não as que foram definidas nesta interface, poderia ser executada. Isto também é válido para atualização e consulta.

3.3.2- Identidade de Objeto

Identidade de objeto é a propriedade de um objeto que o distingue de outros objetos sem considerar seu conteúdo, locação ou endereço, permitindo o compartilhamento de objetos [KHOS/86]. A ídéia básica de identidade de objeto é a seguinte: num modelo com identidade de objeto, um objeto tem uma existência que independe do seu valor. Assim dois objetos podem ser idênticos (eles são o mesmo objeto), ou iguais (eles têm o mesmo valor). Consequentemente têm-se duas implicações: uma é o **compartilhamento de objeto** e outra é **atualização de objeto**.

Compartilhamento de objeto: Num modelo baseado em identidade de objeto, dois ou mais objetos podem compartilhar um objeto. Assim, a representação gráfica de um objeto complexo num Sistema com identidade de objeto pode ser um grafo acíclico direcionado, enquanto num Sistema sem identidade de objeto é representado por uma árvore. Considere o seguinte exemplo - uma pessoa tem um nome, uma idade e um conjunto de filhos. Assume-se que Pedro e Maria, têm um filho de 10 anos que se chama João. Na vida real podem acontecer duas situações: Pedro e Maria

têm ou não têm um filho. Num sistema sem identidade de objeto, Pedro é representado por (Pedro,35,{(João,10,{})}) e Maria por (Maria,37,{(João,10,{})}). Assim não existe nenhuma forma de se expressar que Maria e João têm um filho em comum. Num modelo baseado em identidade, as duas estruturas podem compartilhar ou não a parte em comum (João,10,{}), tendo assim as duas situações.

Atualização de Objeto: Considere agora que Pedro e Maria têm um filho João. Num modelo com identidade de objeto, quando se atualiza João, ou melhor, o objeto João (filho de Maria), atualiza-se automaticamente João filho de Pedro. Num sistema baseado em valor, ter-se-ía que se preocupar com atualizações de ambos sub-objetos [BANC/88].

Talvez seja possível simular identidade de objeto em sistemas baseados em valores, a medida que se introduz identificadores de objetos por toda parte. Mas isto traria uma carga muito grande para o usuário, e esta carga pode ser bastante penosa, por exemplo, para operação de coleta de lixo (*garbage collection*).

Note que modelos baseados em identidade sempre foram conhecidos em linguagens de programação imperativas; cada objeto manipulado no programa tem uma identidade e pode ser atualizado. Esta identidade vem do nome de uma variável, ou de uma alocação física de memória. Este conceito é bastante recente no mundo relacional, onde relações são baseadas em valores [BANC/88].

3.3.3- Tipos e Classes

Um **tipo**, em um SOO, descreve um conjunto de objetos com as mesmas características. Êle tem duas partes: a **interface** para o objeto e a **implementação** do objeto. A parte de **interface** é visível ao usuário do tipo e a parte de **implementação** do objeto é vista somente pelo projetista do tipo. A **interface** consiste de uma lista de operações com seus respectivos parâmetros.

A implementação do tipo consiste da parte de **dados** e da parte de **operações**. A estrutura interna dos dados do objeto é descrita na parte de dados. Na parte de operações é feito um programa para cada operação da parte de interface.

Tipos, em linguagens de programação, são utilizados como ferramentas para aumentar a produtividade do programador de aplicação, pois asseguram a correção do programa. Isto ocorre, pelo fato do usuário ser forçado a declarar a estrutura do objeto que ele manipula, e daí então o sistema verificará se ocorrem erros nas declarações ou manipulações dos objetos. Desta forma, tipos são usados no tempo de compilação para verificar a correção de um programa.

A noção de **classe** é diferente de tipo. Sua especificação é a mesma que a do tipo, mas seu uso vai além da questão de tempo de execução. Classe contém dois aspectos interessantes. O primeiro é referente a objeto **fabricante** (*factory*), que quer dizer que uma classe pode ser usada para instanciar novos objetos, executando a operação *new* na classe. O segundo aspecto se refere a objeto **recipiente** (*container*), que significa dizer que ligada à classe existem suas extensões, ou seja , o conjunto de objetos do sistema que pertencem a uma determinada classe naquele momento. O usuário pode manipular os objetos **recipientes** aplicando operações em todos elementos da classe. Classes não são usadas para verificar a correção de programas, mas sim para criar e manipular objetos.

3.3.4- Herança

Este é, provavelmente, o conceito mais poderoso em programação orientada por objeto. Este conceito permite objetos de estruturas diferentes compartilhar operações que lhes são comuns.

Considere o exemplo de **empregado** e **estudante**. Cada **empregado** tem um nome, uma idade e um salário. Êle ou ela pode morrer, casar-se e ser remunerado. Cada **estudante** tem um nome, uma idade e um conjunto de notas. Êle ou ela pode morrer, casar-se e ter média de notas (MEN)

calculada.

O projetista de uma Base de Dados, em um sistema relacional, definiria uma relação para **estudante** e uma para **empregado**; escreveria os códigos para as operações **morre**, **casa** e **remunera** para a relação **empregado**; e as operações **morre**, **casa** e **calcula MEN** para a relação **estudante**. Desta forma, o programa de aplicação teria 6 (seis) programas, ou seja três operações relacionadas com a relação **empregado** e três com a relação **estudante**.

Com relação a um sistema orientado por objeto, utilizando-se a propriedade de herança, o projetista assumiria que **empregado** e **estudante** seriam pessoas que teriam aspectos em comum e cada pessoa teria propriedades específicas. Introduziria assim, uma classe **pessoa** que teria como atributo **nome** e **idade**, e como operações **morre** e **casa**. Sendo assim, assumiria agora que **empregado** seria um tipo específico de **pessoa** e herdaria atributos e operações de **pessoa**, e teria características específicas que seriam: atributo **salário** e a operação **remunera**. Da mesma forma declararia **estudante** como um tipo específico de **pessoa**, com características específicas que seriam: um atributo **conjunto_notas** e uma operação **calcula MEN**. No caso de sistema orientado por objeto o programa de aplicação teria 4 (quatro) programas, ou seja, duas operações relacionadas com o objeto **pessoa**, uma operação com o objeto **estudante** e uma operação com o objeto **empregado**.

3.3.5- Redefinição (*Overriding*) e Ligação Retardada (*Late Binding*)

Há casos em que é desejável ter várias operações diferentes com o mesmo nome. Considere a operação **mostre**. Ela tem como entrada um objeto e mostra-o na tela. Dependendo do objeto deseja-se usar diferentes tipos de **mostre**, como segue:

1- Se o objeto for uma figura geométrica, deseja-se que ele apareça na tela.

2- Se o objeto for uma pessoa, deseja-se que algumas tuplas sejam imprimidas.

3- Se o objeto for um gráfico, deseja-se sua representação gráfica na tela.

Considere agora o problema de mostrar um conjunto de objetos, cujo tipo é desconhecido no tempo de compilação.

Num sistema convencional, têm-se três operações básicas: **mostra_pessoa**, **mostra_bitmap** e **mostra_gráfico**. O programador faz um teste para cada tipo de objeto e usa a operação **mostra** correspondente. Isto obriga o programador, quando da apresentação de um objeto, estar ciente do tipo de objeto e da operação **mostra** associada antes da compilação, e usá-lo coerentemente. Isto requer uma atenção muito grande do programador, pois qualquer esquecimento será fatal.

Num SOO, define-se a operação **mostra** num tipo de objeto de nível mais geral do Sistema. Assim, **mostra** tem um único nome e pode ser usado independentemente de ser um gráfico, uma pessoa ou uma figura. No entanto, redefine-se o corpo da operação **mostra** para cada tipo de objeto de acordo com a especificidade do mesmo. A este proceso dá-se o nome de **redefinição** (*overriding*). Assim, tem-se um único nome **mostra** denotando três programs diferentes. A este mecanismo dá-se o nome de **superposição** (*overloading*). Para mostrar o conjunto de elementos, simplesmente aplica-se a operação **mostra** para cada um deles. Neste caso, têm-se alguns ganhos adicionais, embora se escreva o mesmo número de programas: o programador não precisa se preocupar com os três programas diferentes; o código escrito é mais simples, pois não existe a necessidade de declaração nos tipos; e finalmente, o código é reutilizável, pois caso seja introduzido um novo tipo no sistema e no conjunto de objetos a ser mostrado, o mesmo programa **mostra** funciona, a não ser que se redefina a operação **mostra** para o novo tipo.

Para oferecer esta nova funcionalidade, o sistema não pode

ligar nomes de operações de programas no tempo de compilação. Portanto, nomes de operações são resolvidos no tempo de execução, este mecanismo é chamado de ligação retardada (*late binding*).

3.4- Sistemas Orientados por Objetos e Sistemas de Banco de Dados

É importante ter-se uma noção clara, de qual é a vantagem trazida pela introdução das funcionalidades de SBD em SOO. Em outras palavras, o que falta a um SOO, para que este se transforme em um SBDOO.

3.4.1- Aspectos de um SBD

As linguagens de programação tradicionais não têm, em geral, o conceito de conjunto. Elas usam outras estruturas para implementar conjuntos, tais como matrizes, listas ou arquivos. Isto se dá pelo fato delas não terem operações específicas para manipular conjuntos.

Em programação lógica, conjuntos existem somente em maior nível de abstração, a Base de Dados consiste de uma coleção de conjuntos nomeados (os predicados). Num nível de menor abstração têm-se listas ou termos e, se se deseja manipular conjuntos neste nível, tem-se que usar a ferramenta do sistema conjunto-de (*set-of*), que transforma um conjunto em uma lista de elementos.

Em SOO, construção de conjuntos e estruturação de classes são fatos distintos. Não existem construções de conjuntos e a única forma de manipular conjuntos é criar tipos e manipular suas extensões.

Uma das principais contribuições que a área de Sistemas de Banco de Dados tem trazido, é o fato de considerar conjunto como característica de grande importância. O modelo relacional introduziu uma álgebra baseada em conjuntos (álgebra relacional) com suas operações associadas. Existe uma limitação de conjuntos em sistemas relacionais que é o fato destes serem considerados somente como conjuntos de tuplas e tuplas serem de valores atômicos.

Relações aninhadas são tentativas de eliminar esta restrição, ou seja pode-se construir conjuntos de conjuntos de tuplas e assim por diante. Em muitos modelos relacionais aninhados existem algumas restrições (por exemplo, construtores de conjuntos e tuplas têm que se alternarem; objetos num conjunto têm que ser do mesmo tipo; etc.) e em todos eles, a Base de Dados é uma coleção de conjuntos nomeados (relações).

Acredita-se que o tratamento uniforme de conjunto é um dos maiores desafios do projetista de SOO.

A maioria dos SOO's não oferece persistência. Sendo assim, a única forma de obter dados de uma sessão para outra é usar um sistema de arquivo e salvar os dados necessários. Isto requer uma operação explícita de armazenamento e requer também a transformação do objeto, do formato da aplicação para o formato de arquivo. Tudo isto traz uma carga extra sobre o programador, torna o código complexo e degrada o desempenho.

Existem algumas características que SOO's não provêm, que são:

. proteção contra falhas no *hardware* ou *software*;

. mecanismos para recuperar ou assegurar atomicidade de transações.

SOO's são Sistemas mono-usuário e não fornecem qualquer controle sobre acesso concorrente de dados.

A maioria dos SOOs executam em memória principal ou virtual. Sendo assim os programas de aplicação estão limitados, em termo do tamanho dos dados que manipulam, ao espaço de endereço virtual. Uma vez que o disco é gerenciado somente através do mecanismo de memória virtual, o desempenho de manipulação de grande quantidade de dados, em

geral, não é boa.

Estes Sistemas não fazem uso de índices, nem de um *buffer* hábil de gerência de esquemas; não fazem agrupamento de objetos no disco; não têm estratégias hábeis para seleção e junção; ou otimização inteligente de consultas [BANC/88].

3.5- Análise de Sistemas de Banco de Dados Orientados por Objetos

Atualmente, SBD's convencionais, juntamente com suas ligações com linguagem de programação de propósito geral, podem oferecer várias facilidades, tais como: persistência, confiabilidade, compartilhamento de dados, podem modelar qualquer tipo de dado e podem executar qualquer computação possível. Mas, mesmo que qualquer aplicação possa ser escrita no topo de tais Sistemas, em muitos casos a aplicação seria extremamente difícil de ser definida e/ou seria bastante vagarosa. Desta forma, para comparar um sistema novo com um Sistema convencional, o critério de desempenho não é o poder computacional, mas sim a velocidade de computação ou a facilidade de programação [BANC/88].

3.5.1- Vantagens dos SBDOO's

Em vários aspectos os SBDOO's são potencialmente superiores aos SBD's convencionais, como segue:

1- Manipulação com Objetos Complexos - A habilidade de armazenar e manipular objetos complexos é uma característica de vários SBDOO's, enquanto SBD's convencionais são restritos a armazenar e manipular somente no plano de tuplas. Isto dá ao SBDOO um alto poder de modelagem, ou seja, estruturas complexas podem ser representadas diretamente, não precisando ser mapeadas para estruturas, por exemplo, relacionais de nível mais baixo.

2- Identidade de Objeto - A noção de identidade de objeto, conforme introduzida no ítem 3.3.2, fortalece ainda mais o poder de

modelagem dos SBDOO's. Identidade de objeto permite ao usuário modelar diretamente algumas situações, tais como compartilhamento de objeto e objetos cíclicos, sem adicionar uma camada extra no topo do Sistema. Ela provê operações, tais como igual e idêntico e provê uma semântica simples e natural para fazer atualizações.

3- Extensibilidade - Esta é uma vantagem importante da orientação por objeto. Pelo fato dos SBDOO's permitirem que novos tipos ou classes sejam adicionadas ao sistema, a capacidade deste sistema pode ser estendida. Isto é muito importante para adaptar o sistema a novos tipos de aplicações.

4- Armazenagem de Programas e Dados - Enquanto dados num Sitema Relacional são armazenados num sistema de banco de dados, programas de aplicação são armazenados em algum outro sistema. Este fato implica que nenhuma das características do SGBD pode ser usada para gerenciar os programas. Num SBDOO, programas e dados são armazenados sobre o mesmo formalismo dentro do sistema.

5- Tipagem e Herança, Redefinição e Ligação Retardada - Sistemas Relacionais não têm qualquer noção de tipagem. Cada relação tem seu próprio tipo e torna-se impossível declarar um determinado tipo e afirmar que várias relações são deste tipo. Herança, redefinição e ligação retardada são ferramentas que facilitam bastante as tarefas do programador.

3.5.2- Desvantagens dos SBDOO's

Enquanto, os SBDOO's são superiores aos SBD's convencionais em muitos aspectos, algumas das vantagens do primeiro são perdidas pelo fato de se considerar uma nova abordagem. Algumas vezes, se perde por não existir ainda uma tecnologia apropriada, outras vezes pelo fato do paradigma fazer coisas intrisicamente mais difíceis [BANC/88].

Algumas das desvantagens dos SBDOO's serão comentadas a seguir:

1- Simplicidade - Uma vantagem óbvia do modelo relacional é sua simplicidade. Neste modelo existem poucos conceitos básicos. Claramente esta simplicidade é perdida quando se move para o mundo de objetos. Aqui existem mais conceitos, alguns deles sobrepondo-se e a falta de um formalismo uniforme claro não auxilia na elaboração de uma estrutura formal sólida para o modelo de dados orientado por objeto.

2- Linguagens de Consulta - Sistemas relacionais são favoravéis por utilizar modo de consulta bastante simples. Eles provêm boas linguagens e interfaces de consultas, enquanto SBDOO's não as têm. As razões para isto são as seguintes:

. as estruturas dos dados são mais complexas e falta um modelo formal de suporte. Algumas linguagens têm sido sugeridas, derivadas ou dos modelos funcionais ou dos modelos semântico de dados, mas ainda não emergiram. Estas tentativas têm demonstrado que não é uma tarefa fácil projetar uma linguagem de consultas simples para este modelo. Existe uma clara ligação entre simplicidade da linguagem de consultas e o poder de expressão do modelo de dados.

. antagonismo - linguagens de consultas e encapsulação: o princípio de encapsulação declara que dados devem ser ocultados do usuário, enquanto que a consulta necessita ver a estrutura interna do objeto.

3- Consultas Declarativas - Pelo fato de SBDOO's serem mais funcionais por natureza, manipulações de dados tendem a ser mais navegacionais. Assim, o usuário perde algumas das características declarativas do sistema relacional. Naturalmente, se se deseja incorporar construtores de conjuntos no modelo, pode-se recuperar algumas destas características declarativas.

4- Interface Relacional - A ausência de interfaces relacionais em SBDOO's é uma desvantagem. Pode ser estranho mencionar isto como um problema. Contudo, do ponto de vista comercial, isto é

uma solução. Nos próximos anos, SR's continuarão aumentando sua fatia no mercado. SQL tem se tornado um padrão de fato. Ela certamente tornar-se-á um padrão para trocas de dados entre sistemas heterogêneos. Então, mesmo se o conceito de objeto tornar-se dominante, a troca de dados entre Sistemas continuará na forma relacional.

5- Velocidade - Certamente, a questão do desempenho será usado como argumento contra SBDOO's, da mesma forma que foi usado, a alguns anos, contra os SR's. Este é sem dúvida um desafio para projetistas de Sistemas. Existem algumas perguntas a serem respondidas, tais como: Pode-se construir um sistema que executará seleções em conjuntos de objetos, tão rapidamente quanto um SR seleciona tuplas numa relação? Pode-se executar a mesma quantidade de transações de crédito/débito por segundo num SBDOO do que num SR?

Nas seções seguintes serão apresentados alguns SOO e definidos alguns conceitos de abstrações que são bastante utilizados nos modelos de dados orientados por objetos.

3.5.3- Atuais Sistemas Orientados por Objetos

Atualmente existem diversos SBDOO em desenvolvimento que oferecem as facilidades de Sistemas Orientados por Objetos, mas são ainda protótipos desenvolvidos em entidades de pesquisa. Podem-se citar como exemplos IRIS [WILK/90], O2 [DEUX/90], ORION [KIM/90] e GERPAC [RICA/87]. A seguir descrevem-se sucintamente as principais características de cada um destes Sistemas.

O IRIS é um protótipo de pesquisa desenvolvido pela Hewlett-Packard e tem como objetivo principal aumentar a produtividade do programador de SBD e prover suporte geral em SBD para o desenvolvimento e integração de aplicações futuras, tais como engenharia da informação, automação de escritório, telecomunicações, etc. Este Sistema é baseado no Modelo de Dado Semântico que suporta tipos abstratos de dados, controle de concorrência, herança,

identidade de objeto e evolução de Esquemas. O modelo de dados do IRIS contém três contruções: objetos, tipos e funções. Atributos de objeto, relacionamento entre objetos e procedimentos de objetos são representados através de funcões. Função é definida sobre tipo, onde sua declaração é separada de sua implementação. Existem tentativas de estender o IRIS a fim de oferecer facilidades na construção de objetos complexos e múltipla herança, conforme fontes bibliográficas disponíveis [WILK/90].

O O2 é um SBDOO em desenvolvimento por um grupo francês denominado Altaïr. Este Sistema além das funcionalidades normais de um SBD, fornece suporte a objeto complexo, identidade de objeto, encapsulação, tipagem, múltipla herança e controle de concorrência. Inclui ferramentas de geração de interface com o usuário (LOOKS) e um ambiente completo de programação (OOPE). Suporta ainda o paradigma de multi-linguagem (CO2 e BasciO2) e provê dois modos de funcionamento : modo de desenvolvimento e modo de execução. No modo de desenvolvimento o programador atua de uma forma interativa na definição de sua aplicação, modificando seu Esquema (removendo ou criando classes, metodos, etc). Quando a aplicação é completada e depurada, pode-se atuar no modo de execução, onde o modelo do usuário será congelado. A partir daí o próprio Sistema se encaregará de otimizar a aplicação do usuário.

O ORION é uma série de SBD's de próxima geração, em desenvolvimento pela Microelectronics and Computer Technology Corporation (MCC), utilizado como veículo de pesquisa para integrar linguagem de programação e SBD. A série ORION inclui três Sistemas: ORION-1 é um Sistema mono-usuário e multitarefa, ORION-1SX é um Sistema cliente/servidor projetado para funcionar em redes locais de computadores e ORION-2 é um Sistema de gerência de objetos distribuídos, onde todos computadores conectados na rede participam na gerência de um SBD compartilhado. O ORION suporta modelos de dados orientados por objetos, gerência de transações, controle de versão, objetos compostos, herança, identidade de objeto e evolução dinâmica de Esquemas.

O GERPAC é um SBDOO em desenvolvimento no Departamento de Engenharia da Computação e Automação Industrial FEE/UNICAMP. Ele é um SGBDOO que suporta o modelo MER/PAC (modelo de dados estendido do modelo Entidade-Relacionamento voltado à descrição dos aspectos semânticos de aplicações PAC (Projeto Auxiliado por Computador)), e controla tanto os aspectos micros (locais) quanto os aspectos macros (globais) de um ambiente de projeto. O GERPAC atualmente suporta conceitos de objetos e relacionamentos complexos, além de um suporte adequado de controle de transações [OLGU/90a] e versões de projeto.

3.6- Conceito de Abstrações

O Mundo Real é compreendido por objetos conceituais, também denominados de entidades, que têm descrições (atributo ou propriedades) associadas a eles e descrições de relacionamentos com outros objetos. Além disso, ocorrem frequentemente operações que provocam modificações nos objetos e nos seus inter-relacionamentos. Descrições dos objetos e operacões estão sujeitas a restrições que definem conceitos e distinguem realidades de outros mundos.

Todo objeto do Mundo Real é simples (definido por si próprio) ou composto (definido como uma abstração de outro objeto). Assim, abstrações são expressas como relacionamentos entre objetos, e têm como finalidade a organização destes objetos.

No Mundo Real, podem ser definidos dois tipos de relacionamentos de abstração: um envolvendo objetos simples para construir um objeto composto e outro envolvendo objeto composto para construir outros objetos mais complexos. Abstrações do primeiro tipo têm relacionamentos de um nível, uma vez que eles são aplicados uma única vez para definir um objeto composto. Abstrações do segundo tipo são representados por relacionamentos de n-níveis, uma vez que eles podem ser aplicados "n" vezes, permitindo um objeto mais complexo ser representado, de uma forma recursiva, por objetos menos complexos. A seguir descrevem-se os conceitos de abstrações que são mais utilizados

[MATT/88].

3.6.1- Classificação

Classificação é a forma de abstração considerada mais importante e de melhor compreensão. Ela consiste em agrupar objetos, que têm propriedades comuns, em um novo objeto para o qual condições uniformes são asseguradas. Em outras palavras, Classificação é a forma de abstração em que um objeto composto (objeto *tipo* ou *classe*), é definido como um conjunto de objetos simples que têm as mesmas propriedades e são denominados instâncias. Isto caracteriza um relacionamento instância-de entre algum objeto simples (instância) e um objeto composto (*classe*). Assim, Classificação é considerada uma abstração com um nível de relacionamento.

Considere, por exemplo, três objetos do mundo real que têm as seguintes propriedades: cor, proprietário, número de registro, potência e preço. Cada objeto pode ser descrito de maneira independente, especificando os valores particulares de suas propriedades, conforme mostra figura 3.1a.

É claro que dificilmente será satisfatório descrever um modelo em termos de informações factuais específicas dos objetos. Geralmente, é importante abstrair os detalhes de cada objeto e tratá-lo de uma forma mais genérica, sem se preocupar com os valores específicos de cada propriedade de objeto. Classificação proporciona meios importantes para introduzir informações genéricas, permitindo ao modelador referenciar à classe como um representante ou protótipo de suas instâncias (figura 3.1b). As propriedades e restrições aplicáveis a todas instâncias estariam presentes na classe. Por outro lado, a Instanciação, que é o oposto da Classificação, pode ser utilizada para obter objetos que obedecem as restrições associadas com as propriedades especificadas pela classe. Uma vez que um objeto pode ser instância de mais de uma classe, instanciações podem ser aplicadas para encontrar objetos que possuem propriedades em mais de uma classe (verificar objeto a3 na figura 3.2).

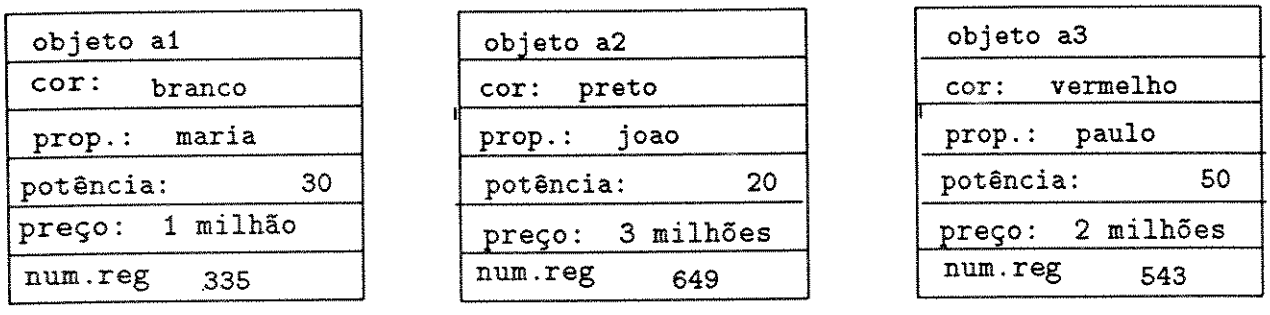


FIGURA 3.1a EXEMPLO DE DESCRIÇÃO FACTUAL

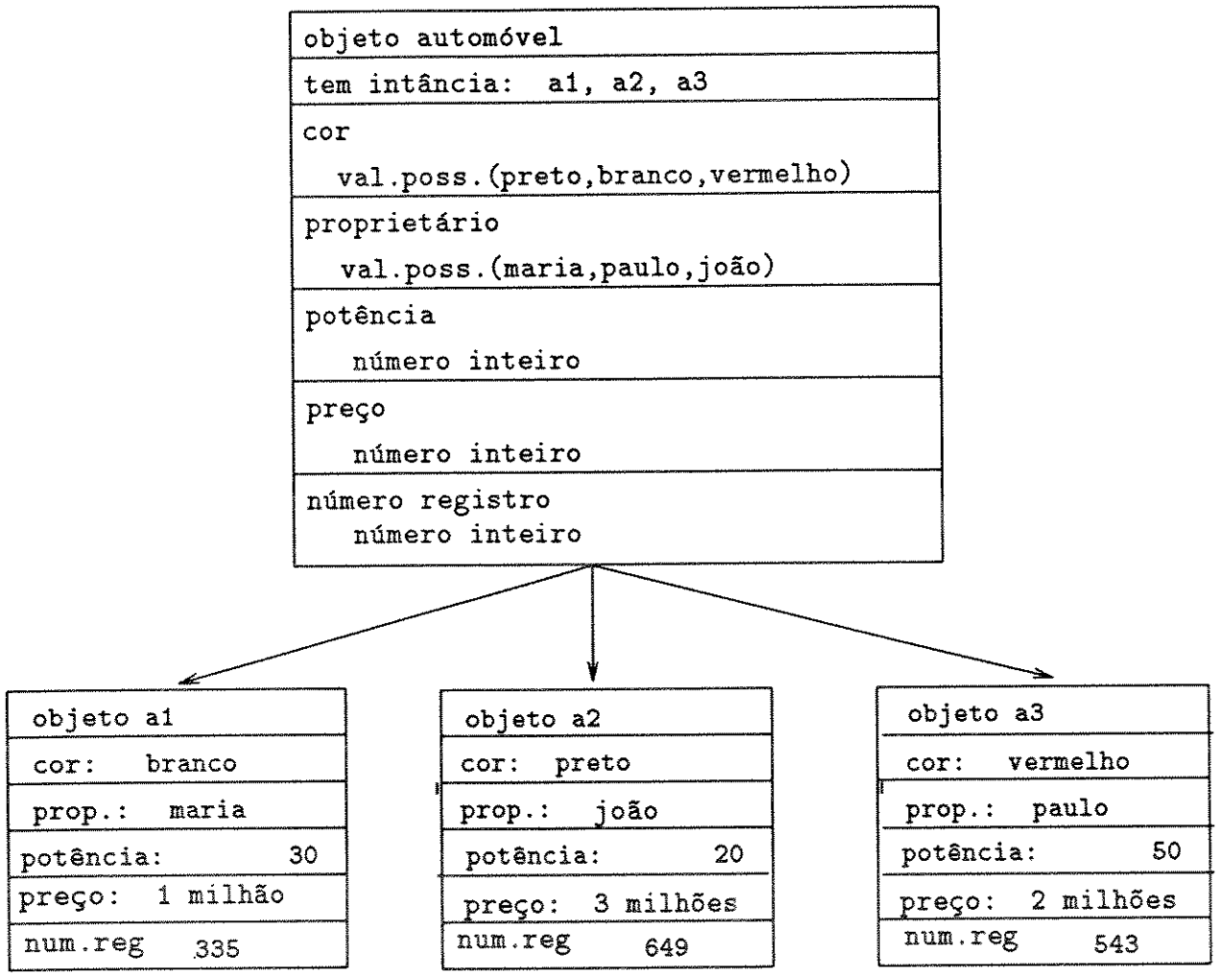


FIGURA 3.1b EXEMPLO DE DESCRIÇÃO POR CLASSIFICAÇÃO

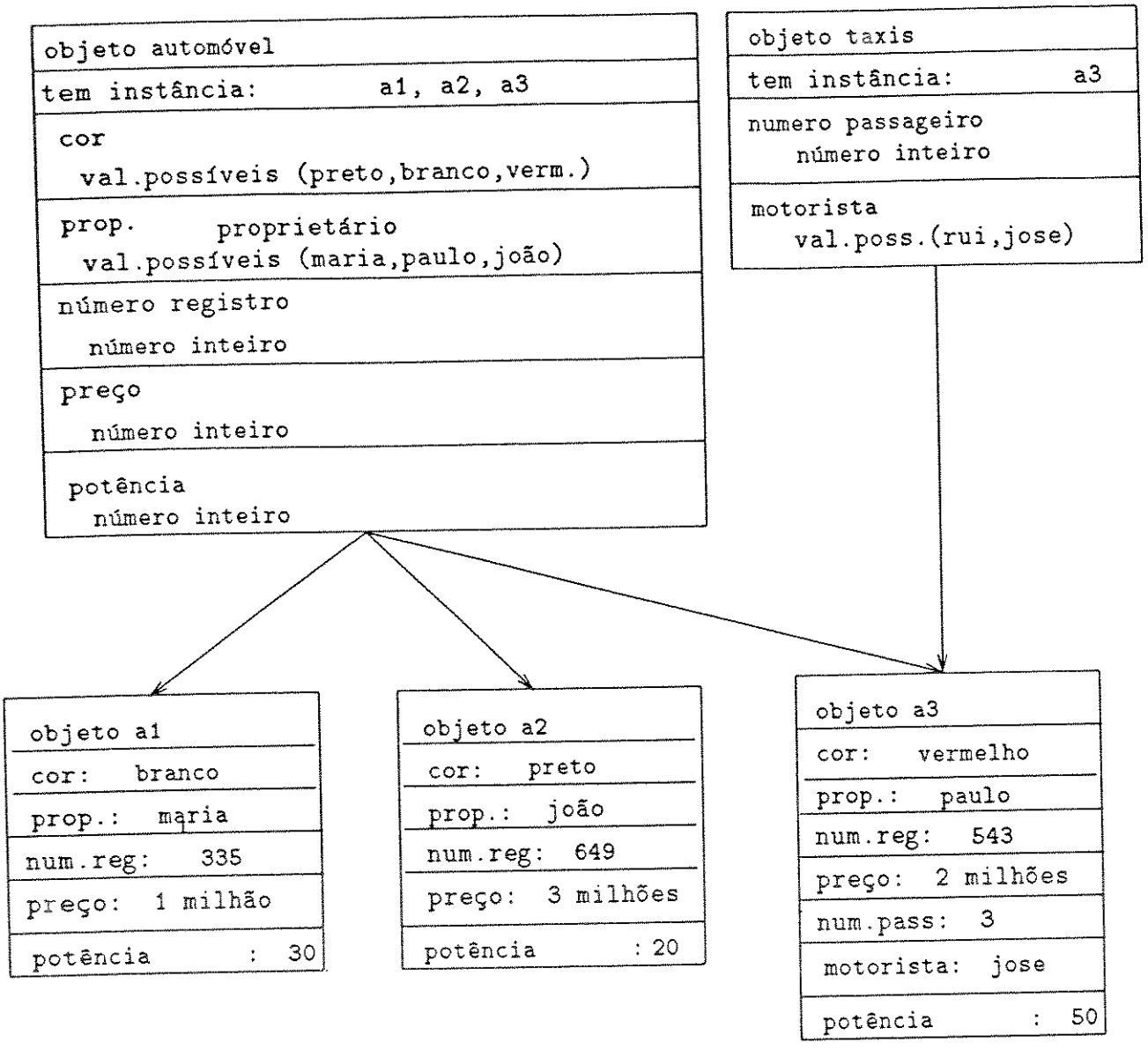


FIGURA 3.2 EXEMPLO DE DESCRIÇÃO DE INSTÂNCIAS

DE MÚTIPLAS CLASSES

3.6.2- Generalização

Generalização é um conceito complementar ao de Classificação. Considere que, no exemplo da figura 3.2, sejam acrescentados caminhões (c1,c2) e barcos (b1). Assim a Classificação terá a descrição mostrada na figura 3.3a. Como no caso da Classificação, é importante ter-se uma forma de referenciar às três classes, ou mesmo a suas instâncias, abstraindo-se de detalhes de cada uma delas. Em outras palavras, deseja-se descrever uma classe mais geral, que capture as características comuns extraídas de outras classes, mas suprima detalhes específicos de cada uma delas. Este mecanismo é provido pela Generalização. Ela permite que objetos compostos mais complexos (denominados de superclasses) sejam definidos como uma coleção de objetos compostos menos complexos (denominados de subclasses), estabelecendo assim um relacionamento **é_um** ou **subclasse_de** entre a super-classe e a subclasse. No referido exemplo, a classe veículo é definido como uma Generalização de automóveis, caminhões e barcos (figura 3.3b).

Uma vez que a Generalização pode ser aplicada recursivamente para especificar outras superclasses de níveis mais alto, ela é considerada um relacionamento de n-níveis que organiza classes dentro de uma **hierarquia de Generalizações** ou **hierarquia é_um**.

Por exemplo, Veículo junto com Aeronave podem ser generalizados para Meios_de_transporte (figura 3.4). Como consequência, obtém-se uma grande homogeneidade entre certas classes, uma vez que classes que têm superclasses comuns são diferentes apenas em detalhes. Este fato não é típico em aplicações convencionais, onde é regra ter-se uma grande heterogeneidade entre objetos tipos (classes) contrastando com uma grande homogeneidade dentro de um objeto tipo (entre instâncias correspondentes).

O processo oposto da Generalização é a Especialização. De acordo com esta metodologia, um modelo é primeiro construído em termos de classes mais gerais, e a partir destas lidando com seus subcasos

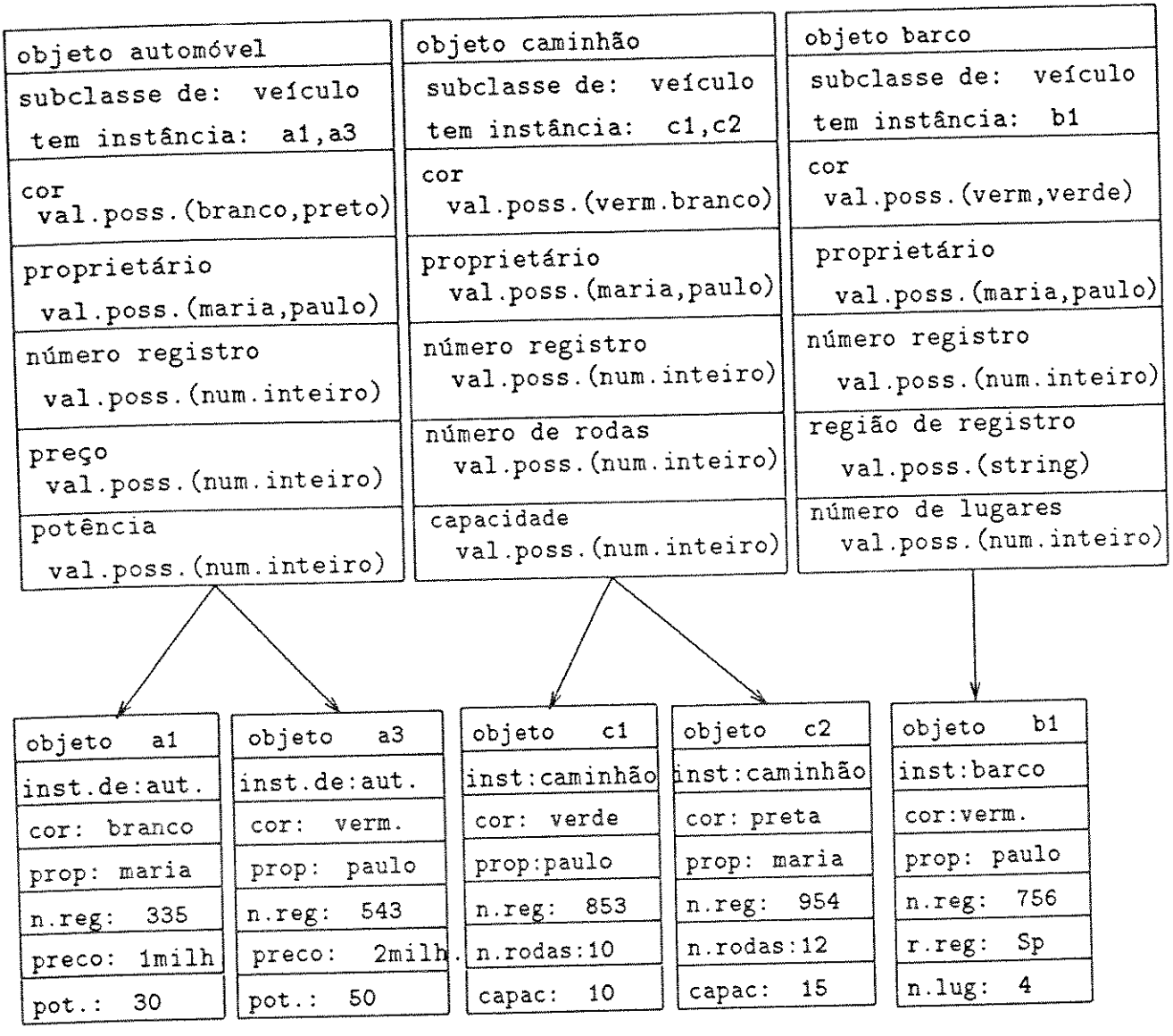


FIGURA 3.3a EXEMPLO DE DESCRIÇÃO POR CLASSIFICAÇÃO

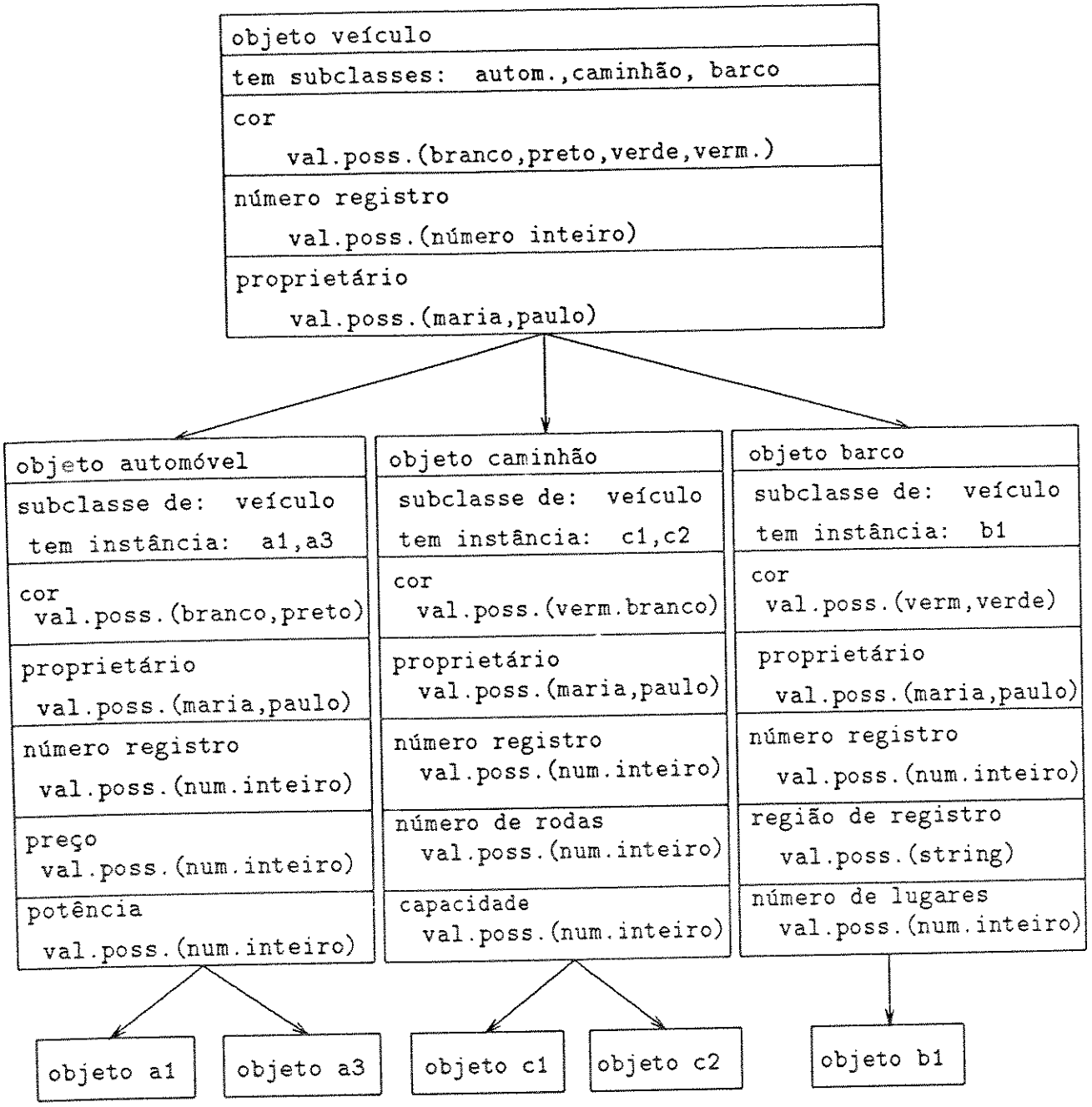


FIGURA 3.3b EXEMPLO DE DESCRIÇÃO POR GENERALIZAÇÃO

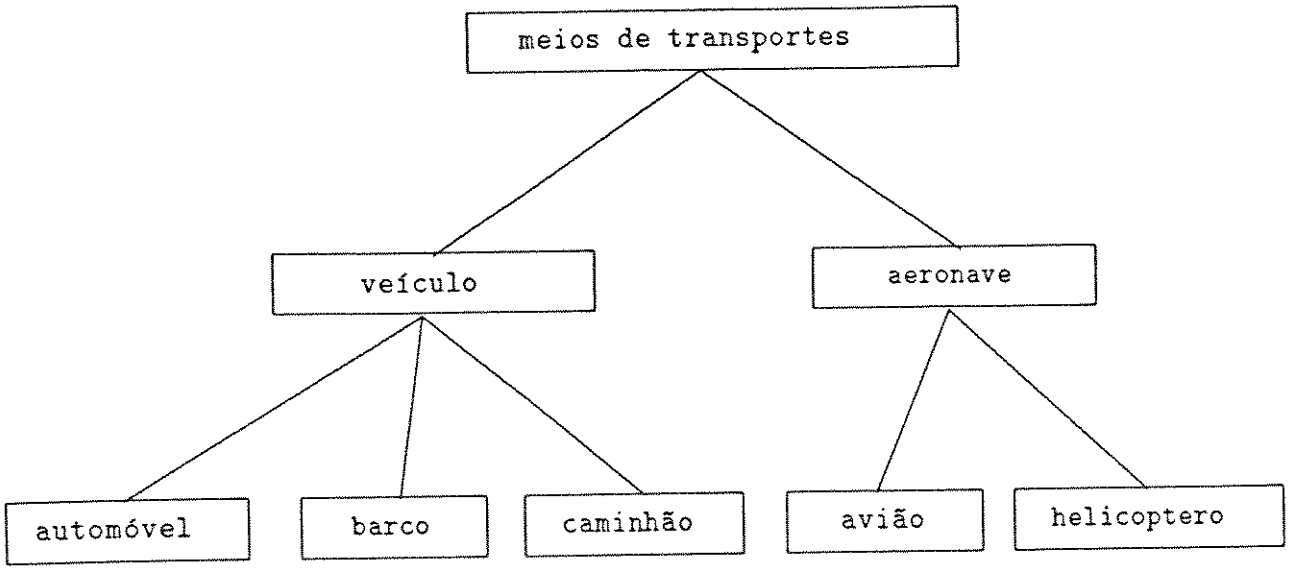


FIGURA 3.4 - HIERARQUIA DE GENERALIZAÇÃO

através de classes mais especializadas.

Vale observar que tanto na Generalização quanto na Especialização, tem-se o mecanismo de herança, ou seja, as características das superclasses são herdadas pelas subclasses.

Uma vez que as propriedades das superclasses são geralmente válidas para suas subclasses, não existe a necessidade de repetir as descrições armazenadas na superclasse para cada subclasse da mesma. Estas propriedades são herdadas por cada subclasse. Como resultado tem-se que:

- descrições podem ser abreviadas sem, contudo, perder a flexibilidade necessária para descrever as propriedades dos objetos;

- alguns erros clássicos são evitados pelo fato de se reduzir a quantidade de repetição nas descrições.

Existem duas formas básicas de ver descrições associadas com uma classe e, consequentemente, para tratar herança. A primeira, que é a forma como modelos de representação do conhecimento vêem classes - classes simplesmente caracterizam protótipos de instâncias. Têm-se, como consequência, que as propriedades herdadas podem ser contraditas pelas subclasses de uma classe ou mesmo por suas instâncias. A segunda, que representa a abordagem seguida pelos modelos de dados semânticos, considera a descrição associada com uma classe como condição necessária (propriedades e restrições) que cada instância ou subclasses deve satisfazer.

3.6.3- Associação Elemento

Algumas vezes, é necessário agrupar objetos, não pelo fato deles terem as mesmas propriedades (Classificação), mas com o intuito de descrever propriedades de grupos de objetos como um todo.

No exemplo de tipos de transportes, considere que cada

veículo tem uma propriedade adicional para expresssar os anos de uso do mesmo. Considere, também, que se deseja representar o conjunto de Veículos de uma determinada pessoa, (por exemplo Paulo) para expressar a seguinte propriedade: média do número de anos que aquela pessoa usa um veículo (para simplificar, veículos são possuídos por uma única pessoa e eles não podem ser vendidos). Considerando que propriedades só existem quando associadas a objetos, é necessário que se defina um novo objeto para conter essas propriedades.

Surge como primeira idéia definir uma nova classe tendo os veículos possuídos por Paulo como instâncias. Contudo, levando-se em consideração o conceito de Classificação, conclui-se que este modelo não é consistente. Segundo aquele conceito, uma classe assegura as propriedades e restrições que lhes são associadas, e que são possuídas e satisfeitas pelas instâncias da mesma classe. Consequentemente, cada instância de Veículo de Paulo (a3, b1, t1) teria que ter a propriedade: média de anos de uso. Contudo, esta é uma propriedade que nenhum dos objetos (Veículos de Paulo) possui e não a pode satisfazer, pois, esta propriedade não é uma característica de cada objeto e sim do grupo como um todo.

Para solucionar esse problema, utiliza-se o conceito de Associação-elemento. Esse conceito define um objeto Conjunto que é introduzido para representar aqueles tipos de propriedades (propriedades referentes a grupos de objetos). Em outras palavras, **Associação Elemento** é uma forma de abstração, em que alguns objetos chamados **Elementos**, são considerados como um objeto de mais alto nível, chamado **Conjunto**. Os detalhes dos objetos Elementos são suprimidos, enquanto que as propriedades do grupo são enfatizadas no objeto Conjunto, estabelecendo um relacionamento **elemento_de** entre Elementos e Conjunto.

O conceito de Associação Elemento também é conhecido como agrupamento ou particionamento. Ele é considerado como relacionamento de um nível.

3.6.4- Associação Conjunto

Como se pode observar, o conceito de Associação Elemento corresponde a uma parte da teoria de conjunto da matemática. Portanto, em adição ao relacionamento de um nível descrito acima (Associação Elemento), que constrói objetos compostos (conjuntos) a partir de objetos simples (elementos), é necessário também ter uma forma de expressar relacionamento entre objetos compostos (conjuntos), para que se tenha um suporte completo na teoria de conjunto. Esses relacionamentos incorporam o conceito de **Associação Conjunto**, que são denotados como relacionamento **subconjunto-de**. Isto significa que o conjunto Veículo-de-Paulo é um subconjunto do conjunto do Veículos-da-família-do-Paulo , que agrupa não somente os Veículos-de-Paulo, como também aqueles de sua mãe e seu pai. Assim, o relacionamento Associação elemento corresponde a relação matemática "$\in$" (pertence a), enquanto o relacionamento Associação Conjunto corresponde a relação "$\subset$" (está contido). Desta maneira, o relacionamento **subconjunto-de** é um relacionamento de "n" níveis, uma vez que ele pode ser aplicado recursivamente definindo, juntamente com o relacionamento **elemento_de**, uma hierarquia de objetos denominado **hierarquia de associação** (figura 3.5).

Da mesma maneira, que toda instância de uma classe, na hierarquia is-a, é também uma instância da superclasse, elementos de um conjunto são também elementos do conjunto superconjunto . Assim também, objetos podem ser elementos de varios conjuntos, como conjuntos podem ser subconjuntos de vários outros conjuntos.

Propriedades de conjunto

Não existe herança numa associação, pois as propriedades descritas nos conjuntos não são caracteristicas dos elementos em si. Essas propriedades são características do conjunto como um todo, e estão associados às propriedades dos elementos.

Adicionalmente as propriedades de conjunto, descrições de

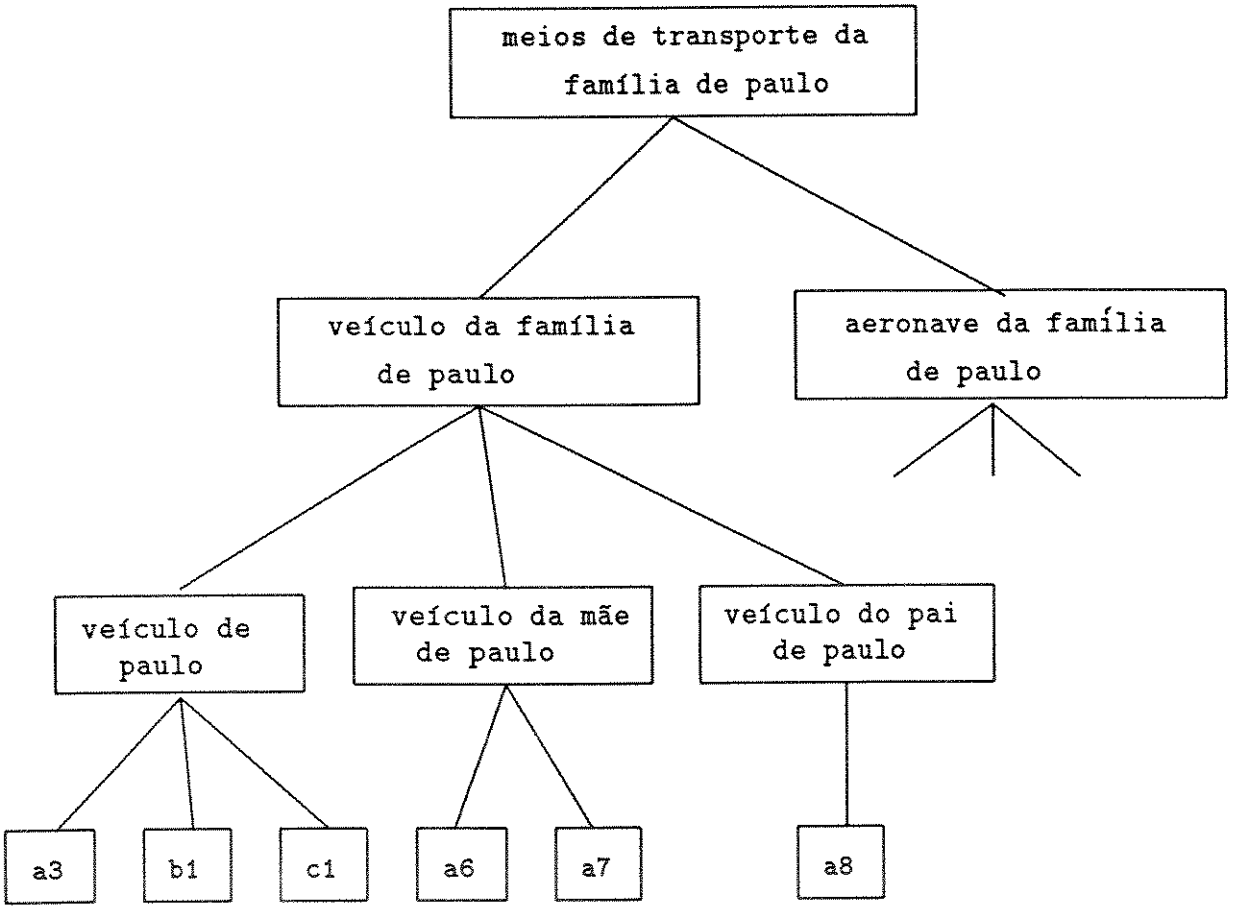


FIGURA 3.5 EXEMPLO DE UMA HIERARQUIA DE ASSOCIAÇÃO

conjunto contêm outras propriedades que são válidas para cada elemento do conjunto. Por exemplo: todo elemento do conjunto Veiculo_de_paulo tem Paulo como valor da propriedade "proprietário". Esta propriedade é chamada de **contrato de sociedade.**

Um objeto conjunto pode tornar-se um subconjunto de outros, restringindo o contrato de sociedade. Por exemplo, elementos do conjunto Veículo_da_família_de_paulo têm Paulo, seu pai ou sua mãe como valor da propriedade "proprietário", enquanto que os elementos do subconjunto Veículo_de_paulo têm seus valores restrito a Paulo. Por conseguinte, restringindo os contratos de sociedade, o cálculo das propriedades do conjunto são restritos a um subgrupo de elementos do superconjunto determinando assim, diferentes valores para essas propriedades. Por exemplo, o cálculo da propriedade "média_de_anos_de_uso" do conjunto Veículo_da_mãe_de_paulo será restrito somente aos da mãe de Paulo e não aos da família toda.

3.6.5 Agregação de Elementos e de Componentes

Muitas vezes não é conveniente tratar objetos como entidades atômicas, Classificação, Generalização ou Associação, mas sim como uma composição de outros objetos. Considere como exemplo o objeto Automóvel. Pode ser importante para uma determinada aplicação, expressar o objeto Automóvel como sendo composto de motor, rodas e uma carroceria; suprimindo, no primeiro instante, detalhes específicos dos objetos constituintes (estes objetos possuem também propriedades e são construídos de outros objetos). Toda esta concepção forma um conceito de abstração chamado **Agregação**, segundo o qual uma coleção de objetos é visto como um objeto de mais alto nível. O objeto Automóvel é tratado como uma agregação dos objetos Motor, Rodas e Carroceria.

Como nos outros conceitos de abstração, Agregação envolve dois tipos de objetos: simples e composto. Numa Agregação, objetos simples são objetos atômicos e não podem ser decompostos. Esses objetos são chamados **Elementos**. Juntos eles controem os objetos compostos de mais baixo nível, ou simplesmente objetos **Componentes**.

Isto, consequentemente, pode ser utilizado para construir Componentes mais complexos de mais alto nível, formando assim uma **Hierarquia de Agregação** ou **Hierarquia parte_de** (figura 3.6). Entre objetos Componentes existe um relacionamento **subcomponente_de**, enquanto entre o objeto Componente e o Elemento existe um relacionamento **elemento_parte_de**. Relacionamento **subcomponente_de** estabelece o conceito elementar que denomina-se **Agregação de Componentes**. Este conceito tem a mesma filosofia dos conceitos de Associação Conjunto e Generalização, na medida que estes últimos também estão associados com objetos mais complexos. Já o relacionamento **elemento_parte_de** estabelece o conceito elementar que denomina-se **Agregação de Elementos**. Este conceito tem a mesma filosofia dos conceitos de Associação Elemento e Classificação. Naturalmente, o relacionamento **subcomponente_de** é uma extensão do relacionamento **elemento_parte_de**.

Propriedades da Agregação

O conceito de Agregação está associado a noção de composição, uma vez que ele expressa, por exemplo, a idéia que Automóvel consiste de um motor, rodas e uma carroceria. Mais estritamente, pode-se dizer que o conceito de Agregação descreve propriedades necessárias que um objeto deve possuir, a fim de que o mesmo exista consistentemente. Em outras palavras, objetos Componentes não podem existir sem seus elementos. Certamente é difícil imaginar um automóvel sem motor ou sem carroceria. Observe-se que este requisito torna a Agregação um conceito bastante diferente dos outros. Classes podem existir sem suas instâncias, e podem ocorrer conjuntos vazios na Associação. Desta forma, as propriedades dos objetos Agregação, diferentemente dos outros conceitos de abstração, não são para ser interpretados como propriedades deles, mas sim como outros objetos representando partes dos objetos Agregações. Observe-se também que automóveis são sempre construídos a partir dos mesmos componentes. Além disso, objetos com componentes diferentes são provavelmente objetos diferentes. Por exemplo, se alguém afirma que seu objeto não é uma agregação de motor, rodas e carroceria, mas sim uma agregação de motor, asas e carroceria, este objeto certamente não está descrevendo

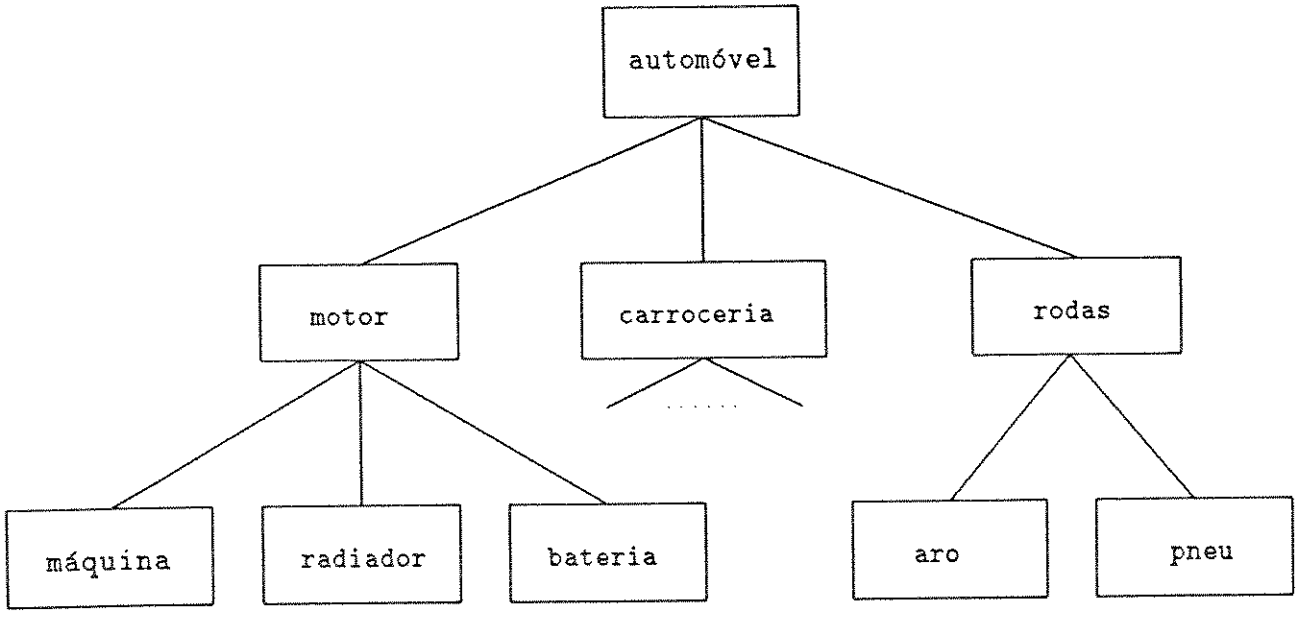


FIGURA 3.6 EXEMPLO DE UMA HIERARQUIA DE AGREGAÇÃO

automóveis, mas talvez um avião.

3.7- Resumo

Basicamente o objetivo dos Sistemas de Banco de Dados Orientados por Objetos (SBDOO's) é integrar tecnologia de Sistemas de Banco de Dados (SBD's) (compartilhamento de dados, segurança dos dados, persistência, liguagens de consultas, etc) com a abordagem Orientação por Objeto em um mesmo Sistema.

Três fatores contribuíram para o interesse em SBDOO's:

1- Grupos que trabalham em Sistemas Orientados por Objetos (SOO's) necessitam das funcionalidades dos SBD's.

2- A comunidade de SBD percebeu o impedimento do não casamento de tecnologia de SBD com linguagem de programação.

3- Interesse de grupos de SBD na tecnologia de Orientação por Objeto para obter ferramentas mais poderosas para modelar o Mundo Real.

Algumas características de SOO são ressaltadas tal como, Encapsulação, Identidade de Objeto, Compartilahmento de Objeto, Classe e Herança.

Encapsulação é o princípio que permite modelar ao mesmo tempo dados e operações. Assim, um objeto tem uma parte de interface e outra de implementação.

Compartilhamento de objetos é uma consequência do conceito de Identidade de Objeto. Num modelo baseado em Identidade de Objeto, dois objetos podem compartilhar um mesmo componente.

Classe em um SOO, descreve um conjunto de objetos com as mesmas características. Ela tem duas partes: a parte de interface para

o objeto e a parte de implementação do objeto. A parte de interface é visível ao usuário da classe e a parte de implementação é vista somente pelo projetista da classe. A interface consiste de uma lista de operações com seus respectivos parâmetros. A implementação consiste da parte de dados e da parte de operações. A estrutura interna dos dados do objeto é descrita na parte dos dados. Na parte de operações é feito um programa para cada operação da interface.

Herança é talvez um dos conceito mais poderoso em orientação por objeto. Este conceito permite objetos de estruturas diferentes compartilharem operações que lhes são comuns. É uma ferramenta poderosa de modelagem na medida que permite uma descrição concisa e precisa do Mundo Real. Herança auxilia na reutilização de códigos por que o programa está num nível onde vários objetos podem compartilhá-los.

Para efeito de comparação utilizou-se os Sistemas Relacionais por estarem mais em voga.

No próximo capítulo será apresentado um núcleo de gerência de objetos denominado UNICOSMOS, utilizado pelos SGBD's para gerenciar as funções de baixo nível. No entanto existem limitações do UNICOSMOS que o impede de dá suporte aos SGBD's nas abordagens de orientação por objeto discutidas no presente capítulo.

Capítulo 4

UNICOSMOS (UNICAMP/FEE Object Storage Management System)

4.1- Introdução

O **UNICOSMOS**, que inicialmente foi chamado de SIGA (Sistema de Gerência de Armazenamento Associativo), é um Sistema de software utilizado como núcleo de SGBD, independente de modelo de dados [RICA/86], [DELG/88], [MAGA/89], [OLGU/90a] e [OLGU/90b]. Evoluiu a partir da proposta de CORAS (Core System for Associative Storage) ([ENC/83] e [WU/83]), onde todos os elementos do mundo real em estudo são considerados como objetos, independentemente da interpretação associada (entidade, atributo, ou valor associado ao atributo). Atributos são representados através de uma classe especial de objetos denominada objeto-relação; associação entre entidades e valor de atributos são representados através de uma associação binária (triade) que liga um valor a um objeto através de um atributo. Um outro tipo de objeto, o objeto-conjunto, permite definir classes de elementos que também são associados através de objeto-relação. Embora esta representação seja muito adequada do ponto de vista do alto nível de abstração alcançado, sob o ponto de vista da implementação há diversas deficiências que devem ser levadas em conta, entre elas:

acesso: o acesso associativo simulado através de *hashing* pode ser limitante com relação a previsão da quantidade de elementos que pode ser manipulada na tabela *hash*. Nesta abordagem, é difícil prever esta quantidade, pois cada elemento representado - entidades, atributos, valores, conjuntos - ocupa uma entrada nesta tabela;

representação dos dados: todos os elementos do mundo CORAS são representados através de seu nome - uma sequência de caracteres com dimensão pré-definida. Não há forma de tratar diretamente outros tipos de dados ou casos em que se desejasse representar elementos com maior dimensão que a estabelecida pelo Sistema;

velocidade: todas as associações, sejam relacionamentos entre conjuntos ou ocorrências de atributos , são representadas através das tríades. Para agilizar o acesso a estas informações, são armazenadas as três permutações de cada tríade, ou seja, as três possíveis representações para uma associação binária. Esta redundância acarreta uma sobrecarga de controle para a implementação, além da questão da ocupação de espaço;

interpretação: não há distinção entre entidades e valores, devendo qualquer interpretação ser suportada pela aplicação.

Para tornar este núcleo mais eficiente, embora reduzindo o nível de abstração suportado, o UNICOSMOS foi proposto. O Sistema será descrito nas próximas seções, englobando seus elementos primitivos, arquitetura interna e uma visão funcional.

4.2- Elementos Primitivos

Os elementos primitivos do UNICOSMOS são objetos-unicosmos. Objeto-unicosmos é o nome dado a cada objeto representado no UNICOSMOS. Os objetos-unicosmos têm a função de representar classes, que internamente descreverão os elementos primitivos do modelo suportado pelo SGBD implementado sobre o UNICOSMOS. Sendo assim estes objetos-unicosmos podem representar relações, tipos de registros, tipos de conjuntos, tipos de relacionamentos, tipos de entidades, dependendo do modelo em questão. Objetos-unicosmos podem ser de três tipos:

Simples - Representam uma classe, à qual podem estar associados atributos.

Conjunto - Definem agrupamentos de outros objetos-unicosmos, simples, conjuntos ou mesmo relações.

Relação - Permitem estabelecer relacionamentos binários entre

outros objetos-unicosmos que não sejam do tipo relação.

4.3- Identificação e Caracterização de Objetos-unicosmos

Os objetos-unicosmos são identificados por seu nome. Este nome constitui a chave de acesso a todas informações sobre o objeto-unicosmos. Para garantir a flexibilidade do Sistema, este nome não é necessariamente único para cada objeto-unicosmos, podendo ser definidos sinônimos que definem igualmente este acesso, contudo os nomes não podem ser repetidos. A chave de acesso para cada objeto-unicosmos é único para todo o sistema.

Cada objeto-unicosmos é caracterizado por seus atributos. No UNICOSMOS estes atributos são armazenados em máscaras associadas ao objeto-unicosmos, que descrevem cada grupo de propriedades do objeto-unicosmos. Estas máscaras contêm informações tais como nome e tipo de dado associado a cada atributo, definindo as características comuns entre os elementos que constituem as instâncias do objeto-unicosmos (suas ocorrências).

A caracterização de um objeto-unicosmos pode não ser única. Em alguns casos, pode haver mais de um conjunto de atributos associado a um objeto-unicosmos, representando diferentes tipos de informação além da descrição de propriedades do objetos-unicosmos. Para atender estas necessidades o UNICOSMOS permite a declaração de mais de uma máscara de atributos associada a cada objeto-unicosmos, sendo cada máscara associada a um rótulo que a identifica. A forma de utilização das diferentes máscaras que podem ser associadas ao objeto-unicosmos depende da aplicação, de forma a tornar este Sistema flexível às diversas necessidades.

Ocorrências são instâncias do objeto-unicosmos sendo definidas através de valores associados aos seus atributos. Essas instâncias que podem ser visualizadas como registros lógicos na base de dados constituem a unidade de manipulação dentro do objeto-unicosmos. O conceito de ocorrência pode ser utilizado para

mapear tuplas, registros, entidades, relacionamentos, de acordo com o modelo suportado.

O UNICOSMOS não permite a definição e tratamento de Tipo Abstrato de Dados. Todos elementos internos são armazenados na forma de caracteres, estando a conversão a outros tipos subordinada aos gerenciadores que o utiliza. No entanto, é permitido associar informações à definição de um tipo de dado, o que permite verificar, por exemplo, se o valor do elemento que está sendo armazenado é válido. Especificações de consistência a nível de interrelacionamento dos valores dos atributos de um objeto-unicosmos ou entre objetos-unicosmos distintos devem ser controlados pelo SGBD.

4.4- Tratamento de Objetos-unicosmos

Cada objeto-unicosmos contém basicamente três tipos de informação:

Nome - permite sua identificação;

Atributos - caracterizam as propriedades da classe por ele representada;

Ocorrências - representam suas instâncias, cada uma delas com uma combinação diferente de valores associados a atributos.

A definição de um objeto-unicosmos passa por duas fases: criação de seu nome e a caracterização da classe por ele representada, através de seus atributos. Dois objetos-unicosmos distintos podem ter um mesmo conjunto de propriedades - atributos - sob o ponto de vista do usuário UNICOSMOS (por exemplo, o atributo "endereço" pode estar associado tanto a classe dos "professores" quanto à classe dos "alunos"), mas internamente são associados conjuntos de valores distintos para cada atributo.

Os elementos dos conjuntos de valores de cada atributo (os

valores propriamente dito) são definidos à medida que as ocorrências da classe são definidas. Cada ocorrência é especificada por um conjunto de associações atributo-valor. Internamente, se o valor especificado para um atributo já foi definido através de outra ocorrência, então apenas a associação deste valor com a nova ocorrência é estabelecida; caso contrário, o valor é inicialmente registrado no conjunto de valores correspondentes para então ser definida a associação.

A associação de diferentes valores a um atributo em uma mesma instância é permitida, caracterizando assim um atributo multivalorado. Para permitir a representação de atributos que não devem ser multivalorados (por exemplo, "data de nascimento"), deve ser feita uma distinção a nível de definição de atributos, que pode posteriormente ser alterada desde que se mantenha a consistência em relação aos dados já armazenados.

As informações sobre objetos-unicosmos podem ser atualizadas sem impacto para o Sistema. Um mesmo objeto-unicosmos pode ter diversos nomes associados através da definição de sinônimos. A remoção de nomes não traz impactos ao Sistema, desde que pelo menos um nome permaneça associado ao objeto-unicosmos. Atributos podem ser acrescidos às propriedades do objeto-unicosmos; Nome e funcionalidade (monovaloração ou multivaloração) de atributos podem ser alterados. A remoção de um atributo só é concretizada se não houver nenhum valor associado ao atributo. No caso de haver valores associados ao atributo, o Sistema encarrega-se de alertar o usuário que existem valores associados ao atributo e a partir dai o usuário terá que remover estes valores para depois remover o atributo desejado.

4.5- Tratamento de Ocorrências

Conforme já mencionado anteriormente, ocorrências representam as instâncias dos objetos-unicosmos. Dependendo da classificação do objeto-unicosmos (simples, conjunto ou relação), o tratamento de suas ocorrências pode diferir. No entanto, um conceito

comum as instâncias de objetos-unicosmos é a identificação interna da ocorrência, uma chave interna de acesso às informações sobre as propriedades da instância. O usuário do Sistema não tem acesso a esta chave, que é definida e manipulada internamente, de forma qua a única maneira de acessar instâncias é através de cláusulas especificando atributo e valores (na verdade, existe outra forma de acesso que será discutida no item 4.6.5). Esta chave é independente dos valores de atributos da instância, de forma que alterações destes valores não acarretam em inconsistências sobre as informações já existentes (como será visto, posteriormente, no caso de instâncias de objetos-unicosmos relação).

O tipo de ocorrência mais elementar é a que está associada ao objeto-unicosmos simples. As ocorrências são totalmente definidas através dos valores associados aos atributos, sem depender das associações do objeto-unicosmos com outros.

Para objetos-unicosmos conjuntos, há dois conceitos de instâncias associados: ocorrências como definidas para objetos-unicosmos simples e/ou objetos-unicosmos pertencentes ao conjunto. Um tratamento especial destas duas formas de ocorrências ou mesmo a interligação entre elas (por exemplo, definindo um atributo "nome de elemento" para o objeto-unicosmos conjunto) é de responsabilidade da aplicação. Sob o ponto de vista do UNICOSMOS são duas informações distintas.

A remoção de uma destas ocorrências implica na remoção de todas as associações atributos-valores de uma determinada instância. Se algum dos valores associados ficar, devido a esta operação, sem ocorrências associadas, então este valor é também removido da base de dados. Caso ainda existam associações do valor com outras instâncias, o valor permanece armazenado.

A atualização de um valor de atributo em uma ocorrência significa mudar a associação entre a instância e o elemento do conjunto de valores (o valor). Se não existir o novo valor que foi

definido para a instância, este é inicialmente criado para então ser definida a associação.

O conceito de ocorrência de um objeto-unicosmos relação exige uma análise mais detalhada. Inicialmente, é interessante relembrar os conceitos de associação a um nível mais abstrato para depois particularizar para o caso de UNICOSMOS. Pode-se dizer que existem basicamente dois tipos de associação entre conjuntos, como mostrado na figura 4.1: as que definem as propriedades de um elemento (por exemplo, p1 e 'José') e as que definem relacionamentos entre elementos (por exemplo, p1 e o1). Note-se que especificar orientação como uma classe distinta e não apenas como arcos de associação, é a forma de permitir caracterizar cada elemento de orientação, permitindo associar propriedades a ela.

No UNICOSMOS, as associações também são definidas a nível de conjuntos. No primeiro caso, na caracterização de propriedades de elementos, esta associação é definida através dos atributos, sendo no entanto que cada objeto-unicosmos tem associado um conjunto de valores particular a cada atributo. Haveria internamente, neste exemplo, dois conjuntos de valores "nomes de professores" e "nomes de alunos", mesmo que o nome dos atributos fosse o mesmo ("nome") para os dois objetos-unicosmos.

O segundo tipo de associação é estabelecido através de objetos-unicosmos relações e tríades. Desta forma, para se permitir uma associação entre um elemento da classe dos 'professores' a um elemento da classe 'orientação', seria necessário criar uma tríade envolvendo os dois objetos-unicosmos - no exemplo, através de um objeto-unicosmos relação 'orientador'. Uma ocorrência de 'orientador' não tem, neste caso, propriedades; apenas define uma associação entre pares de elementos (por exemplo, [p1,o1] e [p2,o2]). É interessante observar que, como estas associações são definidas em termos de identificadores internos, são independentes de valores de atributos dos elementos envolvidos, não sendo afetadas por atualizações nos objetos-unicosmos envolvidos (ao contrário do que ocorre em sistemas

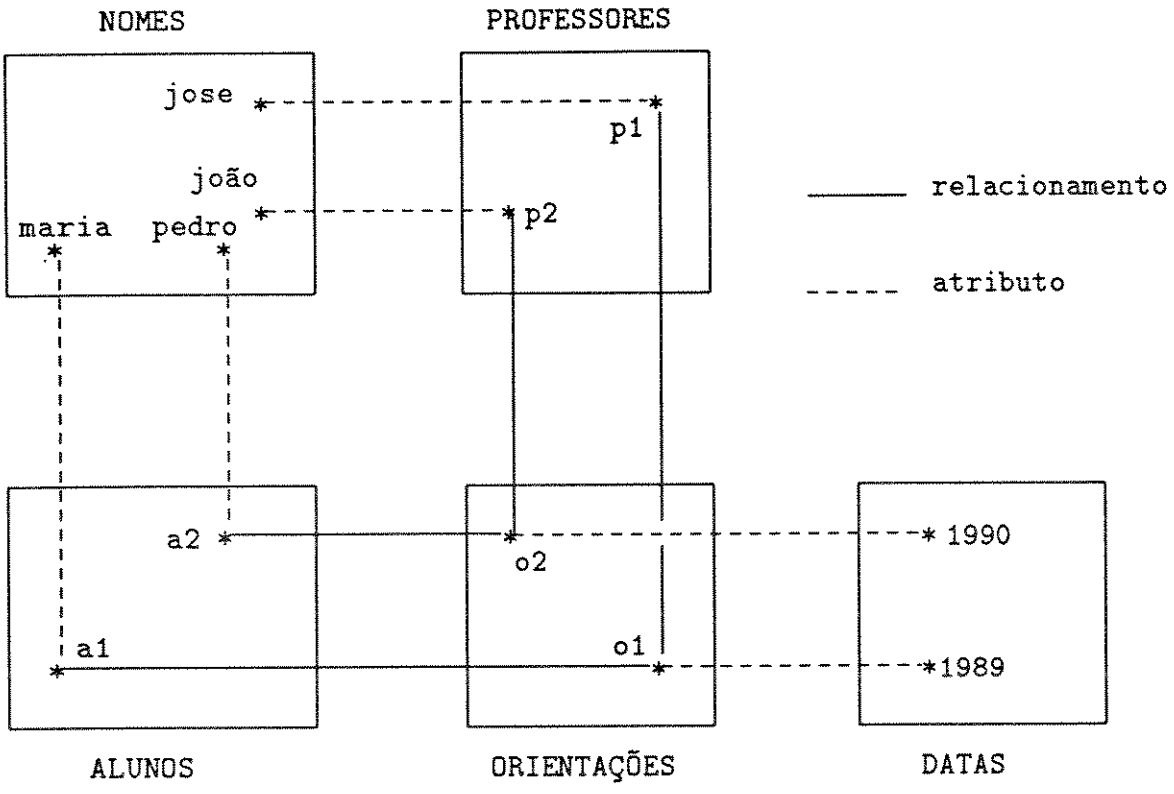


FIGURA 4.1 ASSOCIAÇÕES ENTRE CONJUNTOS

que suportam tabelas).

A remoção de ocorrências de objetos-unicosmos relações envolve outros cuidados. Algumas vezes é interessante que a remoção de um relacionamento provoque a remoção de ocorrências (conceito de dependência), enquanto outras vezes isto não é desejado . Para enquadrar estes dois casos, é definida uma subclassificação do objeto-unicosmos relação, que especifica 'propagação' ou 'não-propagação'. Em uma tríade R(A,B), por exemplo, a remoção de uma ocorrência do objeto A implicará na remoção das ocorrências de B associadas através de R. No entanto, a remoção de uma ocorrência de B acarretará a remoção das ocorrências de A associadas via R se e somente se sua subclassificação especificar a 'propagação'. Esta subclassificação constitui uma propriedade do objeto-unicosmos como um todo e não apenas de seus elementos, como é o caso de atributos.

4.6- Arquitetura Interna

O UNICOSMOS armazena as informações distribuídas entre diversos domínios internos. São eles:

- Domínio de Nomes;
- Domínio de Tríades;
- Domínio de Valores;
- Domínio de Ocorrências;
- Domínio de Acesso;
- Domínio de Objetos.

A seguir, cada um destes domínio será descrito sucintamente, sendo que maiores detalhes sobre suas estruturas internas podem ser encontradas em [RICA/86].

4.6.1- Domínio de Nomes

O Domínio de Nomes, a partir do qual os demais domínios são acessados, constitui a interface do acesso entre o usuário e as

informações armazenadas sobre o objeto-unicosmos. Este acesso é estabelecido através do nome do objeto-unicosmos. Nomes não podem ser repetidos, mesmo para objetos-unicosmos de diferentes tipos, mas não necessitam ser únicos para um objeto-unicosmos. Nomes alternativos podem ser criados, constituindo assim os diversos sinônimos do objeto-unicosmos.

Além dos sinônimos de cada objeto-unicosmos, o Domínio de Nomes também mantém informação sobre a classe do objeto-unicosmos (se simples, conjunto ou relação) e seu identificador interno. Este identificador (ID) corresponde ao endereço lógico onde se armazenam as informações no Domínio de objetos-unicosmos referentes a cada objeto-unicosmos individual. No Domínio de Nomes traduz-se o Nome objeto-unicosmos para seu ID, de forma que os demais domínios referenciam o objeto-unicosmos apenas por este Identificador e não por seu nome. Sinônimos são dois ou mais nomes distintos com o mesmo ID associado. A troca de nome de um objeto-unicosmos não tem, desta forma, qualquer impacto sobre os dados já armazenados.

4.6.2- Domínio de Tríades

O Domínio de Tríades mantém informações sobre os relacionamentos binários (as tríades) entre os objetos-unicosmos. Da mesma forma que no CORAS, as três permutações de uma tríade são armazenadas de forma que a associação possa ser consultada a partir de qualquer um de seus componentes. Cada objeto-unicosmos que faça parte de pelo menos uma tríade tem associada uma entrada no Domínio de Tríades. Esta entrada associada a cada objeto-unicosmos é acessada através do Identificador Interno deste domínio (IR).

No Domínio de Tríades, os objetos-unicosmos associados são identificados por seus Identificadores Internos do Domínio de Objetos (ID). Desta forma, a partir do Domínio de Tríades é possivel acessar o Domínio de Objetos, que centraliza a comunicação com os demais domínios.

4.6.3- Domínio de Valores

O **Domínio de Valores** constitui o depósito dos conjuntos de valores definidos para os atributos dos objetos-unicosmos. Para cada atributo definido no domínio de Objetos existe uma entrada no Domínio de Valores, acessada através de uma Identificação Interna (IC) que endereça uma "seção" do domínio que contém o conjunto de todos os valores ao atributo. Cada um dos elementos deste conjunto (ou seja, cada valor) tem associado um outro Identificador Interno deste domínio, o Identificador do valor (IV).

Valores estão associados a ocorrências. A informação sobre quais valores estão associados a quais ocorrências também está armazenada neste domínio. A cada ocorrência está associado um Identificador Interno do Domínio de Ocorrências (IO). A cada valor armazenado no Domínio de Valores está associada uma lista de IO, com pelo menos um elemento. Cada elemento desta lista corresponde a uma ocorrência que está associada a este valor.

É importante notar que o Domínio de Valores não suporta o conceito matemático de domínios, pois, a dois atributos distintos são associadas listas de valores distintas, mesmo que os atributos tenham o mesmo tipo de dado associado. Entretanto, para cada atributo não há repetição de valores, mesmo que diversas ocorrências distintas atribuam um mesmo valor para o atributo, evitando desta forma a redundância existente em sistemas que manipulam registros ou tabelas.

A representação Interna dos valores armazenados está na forma de caracteres. Não há limitação para a dimensão de cada valor, nem a obrigatoriedade de dimensões pré-fixadas. Caso seja interessante fixar a dimensão que a representação de um atributo vai assumir em uma aplicação suportada pelo UNICOSMOS, esta tarefa deverá ser executada pela própria aplicação através do tratamento de regras, por exemplo.

4.6.4.- Domínio de Ocorrências

Ocorrências são instâncias associadas a objeto-unicosmos. A cada ocorrência está associada uma entrada no Domínio de Ocorrências, endereçada por um Identificador Interno (IO). Uma ocorrência de um objeto-unicosmos é definida através da associação de valores aos atributos deste objeto-unicosmos.

No Domínio de Ocorrências, estes valores são representados pelos Identificadores Internos do Domínio de Valores (IV). Assim, a informação completa sobre uma ocorrência é composta por sua Identificação (IO) associada a uma lista de pares [identificação do atributo, IV do valor associado], onde cada elemento da lista define uma associação atributo-valor.

É interessante notar que não há restrições quanto à unicidade da Identificação do atributo nesta lista, de forma que atributos multivalorados não representam nenhum impacto sobre a estrutura de armazenamento do Sistema (dois ou mais elementos da lista com a mesma Identificação de atributo). Por outro lado, atributos que não tenham valor associado em uma ocorrência (conceito de valor nulo) não ocupam espaço de armazenamento; simplesmente não há um elemento nesta lista associado a este atributo.

4.6.5- Domínio de Acesso

A informação sobre a identificação de cada instância de um objeto-unicosmos está definida no Domínio de Ocorrências. A cada objeto-unicosmos está associado um conjunto de instâncias, que constituem suas ocorrências. Desta forma, o Domínio de Ocorrências contém a definição de todas as ocorrências de todos os objetos-unicosmos definidos.

O Domínio de Acesso define quais ocorrências, entre todas as definidas no Domínio de Ocorrências, estão associadas a um objeto-unicosmos específico. Para cada objeto-unicosmos definido no

Domínio de Objetos está associada uma entrada para o Domínio de Acesso endereçada por seu Identificador Interno IA. A cada IA está associado um conjunto de endereços internos IO para o Domínio de Ocorrências correspondente ao conjunto de ocorrências do objeto-unicosmos.

A especificação deste domínio permite uma forma alternativa de acesso as ocorrências (não-associativo), pois a partir dele é possível acessá-las sem conhecimento prévio de seus valores. Entretanto, a sequência definida pelo acesso através deste domínio não necessita seguir necessariamente nenhum tipo de ordenação (como momento de armazenamento ou índices), devendo-se buscar a otimização das formas de acesso suportadas internamente.

4.6.6- Domínio dos Objetos

O Domínio dos Objetos centraliza a comunicação com todos os demais domínios UNICOSMOS. Além desta função de comunicação, as informações sobre as estruturas de atributos de cada objeto-unicosmos e sobre a relação entre objetos-unicosmos do tipo conjunto e seus elementos também estão armazenadas neste Domínio.

A comunicação entre o Domínio de Objeto e o Domínio de Nomes (e, consequentemente, com os usuários do Sistema) é estabelecida através de uma lista de sinônimos associada ao Identificador Interno do objeto-unicosmos (ID). Assim, se for necessário por exemplo obter qual a classe (simples, conjunto, relação) correspondente a um objeto-unicosmos cujo ID é conhecido, esta informação é buscada no Domínio de Nomes a partir de um dos sinônimos desta lista.

A comunicação com o Domínio de Tríades é estabelecida através de uma lista de Identificadores Internos IR associada ao objeto-unicosmos. Esta lista contém cada entrada IR para o Domínio de Tríades definida para objetos-unicosmos que fazem parte de pelo menos uma tríade (como descrito no Item 4.6.2).

Existem duas listas associadas a cada Identificador ID para

representar informações sobre conjuntos. Uma delas mantém as informações sobre quais objetos-unicosmos conjuntos contém o objeto-unicosmos em questão. A outra descreve, para os objetos-unicosmos conjuntos, quais são seus elementos. Nos dois casos os objeto-unicosmos são representados através de seus Identificadores ID.

Outra informação diz respeito às estruturas de atributos de cada objeto-unicosmos. Esta estrutura foi definida sobre a antiga lista de caracteres associada a cada objeto-unicosmos definida no CORAS [WU/83]. Por questão de flexibilidade, foi definido que um objeto-unicosmos pode ter mais de uma estrutura de atributos associada, pois pode haver mais de um tipo de informação associada a cada objeto-unicosmos. Por exemplo, um tipo de informação poderia corresponder às propriedades do objeto-unicosmos, outro às regras associadas ao objeto-unicosmos (formção, consistência) ou outro tipo de informação semântica. Desta forma, uma lista de caracteres contém as diversas estruturas de atributos definidas para um mesmo objeto-unicosmos, sendo que cada estrutura é identificada por um rótulo. O controle destes rótulos (qual o tipo de informação associada a cada estrutura de atributos e como esta informação é utilizada) é exercido pela aplicação suportada pelo UNICOSMOS.

Para cada estrutura está associado um identificador IA correspondente à comunicação com o Domínio de Acesso, que por sua vez permite a comunicação com o domínio de ocorrências (item 4.6.5), permitindo assim agilizar buscas exaustivas (não associativas). Por sua vez, a comunicação com o domínio de valores é definida por indicadores associados a cada um dos atributos de cada estrutura; Ele será um IC (item 4.6.3).

Uma visão geral dos Domínios UNICOSMOS é apresentada na figura 4.2.

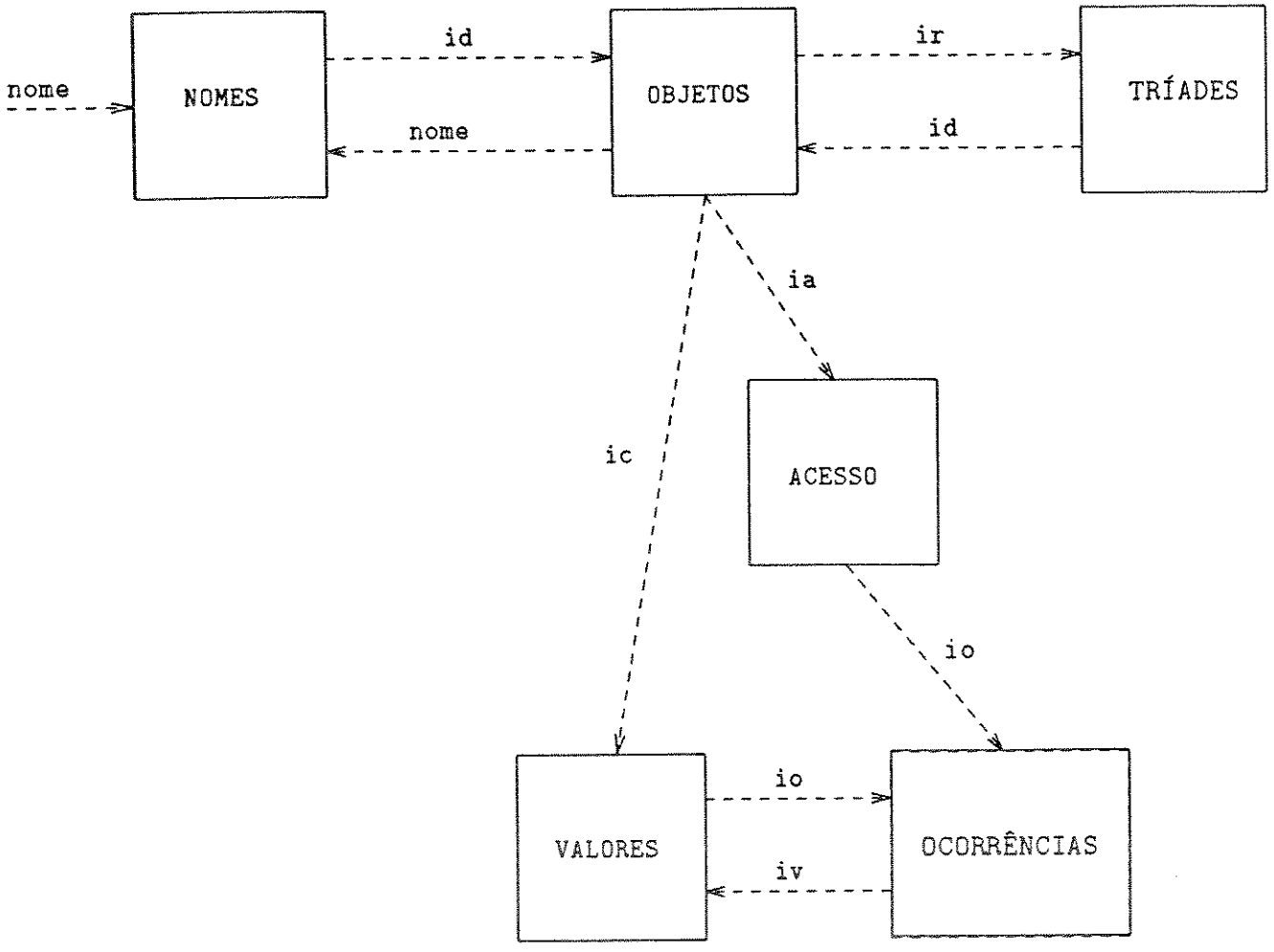


FIGURA 4.2 DOMÍNIOS UNICOSMOS

4.7- Visão Funcional

O UNICOSMOS provê funções para a definição e manipulação de suas primitivas. O usuário UNICOSMOS não necessita ter acesso à estrutura interna de armazenamento, à definição do protocolo de comunicação entre a memória principal e secundária ou à definição do formato de arquivos em disco. Para atingir estes objetivos, são definidos internamente três níveis de função UNICOSMOS:

- Nível Terciário: contém rotinas disponíveis ao usuário UNICOSMOS (a aplicação), englobando as rotinas de definição de base de dados ("sistemas de arquivos") e rotinas de definição e manipulação dos elementos primitivos;

- Nível Secundário: é o nível de visão lógica dos dados, onde são definidas e manipuladas as estruturas internas de dados referentes aos diversos domínios;

- Nível Primário: contém as rotinas que manipulam o nível físico. Estas rotinas definem a forma de acesso aos dados em disco, controlando também o sistema de paginação (manutenção em memória principal apenas de dados relevantes, mantendo os demais em memória secundária).

4.7.1- Nível Primário

As informações referentes a comunicação entre o dispositivo de armazenamento e a memória principal podem ser divididas em duas categorias:

- Temporárias: que só têm relevância enquanto o Sistema está ativo;

- Permanentes: que devem perdurar entre ativações distintas do Sistema.

As informações permanentes incluem a descrição da alocação do espaço em disco referente aos arquivos armazenados de cada Sistema de Arquivos. Como estas informações devem ser mantidas entre sessões de trabalho distintas, constituem um arquivo de dados de apoio ao Sistema de Arquivos de dados operacionais (dados referentes aos domínios internos do UNICOSMOS que também devem ser armazenado em disco.

As informações temporárias dizem respeito ao sistema interno de paginação, ou seja, constituem uma descrição sobre quais registros físicos estão ocupando a memória principal e se esse registros foram ou não modificados enquanto permaneceram na memória.

Existem basicamente quatro grupos de rotinas do nível primário:

- Rotinas de Acesso a Disco: definem as funções de criação e utilização (abrir/fechar) dos arquivos de dados em disco. Definem o acesso direto a registros tanto para escrita como para leitura. Constituem a interface entre o Sistema Operacional e o UNICOSMOS;

- Rotinas de Gerência das informações: manipulam as informações temporárias e permanentes do nível primário. Através destas rotinas, tornam-se transparentes ao Sistema de Paginação as estruturas internas destas informações;

- Rotinas do Sistema de Paginação: acessam as rotinas do Sistema de gerência de informações e as rotinas de Acesso a Disco (escrita e leitura, acesso direto), implementando os algorítmos de paginção para os arquivos de dados;

- Rotinas de Interface com Nível Secundário: as rotinas do nível primário tratam os arquivos de dados do Sistema através de códigos internos. Como não é interessante que haja a utilização desse tipo de informção interna nos demais níveis, são definidas interfaces que as isolam. Estas rotinas englobam todas as funções relativas ao nível

Primário.

4.7.2 Nível Secundário

A cada um dos Domínios UNICOSMOS está associada uma estrutura de dados independente. Os arquivos de dados associados e estas estruturas são denominados Arquivos Internos. Desta forma, existe um arquivo Interno de Nomes, um Arquivo Interno de Objetos e assim por diante. As funções responsáveis pela manipulação de cada um destes Arquivos Internos constituem as rotinas associadas ao nível secundário das funções UNICOSMOS.

Em seu nível mais alto, estas rotinas constituem a interface para as funções do Nível Terciário, que é o responsável pela interconexão das informações distribuídas pelos arquivos internos.

Internamente, as funções desempenhadas por estas rotinas dependem da estrutura de dados e das opções de implementação associadas a cada Arquivo Interno. Por exemplo, se as informações em um arquivo interno estão armazenadas na forma de listas, então deverão ser providas funções para a manipulação de listas e seus elementos.

As rotinas deste nível utilizam rotinas do Nível Primário, de forma que alterações processadas no Nível Secundário (correspondente a uma visão lógica dos Arquivos Internos) têm reflexo sobre o arquivo físico, mantendo utilizadas informações tais como quantidade de páginas utilizadas pelo Arquivo Interno e o espaço disponível em cada página.

4.7.3- Nível Terciário

As rotinas deste nível constituem a interface externa do UNICOSMOS. As funções desempenhadas por estas rotinas são basicamente:

- tornar transparente ao usuário a utilização das funções de nível inferiores;

- realizar a comunicação entre os diversos Domínios UNICOSMOS;

- uniformizar o tratamento de códigos de erro.

O primeiro objetivo visa isolar o usuário UNICOSMOS da arquitetura interna do Sistema, de forma que para utilizá-lo não seja necessário conhecer, por exemplo, quais arquivos serão acessados durante uma operação.

A comunicação entre domínios é realizada através de chamadas às rotinas de interface do Nível Secundário, de forma que a troca de informações entre domínios ocorre neste nível.

O grupo de rotinas associado a um domínio apresenta, em seu nível mais alto, códigos de erro que são independentes dos códigos definidos para os demais domínios. A tradução destes códigos internos em erros do Sistema também é tarefa das rotinas deste nível.

A especificação das rotinas deste nível pode ser encontrada em [RICA/86].

4.8- Resumo

O UNICOSMOS constitui uma alternativa a outros Sistemas associativos que manipulam objetos-unicosmos e conjuntos, como é o caso de CORAS. Apesar do alto nível de abstração apresentado por tais sistemas, o desempenho em geral deixa a desejar. Particularmente em relação ao CORAS, pode-se citar:

acesso: o UNICOSMOS distingue o acesso a objetos-unicosmos do acesso de valores. No CORAS, um valor deveria ser definido como um objeto-unicosmos para permitir sua representação e consequentemente seria acessado da mesma forma que os objetos-unicosmos, através de *hashing*. No UNICOSMOS, valores independem de *hashing*.

representação interna de valores: a vantagem com relação à implementação do CORAS se encontra no fato de que não há limitação lógica para a dimensão da lista que contém a representação de um valor (no CORAS, sendo cada valor representado como um objeto-unicosmos havia limitação correspondente à dimensão de um nome, que constitui a entrada para a tabela *Hash*) sendo que em comum se manteve a representação na forma de caracteres. A restrição permanece para nomes de objetos-unicosmos;

velocidade: o fato de não se utilizar tríades para representar associações do tipo atributo (como era necessário em CORAS para representar a associação entre um objeto-instância e um objeto-valor através de um objeto-atributo) alivia o volume das informações relacionadas a este domínio. Associações armazenadas no Domínio de Tríades representam apenas associações do tipo relacionamento entre classes;

interpretação: ao pré-especificar algumas abstrações (o conceito de atributos representa uma agregação dos conjuntos correspondentes aos conjuntos de valores possíveis) permite-se agilizar o tratamento computacional. No entanto, é exigido que haja um conhecimento sobre os dados armazenados, o que não era exigido no CORAS, onde valores, conjuntos e atributos poderiam ser acessados da mesma forma (eram todos acessados através do nome dos objetos-unicosmos correspondentes). No UNICOSMOS, o acesso a um valor é definido através do acesso a um atributo que, por sua vez, é acessado através do conhecimento do objeto-unicosmos a que pertence.

Os tipos de aplicação que podem utilizar o UNICOSMOS são diversos. A flexibilidade de estruturas de armazenamento e a possibilidade de representação de mais de um tipo de informação por objeto-unicosmos permitem representar uma quantidade de informações semânticas muito maior que a obitida em Sistemas de armazenamento de dados convencionais.

Dentre as aplicações possíveis para o UNICOSMOS, uma é sua

utilização como núcleo de Sistema de Gerência de Banco de Dados (SGBD) não convencionais. A implementação de um SGBD utilizando internamente um Sistemas voltado para a manipulação de objetos-unicosmos, constitui uma alternativa em relação a outras propostas que em geral partem de um SGBD baseado em modelos de dados convencionais, utilizados em aplicações comerciais, com camadas externas que visam adequá-las a estas aplicações. No entanto, o UNICOSMOS ainda apresentam algumas limitações no que se refere a Sistemas Orientados por Objetos. Estas limitações serão apresentadas no próximo capítulo, onde também serão apresentadas as soluções de tais limitações. Ao mesmo tempo também são apresentados aspectos de implementação da nova versão.

CAPÍTULO 5

Extensão do UNICOSMOS para Suporte a Orientação por Objeto

5.1- Introdução

Como visto no capítulo 4, a idéia central que deu origem ao UNICOSMOS, foi a de desenvolver um núcleo que fosse capaz de oferecer suporte aos gerenciadores de Sistemas de Banco de Dados nas funções de mais baixo nível, tais como armazenar ou consultar atributos de objetos, armazenar ou consultar tipos de objeto, etc. Posteriormente surgiu a necessidade de acrescentar ao UNICOSMOS, algumas facilidades de SOO's, que até então não eram suportadas pelo mesmo.

Esta necessidade deveu-se ao fato de que, nas aplicações de engenharia, os dados a serem manipulados são geralmente complexos, diferentemente das aplicações ditas convencionais. Na engenharia, os dados usualmente são compostos por outros dados. Por exemplo, numa indústria automobilística, um carro pode ser um dado que é composto de carroceria, rodas, motor, etc, que por sua vez também são dados. Isso faz com que a manipulação da informação seja mais complexa. Essa complexidade do dado fez com que se tivesse uma aproximação muito grande entre aplicações de engenharia e tratamentos e conceituações oriundas dos SOO's. Pode-se citar como exemplos o mecanismo de herança e conceito de abstração (Agregação, Generalização, Associação ou Agrupamento, etc). Em uma primeira aproximação, esses tratamentos e conceituações poderiam ser implementados a nível de gerenciador, já que, os mesmos não eram suportados pela versão anterior do UNICOSMOS. Esta hipótese exigiria um esforço muito grande de implementação e acarretaria problemas de desempenho, já que os gerenciadores atuam num nível de abstração mais alto do que o UNICOSMOS.

Optou-se assim por estender o UNICOSMOS para que este tivesse condições de oferecer as facilidades oriundas dos SOO's aos gerenciadores de Sistemas de Banco de Dados. Assim, toda manipulação dessas facilidades seria suportada a um nível de abstração mais baixo.

Portanto, o objetivo básico da extensão do UNICOSMOS é facilitar os gerenciadores na definição dos seguintes conceitos:

* Mecanismo de Herança;

* Abstração (generalização, classificação, agregação, etc);

* Tipos Abstratos de Dados.

A nova versão do UNICOSMOS continua sendo desenvolvida em liguagem C sobre sistema operacional UNIX, numa estação de trabalho SUN-SPARC 1.

5.2- Limitações da Versão Anterior do UNICOSMOS

A seguir, são descritas algumas limitações do UNICOSMOS. Estas fazem com que ele não suportasse algumas características de SOO's, ou pelo menos não fornecesse suporte aos gerenciadores de Sistemas de Banco de Dados, a fim de que estes tivessem mais facilidade de implementar tais características.

5.2.1- Limitações no Tratamento de Domínio de Atributo de Objeto

No que diz respeito ao domínio de atributos de um objeto, o UNICOSMOS não proporciona nenhum tratamento sobre o tipo de domínio de um atributo de um determinado objeto. Em outras palavras, o domínio de um atributo de um objeto é definido somente por uma cadeia de caracteres (figura 5.1). A partir daí não é feito nenhum tipo de consistência ou manipulação sobre o domínio definido, tal como, por exemplo:

* Teste de validade de domínio;
* Verificação se o domínio existe;
* Modificação do domínio de um atributo de um objeto.

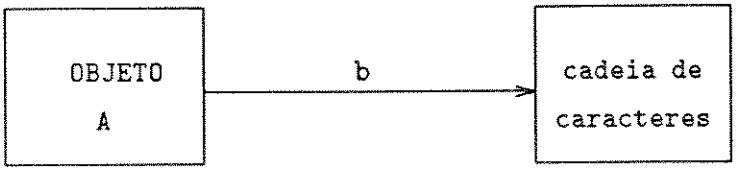


FIGURA 5.1 LIMITAÇÃO NO DOMÍNIO DE ATRIBUTOS

Qualquer tipo de consistência ou manipulação no domínio teria que ser feita a nível de usuário do UNICOSMOS, deixando assim funções de baixo nível sobre a responsbilidade do mesmo. Consequentemente, isto trazia uma sobrecarga para o usuário, pois este iria se preocupar com funções de baixo nível. Inclui-se também o problema de desempenho, pois as funções de consistência ou manipulação teriam que ser implementadas a nível de usuário UNICOSMOS.

5.2.2- Inexistência de Definição de TAD

No UNICOSMOS não existe definição de **Tipos Abstratos de Dados (TAD's)**. Esta é uma grande limitação do UNICOSMOS, pelo fato de não ser possível fazer qualquer tipo de tratamento de objetos com funções específicas associadas aos mesmos. Outra limitação é que não existe nenhuma diferenciação no tratamento de representação de valores de atributos, pois toda representação de valores associada a atributo é uma cadeia de caracteres. Sendo assim, qualquer tratamento teria que ser feito a nível da aplicação que utilizasse o UNICOSMOS.

5.2.3- Ausência de Hierarquia

O UNICOSMOS não suporta a hierarquia de objetos, e consequentemente o mecanismo de herança. Esta deficiência impede que o UNICOSMOS dê suporte para os gerenciadores implementar alguns conceitos de abstração (Generalização, Agregação, Associação, etc) de uma forma mais eficiente e direta. Sendo assim todos esses conceitos têm que ser implementados, totalmente, a nível dos gerenciadores. Isto traz, consequentemente, alguns problemas tais como:

* Menor desempenho e eficiência do sistema;
* Maior esforço de programação por parte do usuário.

Isto porque , os gerenciadores estariam num nível de abstração bem mais elevado do que o UNICOSMOS.

5.3- Suporte a Orientação por Objeto no UNICOSMOS

Na versão corrente do UNICOSMOS, foi adotada uma nova filosofia, que teve como objetivo atenuar as limitações descritas na seção anterior. Esta nova filosofia faz com que o UNICOSMOS se aproxime mais ainda de um SOO, absorvendo algumas características essenciais destes Sistemas. A seguir apresentam-se estas características.

5.3.1- Objeto Domínio

Objeto Domínio foi a solução encontrada para resolver o problema no tratamento do domínio de atributo de um objeto discutido na seção 5.2.1.

No UNICOSMOS, um objeto-unicosmos é definido através de atributos que o levam aos seus universos. Cada universo é chamado de Objeto Domínio. Em outras palavras, um Objeto Domínio é um objeto que define o universo de atuação de um outro objeto através de um atributo seu. Considere como exemplo a figura 5.2. O objeto A é definido pelos atributos "b" e "c", que o levam aos objetos domínios B e C, respectivamente. Os objetos domínios B e C são os universos do objeto A. Diz-se que os objetos B e C são pais do objeto A, ou que este é filho dos objetos B e C, ou ainda que os atributos "b" e "c" são do tipo objetos B e C, respectivamente.

Vale obsevar que existe todo um tratamento para manipulação de Objeto Domínio. Por exemplo, antes da criação dos atributos "b" e "c" do objeto A, os objetos B e C já devem ter sido criados pelo usuário, pois na definição do objeto A o UNICOSMOS verifica se os tipos dos atributos "b" e "c", que são os objetos B e C, existem.

5.3.2- Tipo Abstrato de Dado (TAD)

Presentemente, o UNICOSMOS foi estendido de forma a possuir a capacidade de definir alguns tipos básicos como tipos abstratos de

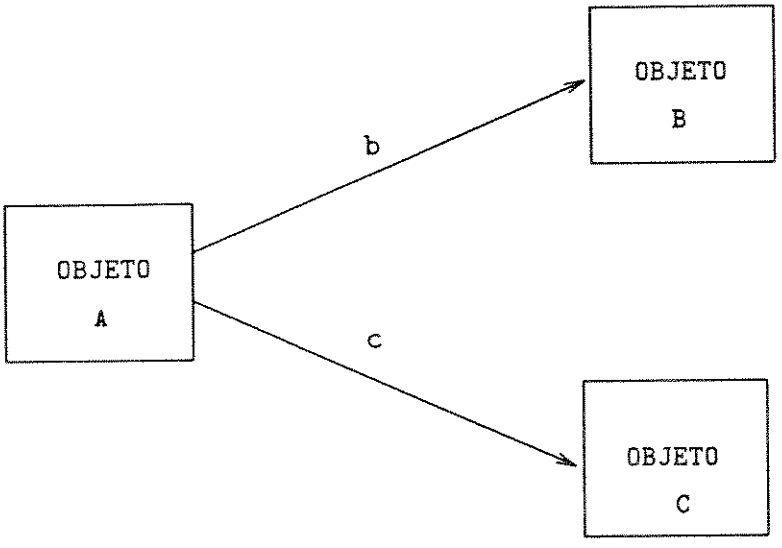


FIGURA 5.2 OBJETO DOMÍNIO

dados. Esses TAD's são objetos que têm um grupo de operações associado a cada um deles. Somente estas operações podem manipulá-los. Todo mecanismo de instanciação de um objeto é feito através dos TAD's que são domínios deste mesmo objeto.

Quando o UNICOSMOS é ativado, ele próprio se encarrega de criar, automaticamente, os TAD's definidos pelo Sistema. Eles são os primeiros objetos a serem criados. Ao mesmo tempo que os TAD's são criados, o Sistema monta uma estrutura dinâmica mantida em memória onde estão todas características de cada TAD com suas respectivas funções de manipulação.

Vale observar que esta estrutura montada pelo sistema foi elaborada de uma forma flexível, o que permite ao usuário do UNICOSMOS acrescentar outros TAD's que sejam de seu interesse.

Existem três TAD's que são definidos pelo UNICOSMOS, que são:

* String;
* Inteiro;
* Tempo.

5.3.2.1- TAD String

O **TAD String** é um tipo básico que assume qualquer cadeia de caracteres na faixa de 040 a 0176 (octal). Associadas ao tipo String, existem algumas operações que tratam da manipulação dos dados deste tipo, e incluem o seguinte: valida_string, string_maior_ou_igual, string_menor_ou_igual, string_maior, string_menor, string_igual e string_não_igual.

5.3.2.2- TAD Inteiro

O **TAD Inteiro** é um tipo básico que assume qualquer cadeia de caracteres que represente um inteiro, ou melhor, qualquer cadeia de caracteres que esteja na faixa de 060 a 071 (octal). Existem algumas

funções associadas ao tipo Inteiro, que tratam da manipulação dos dados Inteiros, são elas: valida_inteiro, inteiro_maior_ou_igual, inteiro_menor_ou_igual, inteiro_maior, inteiro_menor, inteiro_igual e inteiro_não_igual.

5.3.2.3- TAD Tempo

Já o TAD Tempo é um tipo básico que tem uma das seguintes representações (exitem várias formas de tratar dados temporais, uma delas está apresentada em [SHOS/86]) :

1- Qualquer dado que tenha uma data com formato X/Y/Z, X - Y - Z, ou X . Y . Z onde:

X - é o dia do mês;
Y - é o mês do ano;
Z - é o ano.

X assume valores na faixa de 1 a 31 dependendo do mês e do ano (o mês é de 28, 29, 30 ou 31 dias dependendo do ano em questão);
Y assume valores na faixa de 1 a 12 meses;
Z assume valores na faixa de 00 a 99 anos.

2- O formato do ítem 1 acrescido do operador (+) ou (-) e o campo Rd, ou seja, "formato do ítem 1 (+) ou (-) Rd", onde: "R" é um número inteiro e "d" indica o campo do intervalo de dias. "Rd" indica um intervalo de dias que deve ser acrescido ou diminuído (+, -) do formato data.

3- O formato do item 1 acrescido do operador (+) ou (-) e do campo Ts. Ou seja, "formato do ítem 1 (+) ou (-) Ts", onde: "T" é um número inteiro e "s" indica o campo do intervalo de semana. "Ts" indica um intervalo de semanas que deve ser acrescido ou diminuído (+ -) do formato data.

4- O formato do ítem 1 acrescido dos formatos do ítem B e do ítem

C, ou seja, "formato do ítem A (+) ou (-) Rd (+) ou (-) Ts.

Existem algumas funções associadas ao tipo Tempo, que tratam da manipulação dos dados do tipo Tempo. São elas: valida_tempo, tempo_maior_ou_igual, tempo_menor_ou_igual, tempo_maior, tempo_menor, tempo_igual e tempo_não_igual.

5.3.3- Mecanismo de Herança

Com as novas filosofias de Objeto Domínio e TAD descritas nas sub-seções anteriores, é possível montar uma estrutura de hierarquização de objeto. A partir do momento em que objetos são definidos em função de outros objetos, através de seus atributos (conforme mostra o exemplo da figura 5.3), é quase imediata a estruturação de uma hierarquia de objetos.

A partir da figura 5.3 pode ser construída uma estrutura hierárquica do objeto A, em função dos objetos B e C, como mostra a figura 5.4.

De acordo com a figura 5.4, o objeto A poderá herdar os atributos dos objetos C e B, ou seja, "d", "e", "f", e "g" , dependendo do mecanismo de herança que tenha sido definido. Este mecanismo tem os seguintes passos:

1- definir se os atributos de um objeto quer herdar ou não os atributos dos seus objetos domínios;

2- definir se os atributos de um objeto podem ser herdados pelos objetos que o utilize como objeto domínio.

Para que o objeto A tenha a estrutura hierárquica mostrada na figura 5.4, é necessário, em primeiro lugar, que o objeto A herde os atributos dos seus objetos domínios (B e C), ou seja, os atributos "d" e "c" do objeto B, e "f" e "g" do objeto C. Em segundo lugar, os objetos domínios (B e C) tem que permitir que seus atributos sejam

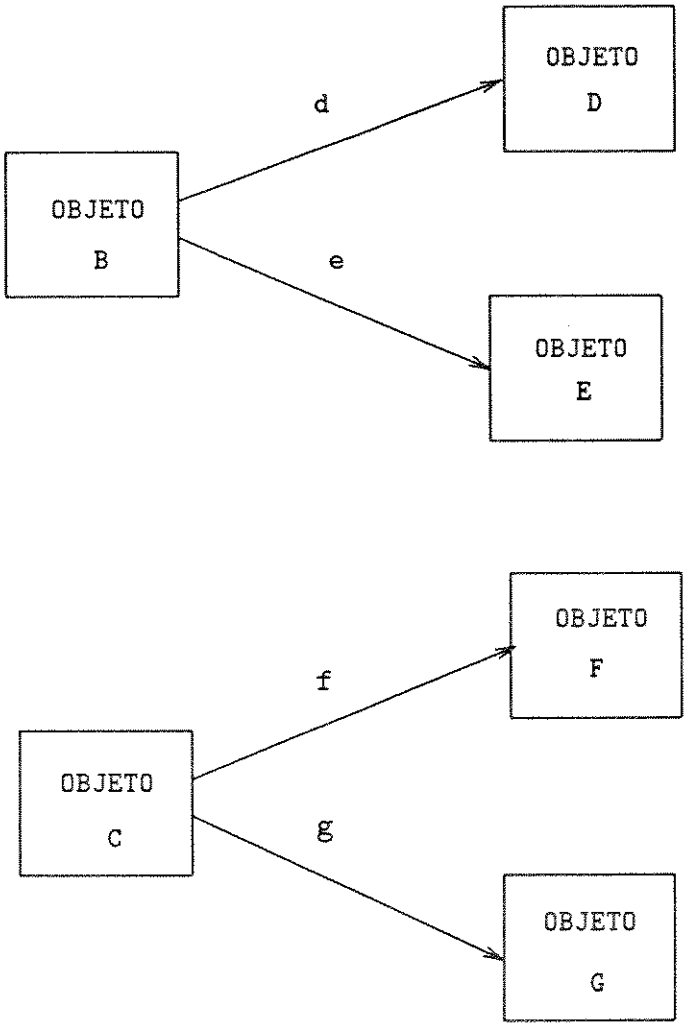


FIGURA 5.3 DEFINIÇÃO DOS OBJETOS "A" e "B"

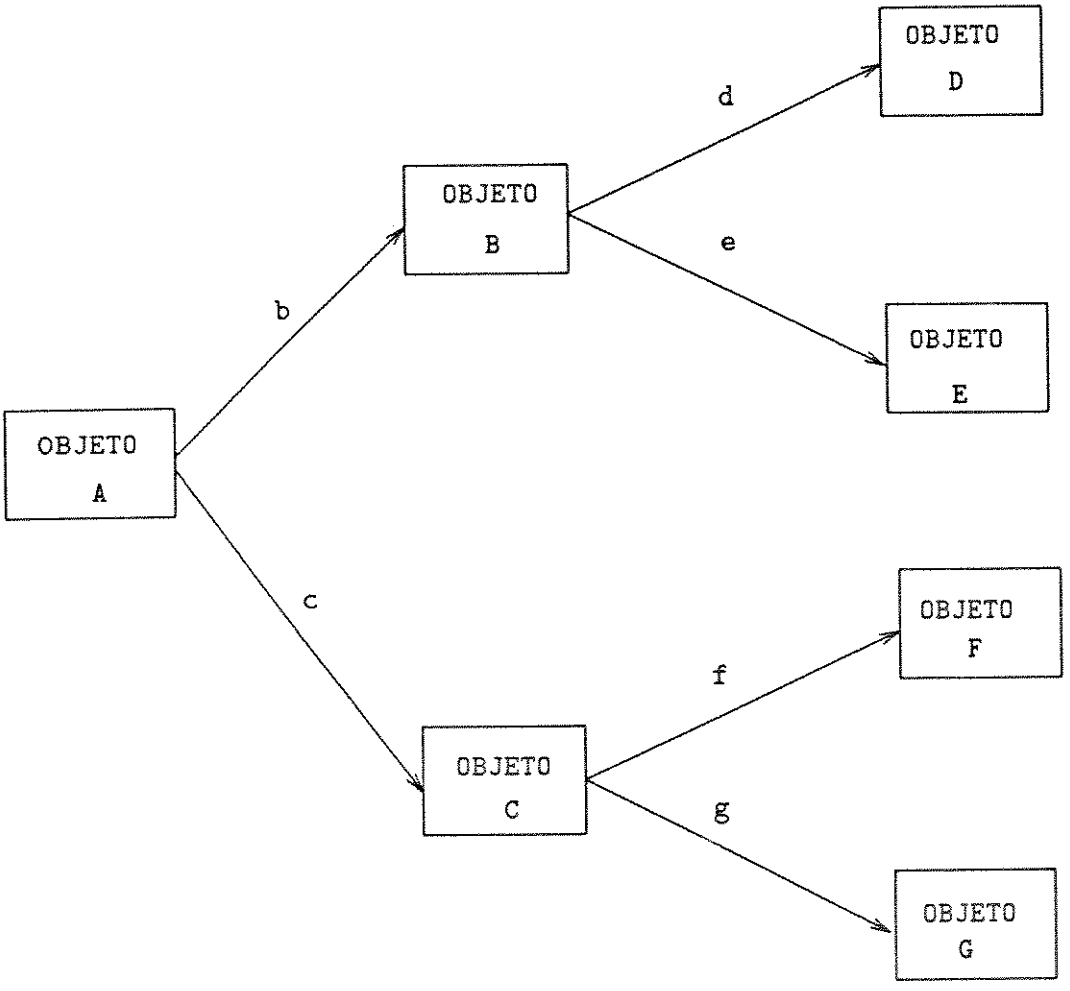


FIGURA 5.4 HIERARQUIA DO OBJETO "A"

herdados pelo objeto A.

Na nova versão do UNICOSMOS, o mecanismo de instanciação é mais complexo do que aquele da versão anterior. Antes, a instanciação de um objeto dava-se somente a um nível, através dos atributos diretos do objeto que se desejava instanciar, pois não havia hierarquia de objetos. No presente momento, a instanciação de um objeto é feita através dos objetos folha da árvore de hierarquia do objeto a ser instanciado. No caso da figura 5.4, a instanciação do objeto A seria feita através dos atributos herdáveis "d", "c", "f" e "g" dos objetos domínios B e C.

Podem ser citados outros Sistemas que permitem o mecanismo de herança tais como o O2 [DEUX/90] e o COMS [LACR/89].

5.4- Aspectos de Implementação

Para que o UNICOSMOS absorvesse esta nova abordagem foram necessárias algumas modificações tanto nas estruturas de dados já existentes, quanto nas funções que as manipulam. Em vários casos foi necessário acrescentar estruturas de dados e funções para manipulá-las. A seguir, serão descritas as modificações e acréscimos que se fizeram necessários e que estão associados a cada característica da abordagem proposta. A absorção desta abordagem causou maior impacto no arquivo Interno de Objetos, descrito a seguir.

5.4.1- Arquivo Interno de Objetos

O Arquivo Interno de Objetos contém as informações sobre a estrutura dos objetos e apontadores às suas instâncias.

Cada objeto está representado no Arquivo Interno de objetos por um cabeçalho, que por sua vez permite o acesso às demais informações armazenadas no arquivo. A posição deste cabeçalho no arquivo - definida não por seu endereço físico mas através de apontadores à página em que se encontra e à celula que ocupa dentro

desta página - constitui a implementação do ID (Identificador Interno) definido na seção 4.6.1 e que é referenciado pelos outros arquivos internos do Sistema.

As informações sobre cada objeto são armazenadas em células que, ligadas, constituem as listas internas do Arquivo Interno de Objetos (figura 5.5). Essas listas internas são quatro, a saber:

1- Lista de nomes: contém todos os nomes definidos para o objeto - o nome inicial e os sinônimos definidos posteriormente;

2- Lista de atributos: contém as máscaras de atributos definidas pela aplicação, sendo que cada atributo tem associado um apontador ao Arquivo Interno de Valores. Mantém também a informação sobre a quantidade de ocorrências associadas a cada máscara de atributos. Sua estrutura interna é a seguinte:

   2.1- Apontador para o sub-rótulo, campo sub-rot, contendo:

      2.1.1- Apontador para o Arquivo Interno de Acesso, campo apont.ia;

      2.1.2- Quantidade de Ocorrências, campo qt.ocorr;

      2.1.3- Código interno do último atributo da lista de atributos, campo c.ul.atr.

   2.2- Quantidade de Atributos, campo qt.atrib;

   2.3- Apontador para o início da lista de atributos, campo apont.atr., contendo por atributo:

      2.3.1- Nome do Atributo, campo nom;

      2.3.2- Código Interno associado ao atributo, campo c.int;

      2.3.3- Identificador do Objeto Domínio associado ao Atributo, campo tip.;

      2.3.4- Apontador Interno ao Arquivo Interno de Valores (IC), campo in.va.;

      2.3.5- Indicador: Atributo é multivalorado ou não, campo mul.;

      2.3.6- Código de querer hierarquia, campo q.h.;

      2.3.7- Código de permiitr hierarquia, campo p.h.;

   2.4- Apontador para próximo rótulo, campo ap.pr.cel.

3- Lista de filhos: armazena os ID (Identificador Interno) dos objetos filhos que pertencem objeto que contém esta lista.

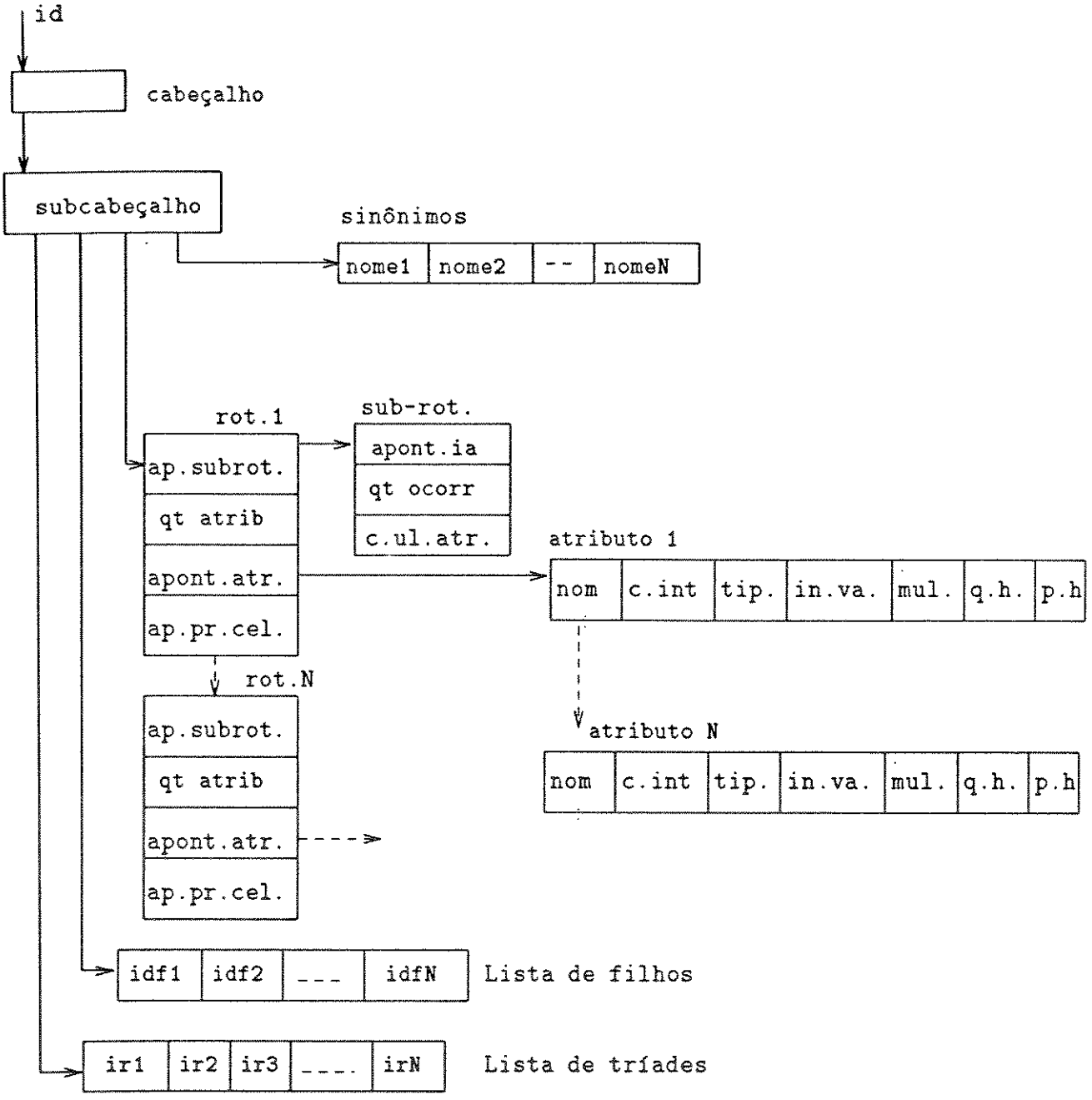


FIGURA 5.5 LISTAS INTERNAS DO ARQUIVO INTERNO DE OBJETOS

4- Lista de entrada ao arquivo de tríades: armazena apontadores semelhantes aos ID, que são definidos para o Arquivo Interno de Tríades (IR), definindo a posição naquele arquivo onde estão armazenadas as informações sobre as tríades nas quais o objeto toma parte.

O cabeçalho contém informações sobre a quantidade de elementos em cada uma das listas acima descritas. A flexibilidade na definição do tamanho de uma lista é garantida através de uma estrutura intermediária entre o cabeçalho e as listas internas - os subcabeçalhos - que além dos apontadores ao início de cada lista na página permitem armazenar apontadores a uma possível continuação da lista em outra página (figura 5.6)

5.4.2- Modificação e Acréscimo para Objeto Domínio

Para absorção do conceito Objeto Domínio foram necessárias as seguites modificações e acréscimos:

* Modificações em funções existentes que manipulam com atributo de objetos;

* Acréscimo de funções que manipulam com filhos de um objeto.

5.4.2.1- Modificações em Funções que Manipulam Atributos

Para definir um atributo de um objeto-unicosmos é necessário definir seu Objeto Domínio. O mecanismo básico do Objeto Domínio é a colocação do identificador de um objeto (Objeto Domínio) no campo que determina o tipo do atributo do objeto que o está utilizando como domínio. Ou seja, colocar o ID do Objeto Domínio no campo "tip" da lista de atributos do objeto (figura 5.5). No mesmo momento é feita também a colocação do identificador deste objeto na lista de filhos do Objeto Domínio. Em outras palavras, considerando as figuras 5.2 e 5.5, o identificador do objeto A é colocado na lista de filhos dos objetos B e C, e o identificador do objeto A é colocado no campo "tip" da

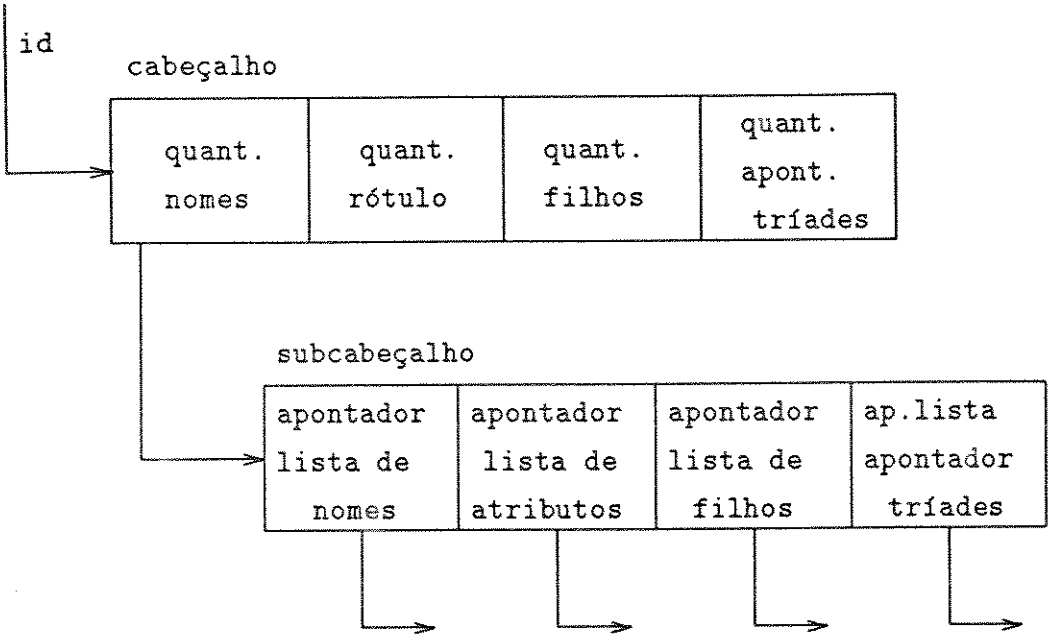


FIGURA 5.6 ESTRUTURA CABEÇALHO / SUBCABEÇALHO

lista de atributos dos objetos B e C. Antes do processo de colocação desses identificadores, tem que haver um teste para verificar se o Tipo do atributo em questão (Objeto Domínio) existe como um objeto na base de objetos. Caso não exista, o sistema não executa o processo em questão e retorna um código de erro para o usuário.

Existem algumas funções que estão associadas ao processo acima descrito, especificadas sucintamente, a seguir:

1- CRIAR ATRIBUTOS: esta função cria a lista de atributos de objeto. Ela continua fazendo as mesmas tarefas da versão anterior; e acrescenta a verificação da existência do Tipo de cada atributo a ser armazenado e o armazenamento do ID do Objeto Domínio de cada atributo no campo "tip" (figura 5.5) do mesmo atributo;

2- ALTERAR TIPO DE UM ATRIBUTO: essa função altera o tipo de um atributo de um objeto, ou seja, ela troca o Objeto Domínio de um atributo por um outro que o usuário deseja, desde que este último exista na base de objetos. A função tem que retirar o ID do objeto da lista de filhos (campo "idf" da figura 5.5) do Objeto Domínio que se deseja trocar, e colocar este ID na lista de filhos do novo Objeto Domínio;

3- REMOVER ATRIBUTOS DE OBJETO: remove atributos de um determinado objeto. Quando a função remove um atributo de um objeto, ela também remove o ID (campo "idf" da figura 5.5) do objeto em questão da lista de filhos do Objeto Domínio daquele atributo que se deseja remover.

5.4.2.2- Acréscimo de Funcões que Manipulam Filhos de Objetos

Existem funções que manipulam a lista de filhos de objetos, que são:

1- OBTER FILHOS DO OBJETO: essa função obtém os filhos de um determinado objeto, ou seja ele obtém os objetos que estão na lista de

filhos de um determinado objeto;

2- OBTER QUANTIDADE DE FILHOS DO OBJETO: essa função obtém a quantidade de filhos de um determinado objeto;

3- VERIFICAR SE UM OBJETO É FILHO DE OUTRO OBJETO: verifica se um determinado objeto é filho de outro objeto, ou seja, se um objeto pertence a lista de filhos de um outro objeto;

4- OBTER FILHOS COMUNS ENTRE DOIS OBJETOS: verifica se dois objetos têm filhos comuns. Se tiver, obtém os filhos comuns.

5.4.3- Módulo de Tipos Abstratos de Dados (TAD's)

Para absorver TAD's, foi desenvolvido um módulo separado, onde se encontram todas as estruturas de dados e funções necessárias para manipulação dos TAD's definidos na seção 5.3.2. A idéia básica para a implementação dos TAD's foi obter uma estrutura dinâmica e flexível, capaz de suportar acréscimo de futuros TAD's que venham fazer parte dos tipos básicos do UNICOSMOS, e promover facilidades e segurança na manipulação dos tipos de informações na Base de Dados. Existem três matrizes contendo informações associadas aos TAD's (nomes, operadores e apontadores para funções dos TAD's) a partir dos quais o próprio Sistema constrói uma estrutura dinâmica - lista de listas - que fica na memória principal, conforme a figura 5.7.

As funções dos TAD's são ativadas por funções do UNICOSMOS que estão em interface com o usuário UNICOSMOS. Essa foi uma alternativa adotada a fim de que o usuário UNICOSMOS não tivesse acesso direto as funções dos TAD's que são funcões de mais baixo nível.

As estruturas de dados bem como as funções que manipulam os TAD's são descritas a seguir.

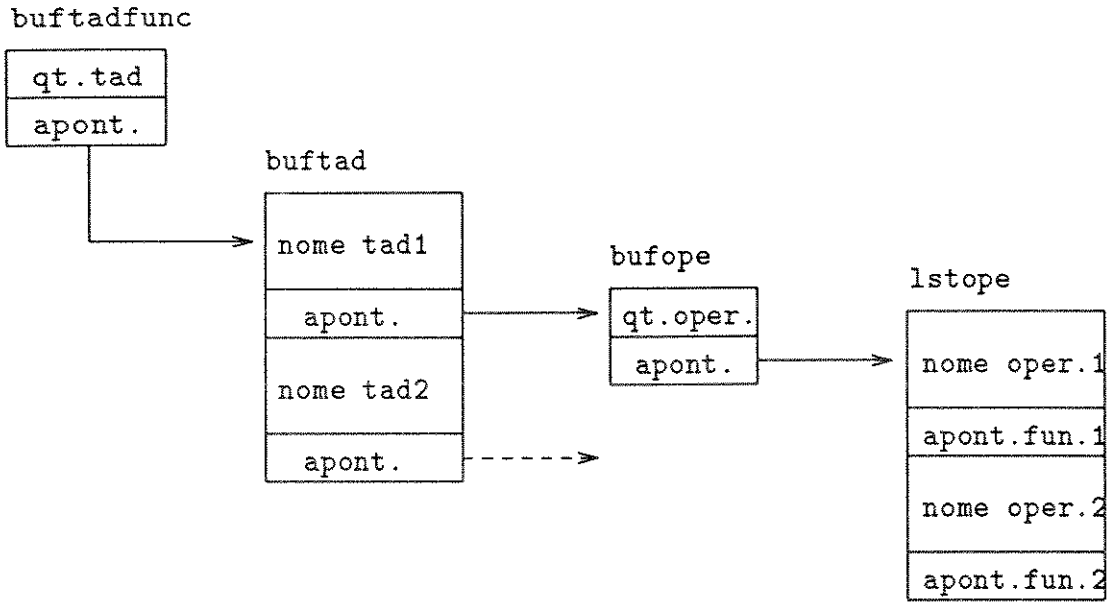


FIGURA 5.7 ESTRUTURA DINÂMICA PARA TAD's

5.4.3.1- Estruturas de Dados para TAD's

Existem três matrizes que contêm informações relacionadas com os TAD's, que são:

1- Matriz de apontadores para funções - é uma matriz composta de apontadores para as funções relacionadas com os TAD's. Cada linha da matriz é associada com funções de um determinado TAD. Esta matriz tem a seguinte forma:

*funcTAD11	*funcTAD12	*funcTAD13	*funcTAD1n
*funcTAD21	*funcTAD22	*funcTAD23	*funcTAD2n
*funcTAD31	*funcTAD32	*funcTAD33	*funcTAD3n
*funcTADm1	*funcTADm2	*funcTADm3	*funcTADmn

2- Matriz com nomes dos operadores relacionais - é uma matriz composta pelos operadores relacionais dos TAD's, onde cada linha desta matriz está associada com a linha da matriz das funções que manipulam os TAD's (matriz do ítem 1). A matriz tem a seguinte forma:

oprelTAD11	oprelTAD12	oprelTAD13	oprelTAD1n
oprelTAD21	oprelTAD22	oprelTAD23	oprelTAD2n
oprelTAD31	oprelTAD32	oprelTAD33	oprelTAD3n
oprelTADm1	oprelTADm2	oprelTADm3	oprelTADmn

3- Matriz com nomes dos TAD's - matriz que contém os nomes dos TAD's. Esta matriz tem a seguinte forma:

nomeTAD1	nomeTAD2	nomeTAD3	nomeTADm

As matrizes acima descritas, serão utilizadas pelo Sistema para a construção da estrutura dinâmica referida na seção 5.3.1. Essa estrutura é feita sobre a seguinte estrutura de dados (figura 5.7):

1- Buffer com os TAD's e as funções: esse buffer - denominado buf_tadfunc - é composto por dois campos que são:

1.1- Quantidade de TAD: campo que dá a quantidade de TAD existente no sistema;

1.2- Apontador para buffer de TAD: apontador para uma lista de buffer - denominado buffer_tad - que contém os dados de um TAD.

2- Buffer com os dados dos TAD: é um buffer - denominado buftad - composto por dois campos (para cada TAD):
   2.1- Nome do TAD: contém o nome de um TAD no sistema;
   2.2- Apontador para lista de operações dos TAD's: contém apontador para uma lista de buffer - denominado buf_ope - que contém os dados das operações de um TAD.

3- Buffer com dados das operações de um TAD: é um buffer denominado bufope - que é composto por dois campos que são:
   3.1- Quantidade de Operações: contém a quantidade de operações que um TAD possui;
   3.2- Apontador para lista de operações de TAD: contém um apontador para uma lista de buffer - denominado lst_ope que contém dados das operações de um TAD.

4- Buffer com dados das operações de um TAD: é um buffer denominado lstope que é composto por dois campos (por operação) que são:
   4.1- Nome da Operação: contém o nome de uma operação de um determinado TAD;
   4.2- Apontador para operação: contém um apontador para função associada com a operação do campo anterior.

Para concluir, existem ainda duas estruturas que estão associadas ao TAD Tempo, que são:

1- Estrutura Data: estrutura utilizada pelo sistema para receber os dados da data associada ao TAD Tempo que são fornecidos pelo usuário. Ela é composta por cinco campos:
   1.1- Validação: Campo utilizado pelo sistema para validação da data fornecida pelo usuário;
   1.2- Dia: Contém o dia da data fornecida pelo usuário;
   1.3- Mês: Contém o mês da data fornecida pelo usuário;
   1.4- Ano: Contém o ano da data fornecida pelo usuário;
   1.5- Dia-Ano: Contém o número de dias do ano relacionado com

a data fornecida pelo usuário. Este campo é preenchido pelo sistema, pois é ele que faz a transformação de datas caso o sistema constate a consistência da data.

2- Estrutura de intervalo: Estrutura utilizada pelo sistema para receber o intervalo associado ao TAD TEMPO que é fornecido pelo usuário. Ela é composta por tês campos que são:

2.1- Validação do intervalo: campo utilizado pelo sistema para validação do intervalo fornecido pelo usuário;

2.2- Numero de Dias: Contém o número de dias contido no intervalo.

2.3- Operação do Intervalo: Contém o tipo de operação a ser realizado sobre o intervalo. Pode ser mais (+) ou menos (-) .

Na seção seguinte, serão abordados os aspectos de implementação das funções que manipulam as estruturas acima descritas e as que manipulam os TAD's propriamente ditos.

5.4.3.2- Funções Associadas aos TAD's

Para efeito de explanação, as funções serão divididas em três grupos a saber:

*Funções Associadas aos Dados do Usuário:

São funções que fazem todo tipo de validação e verificação de dados fornecidos pelo usuário. Elas são utilizadas pelas funções dos TAD's.

*Funções dos TAD's:

São funções que estão atreladas diretamente aos TAD's. São elas que fazem qualquer manipulação direta com os tipos básicos do Sistema.

*Função de Inicialização da Estrutura de TAD's:

É uma função que inicializa a estrutura de dados para manipulação dos TAD's em memória.

1- Funções Associadas aos Dados do Usuário: São funções relacionadas com validação e verificação de dados do Tempo:

1.1- VERIFICAR A EXISTÊNCIA DE TAD: verificar se um determinado objeto é TAD do sistema;

1.2- VALIDAR VALOR COM TIPO DO TAD: valida um determinado valor com o domínio de um determinado TAD;

1.3- VALIDAR OPERAÇÃO COM FUNÇÕES DE TAD: valida se uma determinada operação corresponde a uma função de um TAD;

1.4- VALIDAR DATA: valida uma determinada data de um determinado valor TEMPO.

1.5- VALIDAR INTERVALO DE SEMANAS: valida um determinado intervalo de semana de um determinado valor TEMPO;

1.6- VALIDAR INTERVALO DE DIAS: valida um determinado intervalo de dias de um determinado valor TEMPO;

1.7- ACRESCENTAR DIAS A DATA: acrescenta um intervalo de dias a uma determinada data;

1.8- DECREMENTAR DIAS A DATA: decrementar um intervalo de dias a uma determinada data;

1.9- TRANSFORMAR TEMPO: transformar formato de TEMPO.

2- Funções dos TAD's

2.1- VALIDAR UM TAD STRING: valida se um determinado valor é do tipo STRING;

2.2- VERIFICAR SE STRING É MENOR OU IGUAL: verifica se uma string é menor ou igual a outra string;

2.3- VERIFICAR SE STRING É MAIOR OU IGUAL: verifica se uma string é maior ou igual a outra string;

2.4- VERIFICAR SE STRING É IGUAL: verifica se uma string é igual a outra string;

2.5- VERIFICAR SE STRING NÃO É IGUAL: verifica se uma string não é igual a outra string;

2.6- VERIFICAR SE STRING É MAIOR: verificar se uma string é maior que outra string;

2.7- VERIFICAR SE STRING É MENOR: verifica se uma string é menor que outra string;

2.8- VALIDAR UM TAD INTEIRO: valida se um determinado valor é do tipo INTEIRO.

2.9- VERIFICAR SE UM INTEIRO É MENOR OU IGUAL: verifica se um inteiro é menor ou igual a outro inteiro;

2.10- VERIFICAR SE UM INTEIRO É MAIOR OU IGUAL: verifica se um inteiro é maior ou igual a outro inteiro;

2.11- VERIFICAR SE UM INTEIRO É IGUAL: verifica se um inteiro é igual a outro inteiro;

2.12- VERIFICAR SE UM INTEIRO É NÃO IGUAL: verifica se um inteiro não é igual a outro inteiro;

2.13- VERIFICAR SE UM INTEIRO É MAIOR: verifica se um inteiro é maior que outro inteiro;

2.14- VERIFICAR SE UM INTEIRO É MENOR: verifica se um inteiro é menor que outro inteiro;

2.15- VALIDAR UM TAD TEMPO: valida se um determinado valor é do tipo TEMPO;

2.16- VERIFICAR SE TEMPO É MENOR OU IGUAL: verifica se um determinado tempo é menor ou igual a outro tempo;

2.17- VERIFICAR SE TEMPO É MAIOR OU IGUAL: verifica se um determinado tempo é menor ou igual a outro tempo;

2.18- VERIFICAR SE TEMPO É IGUAL: verifica se um determinado tempo é igual a outro tempo;

2.19- VERIFICAR SE TEMPO NÃO É IGUAL: verifica se um determinado tempo não é igual a outro tempo;

2.20- VERIFICAR SE TEMPO É MAIOR: verifica se um determinado tempo é maior que outro tempo;

2.21- VERIFICAR SE TEMPO É MENOR: verifica se um determinado tempo é menor que outro tempo.

3- Função de Inicialização da Estrutura de TAD's

3.1- INICIAR ESTRUTURA DE TAD's: inicia a estrutura de dados dos tipos abstratos de dados existentes no sistema com suas respectivas funções de manipulação.

5.4.4- Aspectos de Hierarquia

A hierarquia de objetos, como já visto na seção 5.3.3, pode ser vista como uma consequência dos conceitos de Objeto Domínio e TAD. O fato de um objeto-unicosmos ser definido por atributos que levam a outros objetos-unicosmos (objeto domínio) já caracteriza uma representação de hierarquia. Existem estruturas e funções relacionadas com os aspectos de hierarquia e que estão associados aos atributos, instâncias ou valores dos atributos dos objetos-unicosmos. Com relação as estruturas, a idéia básica foi permitir a definição de uma representação hierárquica a partir das quais se pode estruturar, obter e fornecer informações complexas dos objetos-unicosmos. Já as funções, permitem manipulações com as informações estruturadas hierarquicamente. Nesta seção serão detalhados a caracterização dos objetos-unicosmos, bem como o mecanismo de instanciação dessa hierarquização. A seguir descrevem-se as modificações nas estruturas de dados existentes e posteriormente descrevem-se as funções necessárias para manipulá-las.

5.4.4.1- Modificações das Estruturas de Dados Existentes

Para absorção do mecanismo de herança foi necessário fazer modificações nas estruturas de dados da versão anterior do UNICOSMOS. Essas modificações foram divididas em três partes:

* Estrutura de dados relacionada com atributo;
* Estruturas de dados relacionadas com instâncias;
* Estruturas de dados de atualização de valores.

1- ESTRUTURA DE DADOS RELACIONADA COM ATRIBUTO: Essa estrutura de dados está associada com os atributos de um objeto e serve como interface entre o usuário e a estrutura do Arquivo Interno de Objetos da figura 5.5. Ela recebe todos os dados do usuário relativo aos atributos de um determinado objeto. A estrutura tem os seguintes campos:

1.1- Nome do atributo: Esse campo é um apontador para um "char"

que define o nome do atributo de um objeto.

1.2- Identificador do atributo: é um número interno que identifica um atributo de um objeto.

1.3- Tipo do atributo: é um apontador para um "char" que define o tipo do atributo, ou melhor, define o nome do objeto domínio do atributo de um objeto.

1.4- Multivaloração: Define se o atributo é multivalorado ou não.

1.5- Código de permitir herança: Código que determina se um objeto permite que o atributo criado nesse mesmo objeto pode ser herdado ou não por algum outro objeto. Se o código for 1 o objeto permite que o atributo seja herdado, se o código for zero o objeto não permite.

1.6- Código de querer herança: Esse código determina se um objeto quer que um determinado atributo seu herde ou não os atributos do objeto domínio. O código 1 significa que o objeto quer herdar e o zero o contrário.

2- ESTRUTURAS DE DADOS DE INSTANCIAÇÃO: Existem algumas estruturas de dados que servem de interface entre o usuário e o UNICOSMOS, e é através delas que o usuário de aplicação executa o processo de instanciação dos objetos. A seguir descrevem-se estas estruturas com seus respectivos campos:

2.1- Buffer de instâncias: Estrutura que recebe do usuário de aplicação os valores que estarão associados a uma instância. Ela é composta dos seguintes campos:

2.1.1- Quantidade de atributo: quantidade de atributo do objeto que vai ser instanciado.

2.1.2- Identificador da instância: identificador da instância de um objeto.

2.1.3- Nome de objeto: é um apontador para um "char" que define o nome do objeto a ser instanciado.

2.1.4- Dados da instância: um apontador para uma estrutura (Estins).

2.2- Estins: estrutura que contém os dados de uma instância, associadas a um determinado atributo de um objeto que está

sendo instanciado. Ela é composta dos seguintes campos:

2.2.1- Nome do atributo: apontador para um "char" que contém o nome de um atributo do objeto a ser instanciado.

2.2.2- Identificador do atributo: número inteiro que identifica o atributo internamente.

2.2.3- União de valores: Um "union" que pode ser um dos seguintes campos:

2.2.3.1- Um apontador para um "char" representando um valor para um atributo, caso o objeto domínio do atributo seja um TAD.

2.2.3.2- Um apontador para um buffer do tipo do ítem 2.2.1 caso contrário.

Para um melhor entendimento desta estrutura, que será utilizada no mecanismo de instânciação, ver figura 5.8.

3- ESTRUTURAS DE DADOS DE ATUALIZAÇÃO DE VALORES: Essas estruturas estão relacionadas com o mecanismo de atualização dos valores dos atributos pertencentes a instâncias de objetos. Essas estruturas são estruturas de interface com usuário e são descritas a seguir:

3.1- Buffer de atualização - Estrutura que recebe do usuário de aplicação os valores novos e antigos dos atributos que estarão envolvidos no processo de atualização das instâncias de um objeto. Ela é composta dos seguintes campos:

3.1.1- Quantidade de atualização - Quantidade de atributos do objeto que serão atualizados.

3.1.2- Nome do objeto - Apontador para um "char" que contém o nome do objeto que será atualizado.

3.1.3- Lista de atualização: Apontador para uma lista de estrutura de atualização de atributo.

3.2- Estrutura de atualização de atributo: Essa estrutura contém os seguintes campos:

3.2.1- Nome do atributo: Um apontador para um "char" que contém os seguintes campos:

3.2.2- Identificador do atributo: Número inteiro que identifica o atributo internamente.

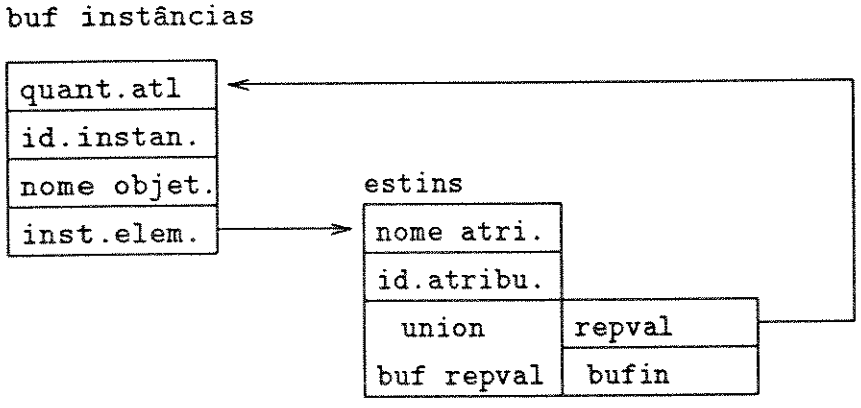


FIGURA 5.8 ESTRUTURA DE DADOS PARA INSTANCIAÇÃO

3.2.3- Estrutura de valores: Um apontador para uma união de buffer.

3.2.3.1- União de buffer: é um "union" que pode assumir um dos seguintes campos:

3.2.3.1.1- Buffer de "char": Um apontador para um buffer de valores.

3.2.3.1.2- Buffer de "id": Um apontador para um buffer de identificadores.

3.2.4- Buffer de valores: Estrutura que contém os seguintes campos:

3.2.4.1- Valor antigo: Um apontador para um "char" que contém o valor antigo do atributo em questão.

3.2.4.2- Valor novo: Um apontador para um "char" que contém o valor novo do atributo em questão.

3.2.5- Buffer de identificadores: estrutura que contém os seguintes campos:

3.2.5.1- Identificador antigo: Apontador para um identificador antigo de uma determinada instância do objeto domínio que pretende-se atualizar.

3.2.5.2- Identificador novo: apontador para um identificador novo de uma determinada instância do objeto domínio em questão.

3.2.6- Atualização de atributo: Um apontador para um buffer de atualização do tipo do ítem 3.1. Este apontador existirá somente se o objeto domínio do atributo não for TAD.

Para um melhor entendimento desta estrutura, ver figura 5.9.

5.4.4.2- Funções Associadas ao Mecanismo de Herança

Para manipular as estruturas de dados relacionados na seção 5.4.4.1 foram criadas várias funções. Essas funções implementam todo o mecanismo de hierarquia. Elas são divididas em duas partes, a saber:

- Funções relacionadas com atributo;
- Funções relacionadas com instâncias.

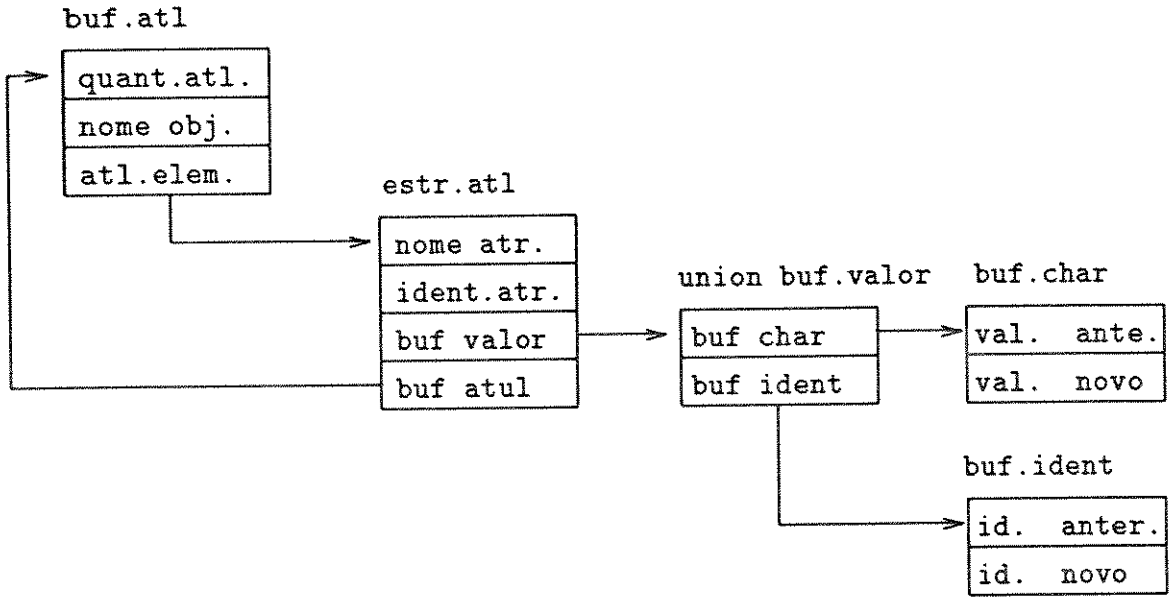


FIGURA 5.9 ESTRUTURA DE DADOS PARA ATUALIZAÇÃO DE VALORES

1- Funções relacionadas com atributos:

Nesta sub-seção serão relacionadas todas as funções que fazem manipulação dos atributos de um objeto. Elas são subdivididas em dois grupos:

- Um grupo de interface com o usuário (nível terciário);
- Um grupo associado com nível secundário.

1.1- Grupo de interface com o usuário: É através dessas funções que o usuário de aplicação tem interface com o UNICOSMOS. Elas tratam da nova filosofia de manipulação de atributo do UNICOSMOS, e estão relacionadas a seguir:

1.1.1- Obter atributos que querem herança: Obtém lista de atributos de um objeto que querem herdar atributos dos seus respectivos objetos domínios.

1.1.2- Obter atributos que querem e podem ser herdados: Obtém lista de atributos de um objeto que querem herdar atributos dos seus respectivos objetos domínios, e podem ser herdados por objetos que venham a ter o primeiro objeto como objeto domínio;

1.1.3- Obter atributos que podem ser herdados: Obtém lista de atributos de um objeto que podem ser herdados por objetos que venham a tê-lo como objeto domínio.

1.1.4- Obter atributos herdáveis: obtém lista de atributos herdáveis de um objeto, ou melhor, obtém lista dos atributos cujos objetos são folhas do grafo acíclico de hierarquia.

1.1.5- Obter dados do atributo: Obtem dados de um atributo, de um objeto, tais como nome do atributo, objeto domínio, multivaloração, permissão de herança, etc.

1.1.6- Alteração de permissão de herança: Altera a permissão de herança de um atributo de um objeto.

1.1.7- Alteração de querer herança: Altera se um atributo de um objeto quer herdar atributos do seu objeto domínio.

1.1.8- Alteração do objeto domínio: Altera o objeto domínio de um atributo de um objeto.

1.2- Grupo associado com o nível secundário: As funções desse grupo são ativadas pelas funções do item 1.1 consequentemente são funções de mais baixo nível. Elas são:

1.2.1- Listas atributos que querem herança: Esta função é ativada pela função do ítem 1.1.1, e tem a mesma filosofia desta.

1.2.2- Listas atributos que podem ser herdadas - esta função é ativada pela função do item 1.1.3, e tem a mesma filosofia desta.

1.2.3- Ler dados de um atributo: esta função é ativada pela do ítem 1.1.5, e tem a mesma filosofia desta.

1.2.4- Obter objeto domínio: Obtém o objeto domínio de um atributo de um objeto.

1.2.5- Modificar permissão de herança: Esta função é ativada pela função 1.1.6, e tem a mesma filosofia desta.

1.2.6- Modificar e querer herança: Esta função é ativada pela função 1.1.7, e tem a mesma filosofia desta.

1.2.7- Modificar objeto domínio: Esta função é ativada pela função 1.1.8, e tem a mesma filosofia desta.

1.2.8- Escrever permissão de herança: Escreve a permissão de herança de um atributo de um objeto.

1.2.9- Ler permissão de herança: Lê a permissão de herança de um atributo de um objeto.

1.2.10- Escrever querer herança: escreve o querer herança de um atributo de um objeto.

1.2.11- Ler querer herança: Lê o querer herança de um atributo de um objeto.

2- Funções relacionadas com instâncias:

Nesta sub-seção serão relacionadas todas as funções que tratam do mecanismo de instanciação na nova versão do UNICOSMOS. São elas:

2.1- Atualizar dados da instância: Esta função atualiza valores dos atributos herdáveis de um objeto, que estão associados a uma determinada instância.

2.2- Armazenar instância: Armazena os valores dos atributos herdados

de um objeto como uma instância deste mesmo objeto.
2.3- Consultar instância: Consulta uma instância de um objeto e apresenta os valores dos atributos herdados do objeto, valores estes associados a instância em questão.
2.4- Localizar instância: Localiza instâncias de um objeto em função de um valor de um atributo seu.
2.5- Verificar se instância é utilizada: Verifica se uma instância de um objeto é utilizada como valor de algum atributo de um outro objeto.
2.6- Remover instâncias: Remove instâncias de um objeto.
2.7- Obter endereço de instâncias: Obtém endereços internos de todas as instâncias de um objeto.
2.8- Ler instâncias: Lê instâncias de um objeto. Esta função ativa a função do item 2.7.
2.9- Buscar ocorrências: Busca instâncias associadas a valores em função do tipo do objeto.
2.10- Obter ocorrências associadas ao objeto: Obtém uma lista de ocorrências associadas a um conjunto de valores de um determinado tipo de objeto. Esta função ativa a função do ítem 2.9.

Todas as funções desta seção estão especificadas no relatório [RELA/89].

5.4.5- Codificação

Toda a implementação desta extensão do UNICOSMOS foi feita na linguagem C, a fim de manter a uniformidade de codificação do UNICOSMOS, já que este inicialmente estava codificado na referida linguagem. O ambiente de desenvolvimento foi uma estação de trabalho Sun com sistema operacional multiusuário UNIX.

5.4.6- O UNICOSMOS no Contexto de SBDOO

O UNICOSMOS estendido está dando sua contribuição no sentido de fornecer suporte básico aos gerenciadores de banco de dados para implementação de conceitos em Orientação por Objeto. Atualmente, o UNICOSMOS está sendo utilizado pelo GERPAC, já referenciado na seção

3.5.3, para dar o devido suporte na implementação de conceitos de Orientação por Objeto, tais como Generalização, Agregação e C-esquema. Assim como o GERPAC, outros Sistemas em desenvolvimento podem utilizar o UNICOSMOS para auxiliá-los em possíveis adequações relacionadas com Orientação por Objeto, como por exemplo: o UNICOSMOS pode auxiliar o IRIS (secão 3.5.3) no suporte a objetos complexos e na manipulação de dados envolvendo aspectos temporais, utilizando o TAD tempo do UNICOSMOS.

5.5- Resumo

Neste capítulo foram apresentadas algumas limitações da versão anterior do UNICOSMOS, bem como também as soluções propostas para sanar tais limitações no tocante ao apoio da Orientação por Objeto.

A primeira limitação apresentada foi o fato de não se ter nenhum tipo de tratamento sobre os domínios dos atributos dos objetos. O domínio de um atributo era representado por uma cadeia de caracteres. Qualquer tratamento sobre o domínio de atributo teria que ser feito a nível de usuário do UNICOSMOS.

A segunda limitação era a inexistência de Tipos Abstratos de Dados (TAD's). Este fato impunha uma restrição muito grande ao UNICOSMOS, pelo fato de não se ter um tratamento de objetos com funções específicas associadas aos mesmos, e de também não se ter nenhuma diferenciação no tratamento de representação de valores de atributos.

A terceira limitação era a ausência do mecanismo de hierarquia. Consequentemente, não se tinha suporte na implementação dos conceitos de abstração (Generalização, Agregação, Classificação, etc), e nem na definição de objetos complexos.

Sendo assim, estas três limitações acima citadas traziam um acréscimo de esforço aos usuários do UNICOSMOS. Com o intuito de sanar

tais deficiências, foram adotadas três abordagens, a saber:

1- Objeto Domínio: é um conceito utilizado pela nova versão do UNICOSMOS para acabar com a primeira limitação. Nesta nova versão, os domínios dos atributos de um objeto são definidos através de outros objetos denominados Objetos Domínios.

2- Definição de TAD's: presentemente, o UNICOSMOS possui a capacidade de definir alguns tipos básicos como Tipos Abstratos de Dados (TAD's). Existem três tipos básicos que são: String, Inteiro e Tempo. Cada um destes TAD's tem um grupo de operações associadas. Somente estas operações podem manipulá-lo. Todo mecanismo de instanciação de objeto é feito através dos TAD's que são Objetos Domínios do objeto a ser instanciado.

3- Mecanismo de Herança: basicamente, a hierarquia foi uma consequência imediata das definições de Objeto Domínio e TAD. A partir dessas definições é possível montar uma hierarquização de objetos, como é mostrada a hierarquia do objeto "A" na figura 5.4, onde este pode herdar os atributos "d", "e", "f" e "g". Vale observar também que na definição de um objeto existe um mecanismo de filtragem de atributos na hierarquização. Um objeto pode determinar quais atributos seus podem ser herdados, e pode também determinar se próprio objeto quer herdar atributos de seus objetos domínios.

A implementação final é detalhadamente descrita em [RELA/89].

Foi discutido também a situação do UNICOSMOS dentro do contexto de SBDOO, no sentido do UNICOSMOS ser utilizado por gerenciadores de banco de dados (GERPAC e outros) para auxiliá-los no suporte a Orientação por Objeto.

No próximo capítulo são apresentados alguns exemplos de aplicação dessas abordagens descritas aqui neste capítulo. Os exemplos mostram como um gerenciador de Banco de Dados poderia implementar

alguns dos conceitos de abstração que foram apresentados no capítulo 3.

CAPÍTULO 6

Exemplos de Aplicação

6.1- Introdução

Neste capítulo serão apresentados alguns exemplos de como a nova versão do UNICOSMOS poderia ser incorporada em um Gerenciador de Banco de Dados, para que este pudesse implementar conceitos de abstração, tais como: Generalização, Agregação e Agrupamento.

6.1- Implementação de Generalização

É imediato implementar o conceito de Generalização no UNICOSMOS. Considerando o exemplo da figura 6.1, têm-se os seguintes passos que devem ser obedecidos para implementá-lo:

1- Definir cada veículo - automóvel, caminhão e barco através de atributos que descrevam suas características específicas. Esses atributos levarão a objetos domínios que determinarão os domínios de valores que os mesmos atributos poderão assumir. Assim os objetos automóvel, caminhão e barco serão definidos da seguinte forma:
Os objetos automóvel, caminhão e barco são definidos objetos do tipo simples. O objeto automóvel é definido pelos atributos "preço" e "potência HP" do tipo "Inteiro". O objeto caminhão é definido pelos atributos "capacidade" e "número rodas" também do tipo Inteiro. Já o objeto barco é definido pelos atributos "região de registro" e "número de lugares" dos tipos String e Inteiro, respectivamente. Os objetos String e Inteiro são tipos básicos, através dos quais os objetos automóvel, caminhão e barco são instanciados.

2- Definir um objeto mais geral - Objeto veículo - que contém todas as características comuns aos objetos que estão sendo generalizados. No referido exemplo o objeto Veículo, que é uma generalização dos objetos Automóvel, Caminhão e Barco, pode ser definido da seguinte forma:

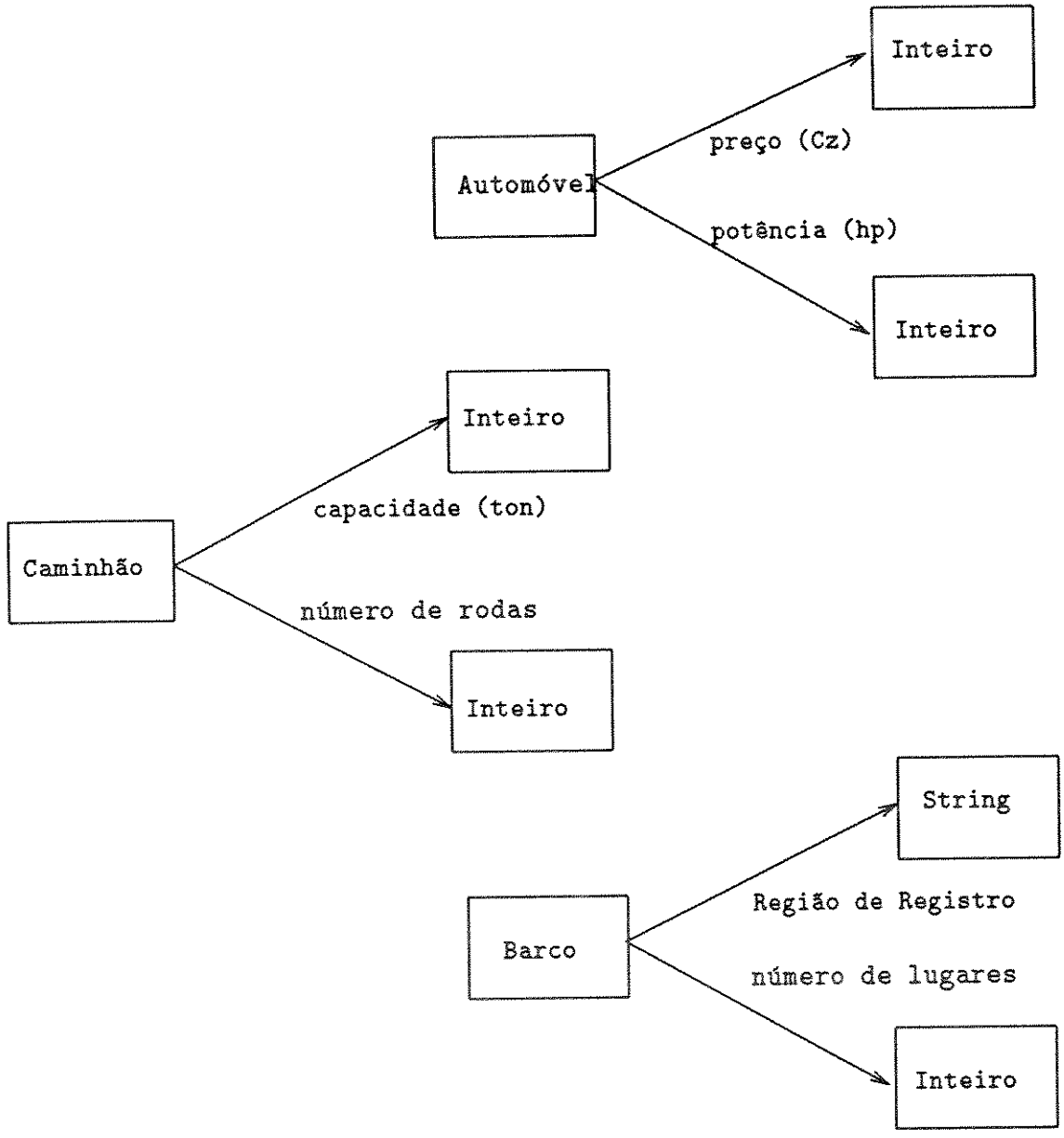


FIGURA 6.1 EXEMPLO 6.1

O objeto veículo é um objeto do tipo simples, definido pelos atributos "cor", "proprietário" e "número de registro", dos tipos string, string e inteiro respectivamente (figura 6.2). O objeto veículo permite que seus atributos sejam herdados. Para tal, na definição dos mesmos deve-se caracterizar esta permissão. Os objetos string e inteiro são tipos básicos através dos quais o objeto veículo é instanciado.

3- Fazer a conexão entre o objeto Veículo - objeto da generalização - e os objetos Automóvel, Barco e Caminhão. Isto é obtido criando um atributo "é_um_veículo", que tem como objeto domínio o objeto Veículo, nos objetos Automóvel, Barco e Caminhão, conforme mostra a figura 6.3. O Atributo "é_um_veículo" deve "querer herdar" os atributos do seu objeto domínio, ou seja, o objeto veículo.

Sendo assim os objetos Automóvel, Caminhão e Barco herdam os atributos "cor", "proprietário" e "numero registro" do objeto Veículo.

Todo este processo de definição dos atributos dos objetos, que participam da Generalização (Veículo, Caminhão, Barco e Automóvel), é definido num mesmo rótulo, por exemplo rótulo 1, o que torna o tratamento da hierarquia da generalização uniforme.

6.2- Implementação de Agregação

Para que um SGBD implemente uma Agregação utilizando o UNICOSMOS basta que os seguintes passos sejam seguidos:

1- Definir os objetos a serem agregados com suas características próprias. Considere os seguintes objetos: Carroceria, Rodas e Motor. O objeto Carroceria é um objeto simples definido através dos atributos "espessura chapa" e "modelo", dos tipos Inteiro e String, respectivamente. O objeto Rodas é do tipo simples definido através dos atributos "diâmetro" e "material", dos tipos Inteiro e String, respectivamente. O objeto Motor é também do tipo Simples definido pelos atributos "combustível" e "número cilindros", dos tipos String e Inteiro, respectivamente, conforme mostra a figura 6.4.

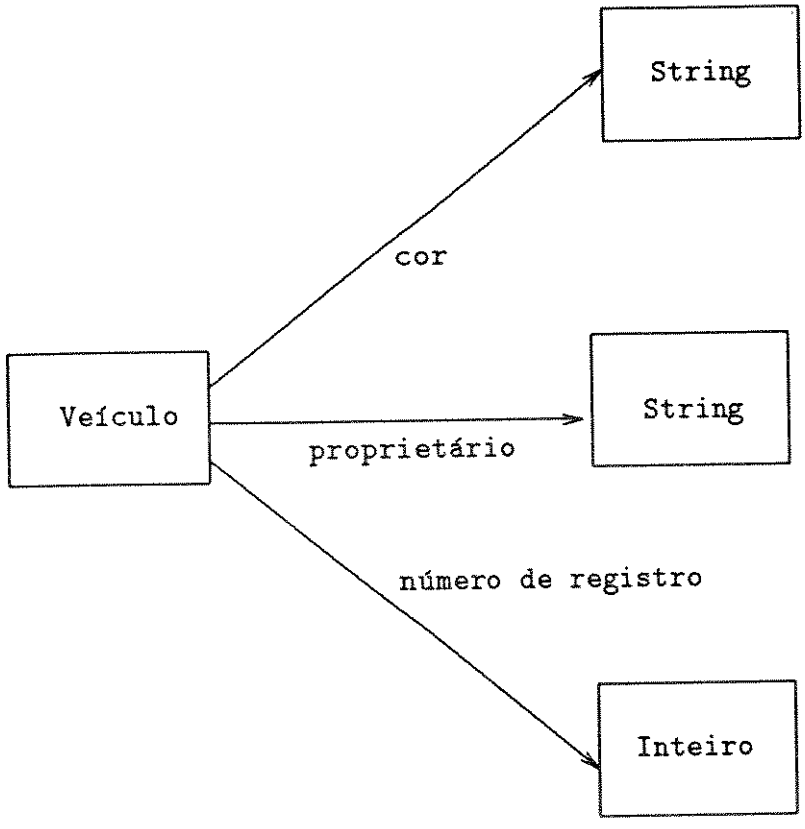


FIGURA 6.2 EXEMPLO 6.2

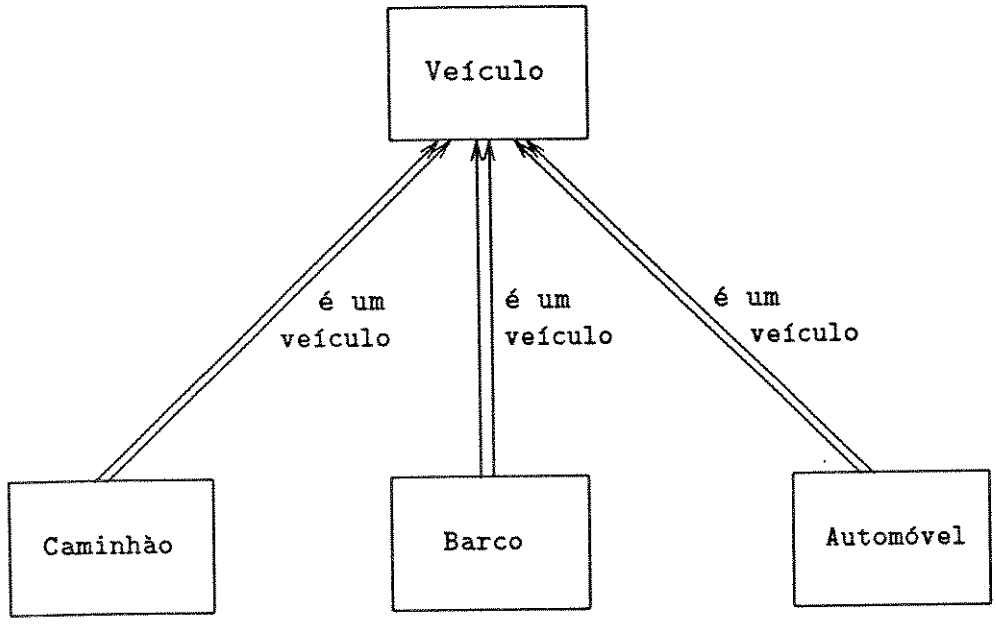


FIGURA 6.3 EXEMPLO DE GENERALIZAÇÃO

Todos atributos dos objetos Carroceria, Rodas e Motor devem ser definidos sob o mesmo rótulo, por exemplo, rótulo 1, que é o mesmo rótulo utilizado no processo de generalização do exemplo anterior. Esta medida tem como objetivo uniformizar o tratamento dos objetos Carroceria, Rodas e Motor, caso estes sejam utilizados, posteriormente, em algum processo de generalização.

2- Definir um objeto mais geral - Por exemplo: objeto Automóvel - com atributos que levam aos objetos domínios, que são considerados como objetos agregados (Carroceria, Rodas e Motor). Estes atributos são "tem_componente_carroceria" que leva ao objeto Carroceria, "tem_componente__rodas" que leva ao objeto Rodas e "tem_componente__motor" que leva ao objeto Motor. Todos estes atributos são definidos num rótulo do objeto Automóvel (figura 6.5), associados à Agregação, por exemplo, no rótulo 2. Os atributos do objeto Automóvel não querem herdar os atributos dos objetos agregados (Carroceria,Rodas e Motor), pois o objeto Veículo, do ponto de vista da agregação, simplesmente está agregando instâncias dos objetos agregados como valores de seus atributos.

Caso haja, posteriormente, a necessidade de considerar cada objeto - Carroceria, Rodas e Motor - como agregação de outros objetos, a definição dos atributos desta nova agregação deve também ser feita no rótulo de mesmo número que o definido no caso do objeto Automóvel, ou seja, rótulo 2. Isto faz o tratamento da agregação uniforme, pois todo tratamento é feito no mesmo rótulo (rótulo 2).

6.3- Implementação de Associação Elemento (ou Agrupamento)

Para um SGBD implementar uma Associação ou Agrupamento utilizando o UNICOSMOS, ter-se-ía que levar em conta os seguintes passos:

1- Definir os objetos a serem agrupados da mesma forma que os do ítem 1 da seção 6.2. Para efeito de exemplo serão agrupados os objetos Automóvel, Barco e Caminhão da seção 6.1.

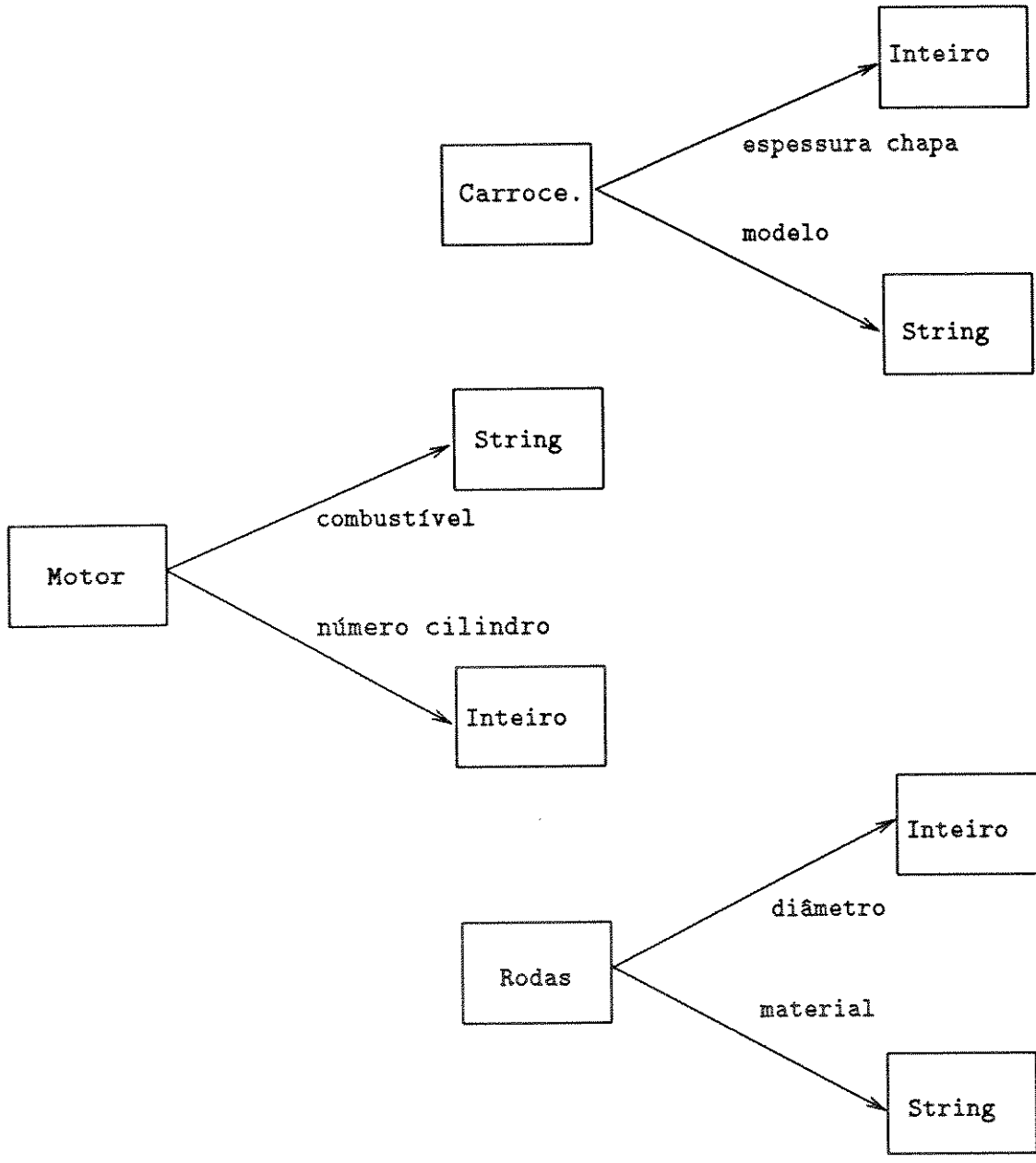


FIGURA 6.4 EXEMPLO 6.4

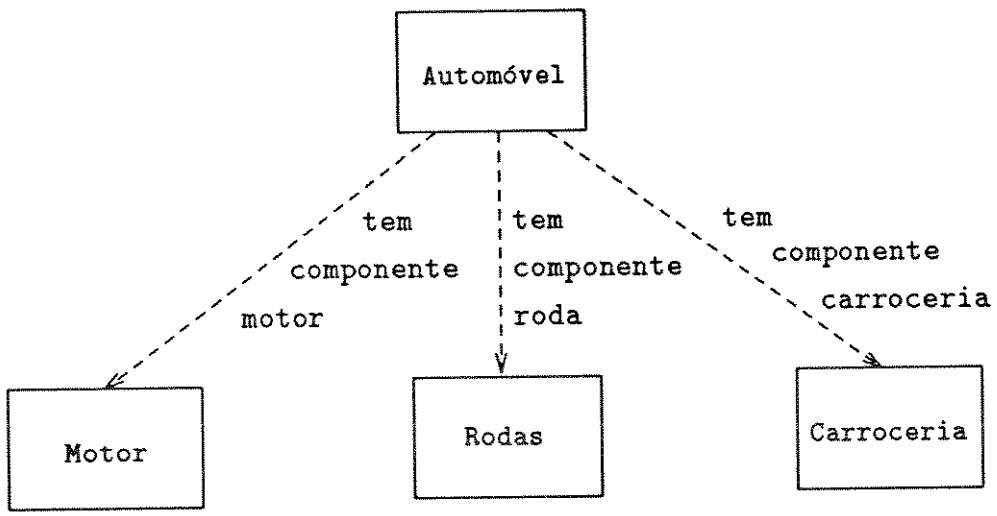


FIGURA 6.5 EXEMPLO DE AGREGAÇÃO

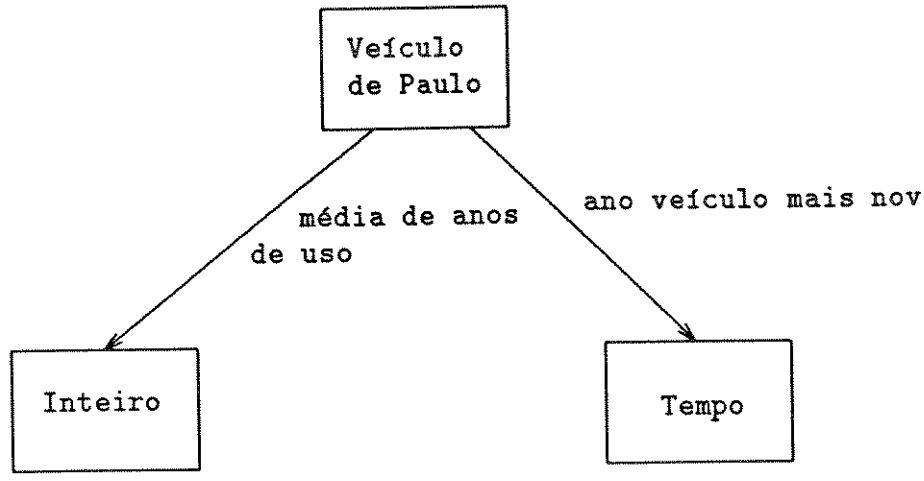
2- Definir um objeto mais geral - Por exemplo: objeto Veículo_de_Paulo que contém características associadas aos objetos que estão sendo agrupados como todo. O objeto Veículo_de_Paulo é um objeto do tipo Simples, definido pelos atributos "média_de_ano_de_uso" e "ano_veículo_mais_novo", e ambos do tipo Inteiro. O objeto Veículo_de_Paulo não permite que seus atributos sejam herdados. Para isto é necessário bloquear o mecanismo de herança na definição dos atributos do objeto Veículo_de_Paulo. Assim o mesmo objeto tem a forma mostrada na figura 6.6A. Todos estes atributos são definidos num rótulo do objeto Veículo_de_Paulo, por exemplo, no rótulo 3.

3- Definir elementos no objeto mais geral - Ainda no objeto Veículo_de_Paulo deve-se definir os atributos que levam aos objetos domínios, que serão considerados como objetos elementos (Automóvel, Barco e Caminhão). Estes atributos são "tem_elemento_automóvel" que leva ao objeto Automóvel, "tem_elemento_barco" que levam ao objeto Barco e "tem_elemento_caminhão" que leva ao objeto Caminhão (figura 6.6B). Todos estes atributos são definidos no mesmo rótulo do ítem anterior. Estes atributos elementos do objeto Veículo_de_Paulo também não querem herdar os atributos dos objetos elementos (Automóvel, Barco e Caminhão), pois o objeto Veículo_de_Paulo, do ponto de vista da associação, simplesmente está associando instâncias dos objetos elementos como valores de seus atributos.

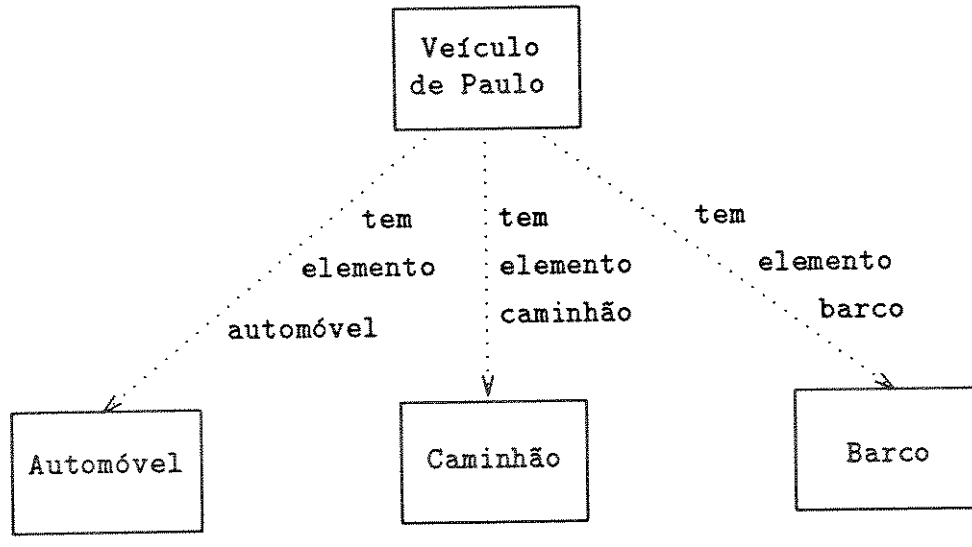
6.4- Resumo

Neste capítulo foram apresentados exemplos da aplicação do UNICOSMOS na implementação dos conceitos de abstração: Generalização, Agregação e Agrupamento.

É imediata a implementação da Generalização no UNICOSMOS. Basta definir os objetos especializados com sua características específicas. Feito isto, definir um objeto Generalização (objeto mais geral) com as características comuns aos objetos a serem generalizados e depois fazer a conexão deste com os objetos específicos através de um atributo é_um definido em cada objeto específico. Este atributo



A) Definição do objeto Associação



B) Associação de elementos

FIGURA 6.6 EXEMPLO DE ASSOCIAÇÃO DE ELEMENTOS

quer herdar os atributos do objeto mais geral e tem como domínio este objeto mais geral.

Já para implementar o conceito Agregação é preciso definir os objetos a serem agregados com suas características próprias. Depois definir um objeto Agregação (objeto mais geral) com atributos que tenham como domínios os objetos a ser agregados. Estes atributos não querem herdar os atributos dos objeto agregados e têm como nome "tem_componente_nome de cada objeto agregado".

Finalizando, para implementar o conceito Agrupamento é preciso definir os objeto a serem agrupados com suas características próprias. Depois definir o objeto agrupamento (objeto mais geral) com características dos objetos agrupados como um todo e definir atributos que levem aos objetos a ser agrupados. Estes atributos terão os nomes da forma "tem_elemento_nome de cada objeto a ser agrupado".

Capítulo 7

7- Conclusões

A complexidade dos dados na área de Projeto, serviu de motivação nesta década à pesquisa de utilização de SBD's no suporte a esta classe de aplicação.

Embora a tendência à (re)utilização de Sistemas já existentes com pequenas adaptações fosse a primeira observada, logo constatou-se que esta nova classe de aplicação colocava novas solicitações (mecanismo de herança, conceitos de abstração e tipos abstratos de dados) não oferecidas pelos Sistemas existentes. Assim, Sistemas de Banco de Dados não-convencionais é a denominação para os novos Sistemas, em pesquisa, que busca atender a especificidade desta classe de aplicação.

Esta busca de novas soluções, leva basicamente a três linhas de atuação:

* implementação de novos Sistemas, integralmente desenvolvidos com esta finalidade. Esta é a solução mais promissora a médio e longo prazo.

* adaptação de Sistemas já existentes às novas necessidades. Esta é a solução adotada no curto prazo.

* divisão da implementação entre um Núcleo que proveja funções básicas e um SGBD que ofereça as facilidades de alto nível lógico ligadas à semântica do modelo adotado. Esta é a solução aqui adotada e justifica-se em função do caráter eminentemente de experimentação em uma nova atividade de pesquisa.

Adotando-se esta última linha como solução e partindo-se do UNICOSMOS (UNICAMP/FEE Object Storage Manegement System) como Núcleo de SGBD já existente, procurou-se dotá-lo daquelas características

básicas de forma a permitir o suporte a um SGBDOO. Esta extensão do UNICOSMOS levou à inclusão das seguintes características ao Sistema original:

* Objeto Domínio;
* Mecanismo de Herança;
* Tipo Abstrato de Dados.

Com a inclusão destas novas facilidades agregou-se ao Sistema as seguintes funcionalidades:

1- Definição de Objetos Complexos

Considerando o novo conceito de Objeto Domínio, um objeto pode ser definido a partir de outros objetos. Isto trouxe uma flexibilidade muito grande ao UNICOSMOS, pois a definição de objetos complexos pode ser obtida a partir de objetos mais simples. Outra facilidade é que na nova versão existem primitivas para manipularem os domínios dos atributos (objetos domínios) de um determinado objeto.

2- Implementação de Tipos Abstratos de Dados

Com a introdução dos Tipos Abstratos de Dados básicos (Inteiro, String e Tempo) no UNICOSMOS, é possível haver um controle efetivo dos tipos de dados que são manipulados, armazenados ou removidos.

3- Controle no mecanismo de Herança

Quando do processo de definição dos atributos de um objeto, o UNICOSMOS agora permite que este objeto possa filtrar os atributos que ele quer ou não que sejam herdados por outros objetos que o utilizem como objeto domínio. Assim, o mecanismo de herança fica sob controle de cada objeto. O objeto também pode determinar se ele quer herdar ou não os atributos de seus objetos domínios. A principal vantagem desses dois aspectos aqui descritos, é que se pode ter um

controle efetivo de todo mecanismo de herança entre os objetos que se relacionam.

Para o desenvolvimento deste trabalho, primeiramente procurou-se estudar as características gerais de Sistemas de Gerência de Banco de Dados (capítulo 2), dando-se ênfase aos Níveis de Abstração de Dados com o objetivo de situar o UNICOSMOS num nível de abstração denominado Nível Organizacional.

Posteriormente, foram apresentadas as principais características de um Sistema de Banco de Dados Orientado por Objeto, dando-se ênfase aos conceitos de abstração (Classificação, Generalização, Agregação e Associação) (capítulo 3).

Depois de apresentados todos os conceitos básicos necessários para o desenvolvimento do trabalho, definiu-se o Sistema UNICOSMOS e apresentaram-se todas as características básicas do mesmo (capítulo 4).

A contribuição mais importante deste trabalho situa-se no capítulo 5, onde foram discutidas as limitações do UNICOSMOS, apresentadas as soluções para resolver tais limitações e especificadas as estruturas e funções da extensão do UNICOSMOS.

A utilização desta extensão do UNICOSMOS em situações de dar suporte a um SGBD, foi discutida no capítulo 6, com a apresentação da implementação dos conceitos de Generalização, Agregação e Agrupamento.

Esta utilização mostrou que, com a nova filosofia do UNICOSMOS, os gerenciadores que o utilizarem como núcleo têm a sua disposição facilidades para implementação dos conceitos de Abstrações que são utilizados pelos SOO.

A nova versão do UNICOSMOS trouxe diversos aspectos positivos que podem ser resumidos em quatro ítens, a saber:

1- Uniformidade no suporte a Orientação por Objeto: suporte no mecanismo de herança, nos conceitos de abstração e nos Tipos Abstratos de Dados são providos a nível de núcleo, não precisando assim ser implementados pelos gerenciadores;

2- Controle no mecanismo de herança: o UNICOSMOS provê suporte no controle de herança, permitindo aos usuários do UNICOSMOS exercer controle na definição de hierarquia dos objetos;

3- Independência de aplicação: o UNICOSMOS não é dependente da aplicação , qualquer gerenciador de Banco de Dados pode utilizá-lo como núcleo.

4- Estrutura de dados dos TAD's extensível: a estrutura de dados construída pelo UNICOSMOS é extensível, permitindo com poucas alterações acrescentar outros TAD's ao núcleo.

Pode-se acrescentar ainda a importância da extensão do UNICOSMOS no contexto atual de SGBDOO, no sentido do UNICOSMOS dar sua contribuição aos gerenciadores que ora encontram-se em desenvolvimento (IRIS [WILK/90], O2 [DEUX/90], ORION [KIM/90] e GERPAC [RICA/87]), no auxílio de possíveis adequações a serem feitasem suas versões originais, conforme discutido na secão 5.4.6.

Por outro lado, três aspectos limitantes podem ser levantados:

1- Linguagem de implementação não orientada por objeto: o fato do UNICOSMOS ter sido implementado numa linguagem não orientada por objeto (linguagem C) dificultou a implementação dos TAD's. Acredita-se que numa linguagem orientada por objeto (C++, por exemplo), a implementação dos TAD's seria mais adequada pois tal linguagem ofereceria facilidades a SOO. No entanto, por falta de um compilador orientado por objeto quando da decisão de estender o UNICOSMOS, optou-se que a extensão do núcleo fosse também implementada na linguagem inicial de implementação do UNICOSMOS (linguagem C). Com

relação a este aspecto, pretende-se num futuro breve converter a implementação do UNICOSMOS para um linguagem orientada por objeto (C++ por exemplo).

2- Impossibilidade de manipulação com várias Bases de Dados: o UNICOSMOS não permite que mais de uma Base de Dados seja manipulada simultaneamente (manipula-se uma Base de Dados por seção). Para ativar outra Base de Dados é necessário desativar o UNICOSMOS e ativá-lo novamente com a outra Base de Dados. Com relação a este aspecto, pretende-se acrescentar manipulação de múltiplas Bases de Dados na próxima versão do UNICOSMOS.

3- Controle de herança parcial: nesta nova versão do UNICOSMOS um objeto não é capaz de definir quais atributos dos seus objetos domínios ele quer herdar, ou seja, das duas uma: ou o objeto herda os atributos que seus objetos domínios permitam, ou herda nenhum. Com relação a este aspecto, pretende-se na próxima versão do UNICOSMOS permitir um controle de herança ainda mais seletivo.

Para finalizar este trabalho citam-se a seguir algumas linhas de continuação do mesmo, bem como novas linhas de trabalhos a serem desenvolvidas. Com relação a continuação, podem ser apresentadas as seguintes sugestões :

* Definir uma linguagem de interface com o UNICOSMOS que permita o usuário deste definir seus próprios TAD's, anexando estes aos tipos já existentes no UNICOSMOS.

* Definir um novo mecanismo de herança para que um objeto possa também determinar quais atributos de seus objetos domínios ele deseja herdar, já que o mesmo pode determinar que atributos seus podem ser herdados pelos objetos que o utilizem como objeto domínio.

Já com relação a novos trabalhos podem ser sugeridos as seguintes linhas de pesquisa:

* Utilização da nova versão do UNICOSMOS por parte de um SGBD que suporte um modelo genérico de objetos. Neste sentido, já existe em desenvolvimento um trabalho de especificação de um Modelo Genérico de Objeto no Instituto de Pesquisa em Software - Campinas - SP.

* Utilização do UNICOSMOS por SGBD's para implementação de conceito de abstrações (Generalização, Agregação, Agrupamento, etc). Presentemente está sendo desenvolvido no Departamento de Computação e Automação Industrial da Faculdade de Engenharia Elétrica da Unicamp um SGBD para PAC (Projeto Auxialiado por Computador) denominado GERPAC (Sistema Gerenciador de Dados para PAC).

Capítulo 8

Referências Bibliográficas

[BANC/88] Bancilhon, F.: "Object-Orineted Database Systems", Invited Lecture 7th ACM SIGART-SIGMOD-SIGACT, march 1988.

[CARV/82] Carvalho, M.A. & Golendziner, L.G.: "Um Sistema Operacional para Suporte a Banco de Dados", IV Congresso Nacional de Informática SUCESU, Rio de Janeiro, outrubro, 1982.

[CHEN/76] Chen, P.P.S.: "The Entity-Relationship Model - Toward a a Unified View of Data", ACM Transactions on Database Systems, vol.1, n. 1, march 1976, pp. 9-36.

[COST/84] Costa, M.A.M.: "Concepção duma Base de Dados", Editora Rés - Portugal -, 1984.

[DATE/86] Date, C.J.: "An Introduction to Database Systems", Addison-Wesley, 1986.

[DELG/87] Delgado, A.L.N.: "Banco de Dados no Contexto de PAC", Dissertação de Mestrado, FEE/UNICAMP, janeiro de 1987.

[DELG/88] Delgado, A.L.N., Magalhães, L.P., Ricarte, I.L.M.: "Supporting Design Environments through The ER Model", 12th IMACS World Conf. Scientific Computation, july 1988, Paris, France.

[DEUX/90] Deux O. et al.: "The Story O2"; IEEE Transactions on Knowledge and Data Engineering, vol 2, nº 2, March 1990.

[DITT/86] Dittrich, K. R.: "Object Oriented Database Systems",

Proceedings 5th ER Conference 1986.

[ENCA/83] Encarnação, J.Schlechtendahl, E.G.: "Computer Aided Design - Fundamentals and System Architectures"; Springer-Verlag, 1983.

[HAMM/76] Hammer, M.: "Data Abstractions for Data Base", Bulletin of ACM SIGMOD, vol. 8, n. 2, 1978.

[KHOS/86] Khoshafian, S. N. & Copeland, G. P.: "Object Identity", OOPSLA'86 Proceedings, september 1986.

[KIM/90] Kim, Won et al.: "Architecture of The ORION Next-Generation Database System", IEEE Transactions on Knowledge and Data Engineering, vol 2, nº 2, March 1990.

[KORT/89] Korth, H. F. & Silberschatz, A.: "Sistemas de Banco de Dados", McGraw-Hill, 1989.

[LACR/89] Lacroix, M. & Vanhoedenaghe, M.: "Inheritance and Genericity in a Complex Object Management System", Proceedings of 1989 ACM SIGMOD WORKSHOP on Software CAD Databases, Napa, California, February 1989.

[MAGA/89] Magalhães, L.P.: "Implementação de um Banco de Dados não-convencional", IV Simpósio Brasileiro de Banco de Dados, Campinas, 5 a 7 de abril de 1989, pp 77-89.

[MART/77] Martin, J.: "Computer Database Organization", Prentice-Hall, 1977.

[MATT/88] Mattos, N. M.: "Abstraction Concepts: The Basis for Data and Knowledge Modeling", 7th International Conference on Entity-Relationship Approach, november 1988.

[NEUM/82] Neumann, T. & Hornung, C.: "Consistency and Transactions in CAD Database", Proceedings of the Eighth International Conference on Very Large Data Bases, México City, september 1982.

[OLGU/90a] Olguin, C.J.M., Magalhães, L.P.: "Um Gerenciador de Transações no Contexto do GERPAC/UNICOSMOS", V Simpósio Brasileiro de Banco de Dados, Rio de Janeiro, 25 a 25 de abril de 1990, pp 205-215.

[OLGU/90b] Olguin, C.J.M.: "Gerenciamento de Transações no Contexto do GERPAC/UNICOSMOS", Dissertação de Mestrado, FEE/UNICAMP, maio de 1990.

[PECK/88] Peckham, J. & Maryanski, F.: "Semantic Data Models", ACM Computing Surveys, vol. 20, n. 3, september 1988.

[RELA/89] Silveira, M.A. Machado: "Relatório de Atividades : Especifição das Funções do UNICOSMOS Estendido", DCA/FEE/UNICAMP, novembro de 1989.

[RICA/86] Ricarte, I.L..: "Definição e Implementação de um Modelo de Dados para o Núcleo CORAS-UNICAMP", Relatório de Atividades 3, FAPESP Proc. 84/2591-1, agosto de 1986.

[RICA/87] Ricarte, I.L.: "Sistemas de Gerência de Dados para PAC", Dissertação de Mestrado, FEE/UNICAMP, junho 1987.

[SETZ/86] Setzer, V.W.: "Projeto Lógico e Projeto Físico de Banco de Dados", V Escola de Computação, Belo Horizonte, 1986.

[SHOS/86] Shoshani, A. & Kawagoe, K.: "Temporal Data Management", Proceedings of The Twelfth International Conference on Very Large Data Bases, Kyoto, August 1986.

[TSIC/77] Tsichritzis, D.C. & Lochovsky, F.H.: "Data Base Management Systems", Academic Press, New York, 1977.

[TSIC/82] Tsichritzis, D.C. & Lochovsky, F.H.: "Data Models", Prentice-Hall, Inc., Toronto, 1982.

[ULLM/82] Ullman, J.D.: "Principles of Database Systems", Computer Science Press, 1980.

[WIED/77] Wiederhold, G.: "Database Design", McGraw-Hill, 1977.

[WILK/90] Wilkinson, Kevin; Lyngbaek, Peter & Hasan, Waqar: "The IRIS Architecture and Implementation"; IEEE Transactions on Knowledge and Data Engineering, vol 2, nº 2, March 1990.

[WU/83] Wu S.-T.: "Implementação de um Núcleo de Banco de Dados baseado na Filosofia CORAS"; Documentação Preliminar, DCA.FEE/UNICAMP, 1983.