Sistemas de Bancos de Dados Orientados a Objetos

Ivan Luiz Marques Ricarte
Depto. Engenharia de Computação e Automação Industrial
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas
mailto:ricarte@dca.fee.unicamp.br
http://www.dca.fee.unicamp.br/~ricarte/

Setembro de 1998

Resumo

Sistemas de banco de dados representam um papel fundamental em aplicações de processamento de informação não-numérica, onde a utilização de tais sistemas permite facilitar a manutenção de dados consistentes que podem ser compartilhados por diversas aplicações, as quais não precisam conhecer o modo como tais dados são internamente armazenados ou representados. A tecnologia de banco de dados já está madura para diversas aplicações, principalmente nas áreas administrativa e comercial.

Entretanto, novas categorias de aplicações têm demandado o gerenciamento de dados mais complexos — são aplicações tais como sistemas de apoio a projetos e de automação de escritórios. Para essas aplicações, o paradigma de orientação a objetos surge como alternativa adequada para a representação e manipulação dos dados, mas é preciso ainda integrar de forma adequada esta tecnologia a sistemas de banco de dados.

Sistemas gerenciadores de bancos de dados orientados a objetos constituem a proposta corrente para a solução desse problema. Neste trabalho apresentam-se os aspectos básicos da modelagem e da tecnologia de tais sistemas, com uma breve discussão sobre tendências na área de processamento de informação não-numérica. Pela análise dessas tendências, observa-se que as técnicas desenvolvidas para a integração de orientação a objetos a banco de dados oferecem contribuições que certamente estarão presentes nas novas plataformas de processamento por objetos distribuídos. Assim, o estudo de sistemas gerenciadores de base de objetos devem ainda permanecer como objeto de interesse por um longo período.

1 Motivação

Um **banco de dados** é uma coleção de dados relacionados que pode ser armazenada sob alguma forma física. Os dados armazenados em um banco de dados representam algum aspecto específico do mundo real — um **universo de discurso** de onde os dados são obtidos — e apresentam algum grau de coerência lógica entre seus componentes. Portanto, uma coleção aleatória de dados não constitui um banco de dados.

Um **sistema gerenciador de banco de dados** é o software que permite criar, manter e manipular bancos de dados para diversas aplicações. A criação e manutenção de bancos de dados é tarefa de uma pessoa ou grupo de pessoas, normalmente referenciada como o **administrador do banco de dados**. A manipulação do banco de dados — atualizações e consultas — é realizada direta ou indiretamente, através de programas aplicativos, pelos usuários do banco de dados. O sistema gerenciador de banco de dados pode ser de propósito geral ou específico para alguma aplicação.

Um **sistema de banco de dados** é constituído por um banco de dados e por um sistema gerenciador de banco de dados, como mostrado na Fig. 1 [16]. Um sistema de banco de dados é usualmente uma aplicação que serve de suporte a outras aplicações, tais como folha de pagamento, controle de pessoal e informações bancárias.

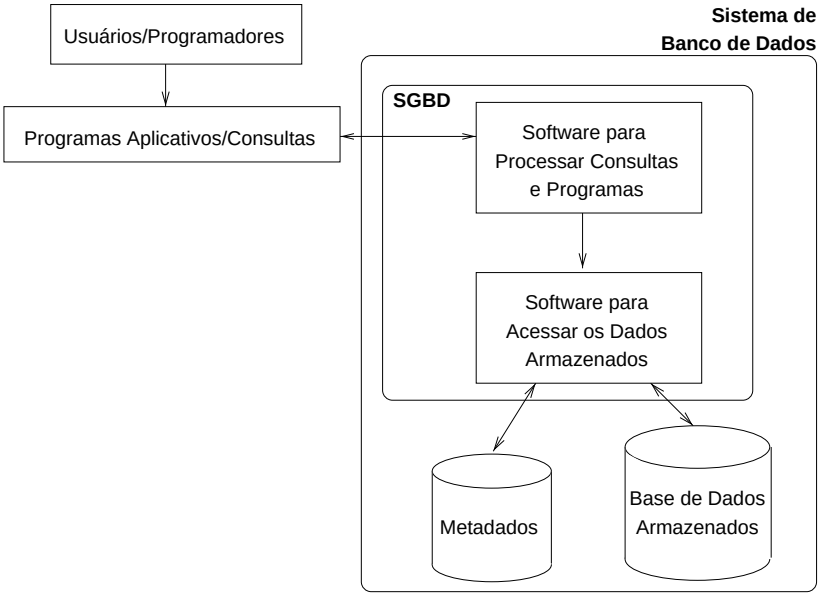

Figura 1: Sistema simplificado de banco de dados.

Tradicionalmente, sistemas de banco de dados estavam relacionados a aplicações nas áreas administrativa e comercial. Com o tempo, as vantagens obtidas com a utilização de um sistema de banco de dados para suportar a armazenagem da informação de uma aplicação — principalmente a independência da aplicação com relação à estrutura interna de armazenagem e a possibilidade de

compartilhar dados entre aplicações — passaram a ser desejadas em outras classes de aplicações, tais como projetos de engenharia (CAD/CAE — *Computer Aided Design, Computer Aided Engineering*), automação industrial e de escritórios (CAM — *Computer Aided Manufacturing,* OA — *Office Automation*), sistemas especialistas de suporte à tomada de decisões (DSS — *Decision Support Systems*) e sistemas de suporte ao trabalho cooperativo (CSCW — *Computer Supported Cooperative Work*). Muitas destas aplicações "não convencionais" impõem requisitos ao sistema de banco de dados que não estavam presentes nas aplicações tradicionais, tais como lidar com maiores volumes de dados, a necessidade de manipulação de dados em formatos não-alfanuméricos — incluindo informação multimídia — e o tratamento de esquemas evolutivos e dinâmicos.

As necessidades dessas novas categorias de aplicações constituem a maior motivação para o surgimento de novas abordagens para o gerenciamento de dados. É neste sentido que a integração do paradigma de orientação a objetos a sistemas de banco de dados tem surgido como a tendência mais promissora em termos de gerenciamento de dados para as mais diversas categorias de aplicação.

Neste trabalho apresenta-se um breve relato da evolução dos sistemas de banco de dados em direção a sistemas gerenciadores de base de objetos. Na Seção 2 descreve-se a "primeira onda" do processamento da informação não-numérica, culminando com a proposta do modelo relacional e o desenvolvimento dos primeiros sistemas gerenciadores de banco de dados relacionais. Sistemas gerenciadores de banco de dados relacionais ainda dominam boa fatia do mercado de processamento de informação, utilizando uma tecnologia dominada e sendo perfeitamente adequados para grande parte das aplicações comerciais e administrativas.

Na Seção 3 apresentam-se os resultados de um período de transição, onde a limitação dos modelos tradicionais de representação de dados tornou-se patente para as novas aplicações computacionais que surgiam. Nesse contexto foi que a orientação a objetos foi proposta como o paradigma adequado de representação de dados, embora ainda fosse incerto como o casamento entre orientação a objetos e sistemas de banco de dados devesse ocorrer. Foi um período de experimentação, durante o qual foram lançados os fundamentos para a modelagem de dados segundo o paradigma de orientação a objetos.

Na Seção 4 apresenta-se o resultado do amadurecimento da aplicação do paradigma de orientação a objetos a sistemas de banco de dados. Aspectos da tecnologia que oferece suporte à implementação de sistemas gerenciadores de base de objetos são apresentados. A Seção 5 apresenta alguns exemplos representativos de gerenciadores de objetos, onde algumas destas técnicas são ilustradas.

Finalmente, na Seção 6 são apresentadas as tendências para o futuro do gerenciamento de dados em aplicações computacionais, que indicam ser o paradigma de orientação a objetos a base para a evolução dessas aplicações. Assim, sistemas gerenciadores de base de objetos deverão de fato representar um papel fundamental como repositórios de informação no futuro das aplicações computacionais por um longo período.

2 Evolução de sistemas de banco de dados

Para melhor compreender o estado atual da tecnologia de banco de dados, é interessante observar como esta tecnologia surgiu e evoluiu.

O surgimento dos primeiros computadores está intimamente ligado à realização de cálculos numéricos repetitivos. Este era a motivação de Babbage para o projeto e elaboração de seu Engenho Analítico no Século XIX. Da mesma forma, os trabalhos pioneiros de Zuse, na Alemanha, e de Atanasoff, Aiken, Mauchly e Eckert, nos Estados Unidos, eram computadores voltados para a solução de problemas numéricos. Por um longo período, o desenvolvimento de computadores científicos (para aplicações numéricas) foi independente do desenvolvimento de máquinas para aplicações comerciais. Somente no início da década de 1960 a integração entre estas duas tendências ocorreu de modo mais acentuado, principalmente através dos esforços da IBM para otimizar sua linha de produção de máquinas para as duas áreas de aplicação.

O conceito que motivou o surgimento de bancos de dados é quase tão antigo quantos os primeiros sistemas computacionais para aplicações não-numéricas, remontando ao final da década de 1950. Entretanto, o surgimento de sistemas gerenciadores de banco de dados sofreu influências também de outras áreas de aplicação, notadamente da área de defesa e de aplicações militares.

2.1 Primórdios do processamento não-numérico

O desenvolvimento de linguagens de definição de dados foi um fator importante no desenvolvimento de sistemas de banco de dados [18]. Entre elas destacam-se COMPOOL e COBOL.

COMPOOL foi uma linguagem desenvolvida no Massachusetts Institute of Technology no contexto do projeto SAGE, de controle de tráfego aéreo, suportado pela Força Aérea norte-americana. Ela fornecia um mecanismo para definição de atributos compartilhados por centenas de programas em tempo-real.

COBOL é uma linguagem de programação com uma seção centralizada para a definição de dados, a DATA DIVISION. A criação desta linguagem foi coordenada pelo consórcio CODASYL para integrar linguagens de aplicações comerciais, como FACT (Honeywell), GECOM (General Electric) e Commercial Translator (IBM).

O uso de uma linguagem comum de definição e processamento dos dados, como no caso de COBOL, era atrativo pois a linguagem de definição seguia o modelo da sistemas baseados em fitas e cartões. Entretanto, cada fabricante adotava seu próprio esquema de armazenamento, o que dificultava a transferência de dados entre sistemas de fabricantes distintos.

Geradores de relatórios foram criados para facilitar o trabalho de produzir saídas formatadas a partir de dados armazenados no sistema. Embora tais tarefas pudessem ser (e eram) produzidas a partir de linguagens de programação convencionais, os programas produzidos eram muito extensos. O uso destes *report generators* permitia então que usuários examinassem e manipulassem grandes volumes de dados, sendo um precursor das linguagens de consulta a bancos de dados.

A General Electric desenvolveu o *Generalized Report Generator* para o centro de operações da Atomic Energy Commission em Hanford, estado de Washington, em 1956. Este sistema foi o ancestral de todos os sistemas de geração de relatórios, incluindo os sistemas da IBM, RPG (System/360, 1965) e seu sucessor, RPG II (1967).

2.2 Desenvolvimento de gerenciadores de banco de dados

Os primeiros protótipos desenvolvidos com a funcionalidade de gerenciar dados de uma aplicação, se comparados com os sistemas atuais, oferecem pouca semelhança de funcionalidade. Entretanto, foram esforços significativos para a época e que influenciaram de diversos modos o desenvolvimento dos sistemas posteriores [18]. Alguns destes esforços são citados a seguir.

Base-Ball foi um sistema especialista de acesso a dados através de uma interface em linguagem natural. Os dados, assim como o vocabulário de interação entre usuários e a base de fatos, eram relacionados a beisebol.

B-Tree é uma estrutura extensivamente utilizada para acesso eficiente a arquivos, cuja primeira implementação foi apresentada em artigo publicado na *Communications da ACM* de janeiro de 1962, por Collilla e Sams.

QUERY era uma linguagem de consulta a dados com tradução para linguagem de máquina, descrita em artigo na *Communications da ACM* também de janeiro de 1962, por Cheatham e Warshall.

Recol *(Retrieval Command-Oriented Language)*, era um sistema de gerenciamento de acesso à informação generalizado para unidades de fita, descrito por Climenson na *Communications da ACM*, de março de 1963.

ACSI-Matic foi um sistema projetado para explorar o uso efetivo de memória e inferências. Foi o primeiro sistema a oferecer um pacote generalizado para recuperação de dados em discos com requisições em *batch*, a oferecer gerenciamento dinâmico de memória principal e a utilizar um montador com rotina para alocação dinâmica de memória. Era parte de um projeto do exército norte-americano, descrito inicialmente em 1960, com um protótipo implementado em 1964 pela RCA.

ADAM *(Advanced Data Management System)* foi um sistema desenvolvido com recursos da Força Aérea norte-americana a partir do protótipo ETF *(Experimental Transport Facility)*, da MITRE Corporation. Tinha por objetivo ser um laboratório para modelagem, desenvolvimento de protótipos, verificação de projeto e avaliação de sistemas gerenciadores de dados.

Colingo era um conjunto de sistemas de gerência de dados orientados a fitas, oferecendo uma estrutura estilo COBOL em máquinas IBM.

Após estes primeiros esforços no sentido de oferecer sistemas manipuladores de dados, surgiram de fato as primeiras famílias de gerenciadores de dados. Esforços pioneiros incluíam o *Mark IV* para máquinas IBM1401, desenvolvido por Postley na Informatics no período 1964–1968; o IDS *(Integrated Data Store)*, desenvolvido por Bachman no período 1964–1966; e o *Formatted File*, desenvolvido pela força aérea norte-americana no período 1961–1965. O *Formatted File* influenciou

o produto da IBM, GIS *(Generalized Information System)*, que introduziu como contribuições o armazenamento de tabela com o formato do arquivo e a manipulação de grandes arquivos.

O **IDS** combinava tecnologia de acesso a posições aleatórias com linguagens procedimentais de alto nível (inicialmente, Gecom; posteriormente, COBOL, em 1966) oferecendo um modelo de dados do tipo rede. Nesta mesma família de gerenciadores incluem-se também APL *(Associative PL/I)*, desenvolvido na General Motors (1966), e dataBASIC, da Honeywell (1970). As principais características destes gerenciadores, que foram contribuições importantes para gerenciadores de dados que se seguiram, incluem:

- verbos para manipulação de dados na interface de alto nível;
- descrições separadas para armazenamento e para programação;
- princípios elementares de manutenção da consistência, com mecanismos para remoção e inserção implícitas;
- conceitos de paginação de dados; e
- acesso compartilhado aos dados.

Os primeiros desenvolvimentos comerciais em gerenciadores de banco de dados foram marcados pelos esforços da CODASYL, com o *Data Base Task Group* (DBTG, 1969–1971) e o projeto IMS (1966–1969) da Agência Espacial norte-americana (NASA), que influenciou o produto IMS/360 da IBM.

A abordagem do DBTG foi propor uma extensão a COBOL para a manipulação de bancos de dados. Esta abordagem foi adotada por diversos produtos, entre os quais destacam-se o **IDMS** *(Integrated Data Management System)*, desenvolvido pela IBM para IBM System/360; **IDMS-11** e **DBMS-10**, implementações de IDMS pela Digital Equipment Corporation para suas máquinas PDP-11/45 e PDP-10, respectivamente; e o **IDS/II**, uma adaptação de IDS realizada pela Honeywell para adequação ao padrão DBTG.

O projeto IMS foi resultado de um desenvolvimento conjunto das empresas Rockwell, IBM e Caterpillar para apoio ao programa Apollo, da NASA. Foram frutos deste projeto os produtos **ICS/DL/I** *(Information Control System, Data Language/I)*, da Rockwell; e **IMS/360**, da IBM desenvolvido para suas máquinas da família System/360.

2.3 O modelo relacional de dados

A maior contribuição dos primeiros gerenciadores de dados foi introduzir a separação entre os dados armazenados e a descrição da estrutura dos dados, o **esquema** de dados. As restrições sobre tipos de dados e de esquemas que são permitidos por um sistema constituem seu **modelo de dados**.

A abordagem proposta pelo grupo CODASYL/DBTG suporta um modelo de dados conhecido posteriormente como o modelo rede *(network)*, enquanto que a abordagem de IMS introduziu o modelo de dados hierárquico.

Ambos modelos de dados têm por princípio o estabelecimento de ligações entre grupos de registros. No modelo de dados rede os registros são organizados na forma de grafos dirigidos. No modelo de dados hierárquico cada registro tem um único antecessor, limitando a organização dos dados a uma estrutura de árvore, ou hierarquia.

Apesar do avanço em relação aos primeiros sistemas gerenciadores de dados, os modelos rede e hierárquico apresentavam como limitação uma forte dependência entre as estruturas internas de armazenamento e a representação abstrata dos dados, influenciando assim os mecanismos de manipulação. Uma característica genérica dos produtos baseados nestes modelos estava no modo como as ligações entre registros era armazenada, usualmente utilizando endereços físicos, com uma forte dependência entre a representação do armazenamento e a máquina onde o sistema estava implementado. No entanto, para aquela época portabilidade não era uma questão maior, pois muitos clientes concentravam suas aplicações em uma única máquina de grande porte.

Um grande marco na evolução em modelos de dados foi a proposta do **modelo relacional** [12], que apresentava um modelo matemático para a representação e manipulação de dados totalmente independente de características de implementação.

O modelo relacional oferece uma visão do conjunto de dados como uma coleção de tabelas ou relações (no sentido matemático da palavra), onde cada tabela é um conjunto de registros ou tuplas, n-uplas ordenadas de valores literais pertencentes a domínios atômicos. O modelo oferece a possibilidade de expressar consultas e manipulações sobre a base de dados usando linguagens independentes de sistemas, baseadas em álgebra relacional e em cálculo relacional [26].

A **álgebra relacional** oferece operações básicas como seleção de tuplas dentre as pertencentes a uma relação (σ), a projeção de colunas da tabela gerando uma nova relação com apenas as colunas selecionadas (Π), o produto cartesiano de duas relações gerando uma nova relação com todas as combinações possíveis de tuplas entre as duas relações originais ($\times$), a união de duas relações com estruturas compatíveis ($\cup$) e a diferença de duas relações com estruturas compatíveis ($-$), onde o resultado é uma terceira relação que contém as tuplas da primeira relação que não estão presentes na segunda relação.

As principais operações derivadas das operações básicas na álgebra relacional incluem:

- a interseção de conjuntos, que pode ser expressa em termos de diferenças,

$$R \cap S = R - (R - S)$$

- a junção com predicados, que pode ser expressa em termos de seleção e produto cartesiano,

$$R \bowtie_\Theta S = \sigma_\Theta R \times S$$

- a junção natural, onde o predicado de junção é a igualdade entre valores de atributos com domínios compatíveis nas duas relações envolvidas.

A álgebra relacional permite expressar as operações elementares que devem ser seguidas, em ordem, para a obtenção do resultado desejado. Neste sentido, é uma linguagem procedimental, pois suas expressões definem sequências de operações. No entanto, não é uma linguagem procedimental completa, pois não inclui as construções de controle de execução usualmente presentes em linguagens de programação.

O **cálculo relacional** é baseado em construções não-procedimentais, usando descrições formais do que é desejado na manipulação. Duas variantes básicas são o cálculo relacional de tuplas e o cálculo relacional de domínios. É possível mostrar que estas três linguagens relacionais — a álgebra, o cálculo de tuplas e o cálculo de domínios — têm poder equivalente de expressão [26].

As linguagens relacionais, matemáticas, ofereceram os fundamentos para o desenvolvimetno de diversas linguagens computacionais de consulta e manipulação de bases de dados. A linguagem SQL usa uma combinação de construções da álgebra e do cálculo relacionais. Outras linguagens de consulta conhecidas incluem Quel (baseada no cálculo relacional de tuplas) e QBE (baseada no cálculo relacional de domínios).

As vantagens do modelo relacional podem ser resumidas em dois pontos principais. O primeiro é seu forte suporte teórico para descrição da representação e da manipulação dos dados. O segundo é a facilidade oferecida para o mapeamento de tuplas para os dispositivos de armazenamento, com uma associação praticamente direta para registros de dimensão fixa em arquivos com acesso direto em disco. Estas vantagens motivaram os pesquisadores da época a investir no estudo de mecanismos de suporte à implementação de sistemas gerenciadores de banco de dados relacionais.

A década de 1970 foi assim marcada pelo desenvolvimento de protótipos de sistemas relacionais. Os principais projetos desta fase foram o *System R*, da IBM, e INGRES, da Universidade da Califórnia em Berkeley.

O **System R** [2, 6] foi um projeto-protótipo do laboratório de pesquisa da IBM em San Jose, Califórnia, iniciado em 1974. Concluído em 1979, deu ainda origem ao produto SQL/Data System (1981) e ao projeto R* para estudo de sistemas de bancos de dados distribuídos. Posteriormente, em 1983, a IBM lançou também o produto DB2 para manipulação de grandes bases de dados relacionais. As principais contribuições deste projeto foram a definição da linguagem de consulta SQL, a introdução de mecanismos para a compilação e otimização de consultas, a introdução de mecanismos de integração da linguagem de consulta a linguagens de programação, e a introdução dos conceitos de *locking* em duas fases e com múltiplas granularidades.

INGRES [45, 46] foi um sistema gerenciador de banco de dados experimental desenvolvido por um grupo de pesquisa da Universidade de Berkeley. Posteriormente, um produto do mesmo nome derivado deste projeto foi lançado pela empresa Relational Technology, Inc., sendo atualmente comercializado pela empresa Ingres Inc. O protótipo inicial do sistema foi desenvolvido em um PDP-11, com sistema operacional Unix. INGRES foi projetado como um sistema gerenciador com diversos

processos concorrentes com mecanismos de comunicação inter-processos. A linguagem Quel foi introduzida por este projeto, juntamente com um mecanismo de *embedding* para a linguagem de programação C. As técnicas de otimização de consulta não eram realizadas por pré-compilação, como em System R, mas durante a interpretação das consultas, através de um processo de decomposição de consultas. Deste projeto derivou-se também o projeto POSTGRES [41], sistema gerenciador de banco de dados relacional com características avançadas de controle de restrições e apoio a novos tipos de dados.

Estes protótipos influenciaram o desenvolvimento de muitos produtos, e sistemas gerenciadores de banco de dados relacionais atingiram grande sucesso comercial. Entretanto, o desenvolvimento da tecnologia computacional vem motivando a implementação de aplicações cada vez mais complexas, cujas necessidades de representação de informação nem sempre eram adequadamente atendidas por tais produtos. Assim, o modelo relacional de dados enfrentou (e continua enfrentando) algumas críticas, das quais se destacam:

1. é apenas adequado para aplicações "estruturalmente simples";
2. o poder de expressão das tabelas é limitado;
3. desenvolver um bom projeto relacional, normalizado, não é tarefa simples.

É a partir de situações como essas que a busca por novos mecanismos de expressão de dados é motivada. Neste caso, havia uma necessidade de se buscar modelos de dados com maior poder de expressão que o modelo relacional de dados. As abordagens resultantes podem ser resumidas em três grandes categorias de modelos: os modelos relacionais estendidos, os modelos de dados conceituais e os modelos orientados a objetos.

Os **modelos relacionais estendidos** incorporam mecanismos não previstos na proposta original do modelo relacional [13]. Os principais mecanismos descritos em diversas propostas incluem procedimentos armazenados, versões, objetos binários (BLOBs) e definição de tipos abstratos de dados. A não existência de um modelo estendido unificado dificulta uma aceitação mais ampla dessa linha de evolução de modelos. Sistemas representativos desta tendência incluem op já citado POSTGRES, da Universidade da Califórnia em Berkeley, que evoluiu posteriormente para um produto comercial denominado Montage [50], e Starburst [27], do IBM Almaden Research Center.

Os **modelos de dados conceituais** oferecem construções no nível mais próximo do projetista para organização do mundo de interesse [5, 38]. O termo "conceitual" refere-se ao fato de que as construções oferecidas por estes modelos estão em níveis de abstração mais próximos do universo de interesse do usuário do que dos aspectos de implementação da máquina. Várias propostas de arquiteturas de bancos de dados utilizam três níveis de abstração para bancos de dados (Fig. 2), embora em geral não haja um consenso sobre os nomes de cada nível. Aqui adotou-se a nomenclatura de Parsaye [36]. De qualquer modo, os modelos de dados conceituais estão próximos do nível mais alto.

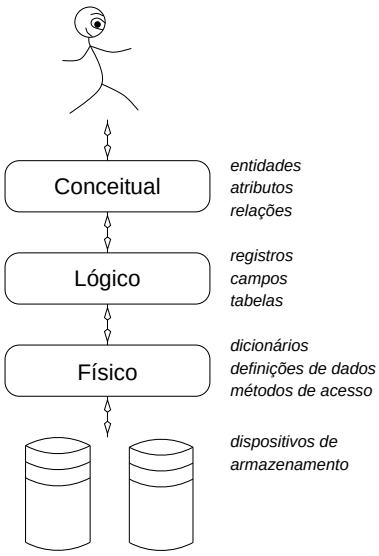

Figura 2: Arquitetura de banco de dados.

Nesta estruturação de banco de dados em uma arquitetura de três níveis, a informação do mundo real é modelada por usuários independentemente das estruturas computacionais que serão utilizadas no nível conceitual. Neste nível, construções para a representação dos dados são (por exemplo) entidades, atributos e relações. No nível conceitual, a informação é estruturada, onde algumas construções computacionais podem já ser consideradas. Construções para a representação dos dados seriam registros, campos e tabelas. O nível interno ou físico corresponde a representações do dado onde são considerados detalhes de armazenamento, representação e desempenho. Neste nível seriam consideradas construções computacionais como dicionários, definições de dados, métodos de acesso e dispositivos de armazenamento.

Algumas das principais propostas no sentido de oferecer modelos de dados conceituais incluem o modelo entidade-relacionamento, que organiza a informação em tipos de entidades e tipos de relacionamentos; os mecanismos de abstrações em bancos de dados, agregação e generalização; e os modelos semânticos e funcionais, que organizam a informação através de objetos e funções.

O **modelo entidade-relacionamento** [11] foi proposto por Chen em 1976 como um modelo conceitual, não estando diretamente relacionado a preocupações relativas à implementação de suas primitivas. Por este motivo, juntamente com a proposta do modelo há descrições para realizar o mapeamento do modelo E-R para os modelos relacional e rede, para os quais havia sistemas gerenciadores de banco de dados disponíveis. Algumas das características de interesse do modelo E-R são o encapsulamento dos detalhes estruturais em construções de alto nível (tipos de entidade e tipos de relacionamento) e mecanismos para manutenção de integridade e consistência dos dados.

Os conceitos de **abstração em bancos de dados**, agregação e generalização, foram propostos

por Smith e Smith [44] em 1977 como ferramentas de projeto de banco de dados, juntamente com propostas de mecanismos para seu suporte em sistemas gerenciadores de banco de dados, principalmente relacionais. Estes conceitos básicos reaparecem em muitos outros modelos sob diferentes formas.

Modelos semânticos e **modelos funcionais** [22, 37] estão voltados principalmente para a integração entre sistemas de bancos de dados e sistemas de representação de conhecimento, sendo fortemente influenciados por técnicas da inteligência artificial. Assim, incorporam mecanismos para expressar tipos complexos, relacionamentos entre objetos (incluindo hierarquias na forma IS-A) e derivação de esquemas. Também não houve uma abordagem de modelagem semântica universalmente aceita, da mesma forma como ocorreu com modelos relacionais estendidos.

A abordagem de modelagem por orientação a objetos, apesar de próximo em princípio aos modelos de dados conceituais, vem recebendo atenção e aceitação muito mais ampla. Por este motivo, esse tópico será tratado em mais detalhes na próxima seção.

3 Modelagem de dados orientada a objetos

Os sistemas comerciais de gerência de banco de dados, baseados principalmente no modelo relacional de dados, obtiveram ampla aceitação e divulgação, em diversos níveis de escala de aplicação. Entretanto, tais sistemas mostraram-se inadequados para uma ampla gama de aplicações que surgiu com a ampliação do poder de processamento dos computadores modernos. Essas aplicações incluem, entre outras, atividades em projetos de engenharia (siatemas CASE para engenharia de *software*, CAD para desenvolvimento de *hardware* e apoio à engenharia mecânica, engenharia civil e arquitetura), autoria e manipulação de documentos eletrônicos, manufatura e controle.

O paradigma de orientação a objetos surgiu como uma alternativa para a modelagem de bancos de dados que permitiria incorporar muitos dos conceitos propostos pelas outras abordagens alternativas ao modelo relacional de dados. Embora inicialmente tenha também sofrido pelo excesso de abordagens não-compatíveis, sua ampla aceitação motivou esforços de padronização que vêm unificando o chamado “modelo de objetos”.

Deste modo, o desenvolvimento de sistemas gerenciadores de bases de objetos (SGBOs) surgiu como uma alternativa com o potencial adequado para atender à demanda de tais aplicações, sendo por este motivo objeto de intenso interesse de pesquisa e desenvolvimento. Antevia-se que tais sistemas ofereceriam o ferramental necessário para as novas categorias de aplicações que não estavam sendo adequadamente suportadas por bases de dados relacionais.

3.1 Princípios da orientação a objetos

O conceito de **objeto** como construção de programação foi inicialmente introduzido na linguagem Simula 67, onde a execução de um programa era realizada através da composição da execução de vários objetos inter-relacionados. Objetos com uma mesma estrutura constituíam uma **classe**, que

era descrita por uma declaração de classe. Os conceitos introduzidos por Simula estiveram por muito tempo limitados à utilização em aplicações de simulação. Apenas a partir da década de 1980 houve um maior esforço no sentido de valorizar tais conceitos e incorporá-los a outras linguagens de programação.

Nygaard [31] define programação orientada a objetos como

> *. . . o desenvolvimento de um sistema em um processo de informação através de transformações de seu estado. A substância do processo é organizada como componentes do sistema, denominados objetos. Uma propriedade mensurável da substância é uma propriedade de um objeto. Transformações de estado são consideradas como ações por objetos.*

Um aspecto chave em programação orientada a objetos é o conceito de abstração, que permite abstrair (nesta definição) substância como declarações de classes; valores e propriedades mensuráveis como declarações de tipos; e ações como declarações de métodos ou procedimentos. Um objeto **encapsula** uma estrutura que só é manipulável através de métodos componentes de uma interface pública.

Como Stroustrup destaca [47], programação orientada a objetos é uma técnica, um paradigma. Uma linguagem de programação orientada a objetos nada mais é do que uma linguagem de programação que favorece a aplicação dessa técnica. Entretanto, é possível realizar programação orientada a objetos sem usar uma linguagem de programação orientada a objetos, assim como não há garantia de que um programa será orientado a objetos simplesmente porque foi implementado em uma linguagem de programação orientada a objetos.

Além dos conceitos fundamentais de classes, objetos e métodos, dois conceitos inerentes à programação orientada a objetos são herança e polimorfismo. **Herança** é o mecanismo que permite reaproveitar declarações de classes, criando classes derivadas a partir de declarações de classes bases já existentes. Juntamente com a técnica de padrões de projeto *(design patterns)*, herança torna-se um mecanismo de reuso poderoso. **Polimorfismo** é o mecanismo que permite expressar que ações similares em classes distintas venham a ter um comportamento uniforme. Por exemplo, espera-se que um método denominado `print` em duas classes distintas tenha um comportamento comum, independentemente das diferenças internas de sua implementação.

Embora Simula tenha introduzido o conceito de objetos em programação, Smalltalk foi a linguagem responsável por sua popularização [29]. Em Smalltalk [19] toda a informação sobre o processo de interesse é organizada através de objetos, por mais simples ou complexa que seja a unidade de informação. Programas manipulam objetos, sendo que a definição sobre a realização do objeto, assim como de métodos invocados, ocorre apenas durante o período de execução, através do mecanismo de ligação dinâmica (ou *late binding*). A execução de um programa Smalltalk ocorre por interpretação, podendo haver uma conversão de um programa fonte para um programa em formato intermediário de interpretação, *bytecodes.*

Atualmente, as linguagens de programação orientadas a objetos de maior utilização são C++ e

Java. C++ [48], ao contrário de Smalltalk, é uma linguagem fortemente tipada, onde a maior parte das verificações de operações sobre objetos são realizadas em tempo de compilação, através do mecanismo de *early binding.* Informação em C++ pode estar representada sob a forma de objetos, representantes de entidades abstratas ou do mundo real, ou sob a forma de **literais**, representando valores simples como inteiros, números reais, caracteres e *strings.* Como C++ não é uma linguagem interpretada, sua execução é rápida e eficiente, fazendo com que C++ seja a linguagem preferencial onde o desempenho das aplicações é um aspecto importante. Críticas à linguagem referem-se principalmente à sua complexidade e à dificuldade de se realizar o transporte de aplicações em C++ entre diferentes plataformas computacionais. Além disto, C++ é uma linguagem que, devido à sua proximidade e compatibilidade com a linguagem C, favorece a implementação de programas não-orientados a objetos.

Java [20] é uma linguagem de programação definida pela Sun Microsystems como uma "simplificação" de C++, com o objetivo inicial de desenvolver aplicações de *software* para sistemas embarcados. Tendo este objetivo, a linguagem deveria ser simples, eficiente e oferecer um alto grau de portabilidade entre as possivelmente várias plataformas de execução distintas. Inicialmente, considerou-se a utilização de C++ para atingir este objetivo, mas constatou-se que essa linguagem — assim como muitas outras analisadas — não era adequada para tal fim. Assim, definiu-se uma nova linguagem de programação tomando C++ como base mas onde os seus mecanismos complexos, causadores da maior parte dos erros de programação, foram eliminados. Assim, Java não utiliza ponteiros, *templates* ou herança múltipla. Para alcançar portabilidade, empregou-se o conceito de compilar programas Java para *bytecodes* que são interpretados por uma máquina virtual [21]. A portabilidade é alcançada a partir do momento em que máquinas virtuais Java estejam disponíveis em diversas plataformas.

3.2 Incorporação de orientação a objetos a sistemas de banco de dados

A definição de Nygaard para programação orientada a objetos sugere ser este um paradigma adequado à manipulação de sistemas de informação. De fato, alguns destes conceitos estavam presentes em propostas de modelos de dados, como o agrupamento de entidades (objetos) em tipos de entidades (classes) no modelo Entidade-Relacionamento de Chen. Assim, parecia ser o caminho natural de evolução a fusão de orientação a objetos e sistemas de banco de dados. Entretanto, apesar da proximidade em seus objetivos finais, as duas abordagens de soluções — de linguagens e de modelos — estavam bem distantes naquele momento, em meados da década de 1980.

De um lado, sistemas de banco de dados atendiam às necessidades das aplicações que manipulavam dados nos mais diversos tipos de sistemas de processamento de informação. Estes sistemas ofereciam o **armazenamento persistente** dos dados, ou seja, os dados registrados no sistema tinham um período de vida que se estendia além do período de vida dos processos que os manipulavam. Em um programa orientado a objetos, variáveis tipicamente se extingüem com o final da execução do programa. Sistemas de banco de dados também eram eficientes em oferecer aramzenamento eficiente para **grandes volumes** de registros, muito maiores que a capacidade de memória principal dos

computadores.

Outro aspecto bem atendido por sistemas de banco de dados referia-se ao oferecimento de mecanismos de alto nível para a interação com usuários, tais como a definição de **linguagens de consulta** declarativas e mecanismos para a **geração de formulários** de interação.

A operação de bases de dados em ambientes distribuídos já apresentava soluções sob a forma de **sistemas de banco de dados distribuídos**. Mesmo que a solução não fosse a ideal, uma vez que os problemas de ambientes com bases de dados heterogêneas e ambientes *multidatabases* ainda são objetos de estudo [23], era muito mais do que oferecido por qualquer linguagem de programação. Outro aspecto melhor atendido por sistemas de banco de dados era a questão do suporte a múltiplos usuários em **acesso concorrente** às bases de dados.

Sistemas de banco de dados também ofereciam mecanismos interessantes de manutenção de consistência dos dados, como os mecanismos de controle de **transações atômicas**; de **recuperação** de dados em casos de falhas; e sistemas de regras *(triggers)* para a avaliação de condições durante a manipulação de dados que permitiam "disparar" procedimentos de verificação e manutenção da consistência.

Acima de tudo, a utilização de sistemas de banco de dados permitia que aplicações alcançassem a **independência de dados**, ou seja, era possível implementar aplicações sem que fosse preciso conhecer a estrutura interna de como os seus dados estavam registrados. Duas conseqüências desta propriedade eram fundamentais para a implementação de aplicações em sistemas de informação. Em primeiro lugar, mudanças na estrutura interna de armazenamento não afetavam a implementação da aplicação. Em segundo lugar, era possível compartilhar os dados entre diversas aplicações, permitindo a manutenção simplificada de um único repositório de dados para as aplicações de uma corporação.

Apesar da distância entre linguagens de programação orientadas a objetos e sistemas de banco de dados, as vantagens antevistas para a integração entre as duas abordagens era evidente. A perspectiva de se ter um modelo de objetos, onde entidades do mundo real pudessem ser representadas univocamente no esquema da base de dados por uma construção com alto nível de abstração (o objeto), parecia ser a solução ideal para o problema do *gap* semântico entre modelagens e o mundo real.

Desta forma, houve um grande investimento da parte dos pesquisadores da área de banco de dados em se desenvolver modelos integrando os conceitos de orientação a objetos com funcionalidades de bancos de dados. As motivações, assim como as necessidades, para se obter mecanismos de modelagem de objetos eram muitas, como descrito a seguir [10].

Identificadores de objetos. Em aplicações de engenharia, de projeto e em algumas aplicações comerciais, a identificação de entidades por valores de atributos pode ser artificial e sujeita a inconsistências. Por este motivo, ter uma identificação única, independente de valores de atributos e gerenciada pelo sistema, é importante para tais aplicações.

Objetos compostos. Sistemas de banco de dados relacionais não lidavam de modo adequado com o conceito de objetos formados pela agregação de outros objetos. A importância deste conceito já havia sido enfatizada pelo trabalho de Smith e Smith sobre abstração de dados, sendo essencial em sistemas de banco de dados de apoio a projetos de engenharia. Nestas aplicações, é usual a necessidade de se oferecer mecanismos para definir objetos que contêm outros objetos, assim como de oferecer operações de acesso a subcomponentes nas hierarquias que definem tais objetos.

Referências e integridade. Composição por agregação é apenas um caso especial de uma necessidade maior, que é o estabelecimento de associações entre objetos. É fundamental a capacidade de se expressar relacionamentos genéricos entre objetos, com mecanismos adequados para se utilizar esta informação. Adicionalmente, é preciso que o sistema seja capaz de manter a integridade dessas referências.

Hierarquias de tipos. O mecanismo de herança presente em linguagens de programação orientadas a objetos deve ser oferecido como um mecanismo de definição de esquemas de dados e objetos, sendo importante para a maior parte das aplicações.

Procedimentos associados. É interessante que a base de objetos possa registrar tanto a estrutura quanto o comportamento dos objetos da aplicação, onde o comportamento é expresso pelos procedimentos (métodos) que manipulam o conteúdo dos objetos. Em sistemas tradicionais, o comportamento é relevado à aplicação.

Encapsulação de objetos. Isolar a estrutura interna de objetos da aplicação é um mecanismo importante para obter reuso, inter-operabilidade e modularização das aplicações. A base de objetos deve oferecer subsídios para que este mecanismo das linguagens de programação orientadas a objetos reflita-se no sistema gerenciador da base de objetos.

Coleções ordenadas. Sistemas relacionais operam exclusivamente com o conceito de conjuntos (coleções não ordenadas), mas há diversas aplicações onde a ordem dos elementos de uma coleção é importante. Um sistema gerenciador da base de objetos deve oferecer mecanismos adicionais para a representação de coleções mais genéricas que simplesmente conjuntos.

Grandes blocos de dados. Alguns atributos de objetos podem ocupar espaços de armazenagem muito maior que o espaço usualmente ocupado pela representação de valores literais. Exemplos são registros de textos, imagens e sons. O mecanismo de BLOBs presentes am alguns sistemas relacionais estendidos, apesar de permitir o armazenamento destes volumes de dados, não oferece mecanismos para a interpretação destes dados, delegando tal tarefa exclusivamente à aplicação.

Acesso remoto. Em ambientes de projeto, a operação de computadores em redes é a regra geral e não a exceção. É importante que um sistema gerenciador da base de objetos ofereça mecanismos eficientes para acesso remoto aos objetos, com mecanismos de *checkout, locking* por períodos que podem ser extensos e *checkin* de objetos.

Evolução de esquema. Aplicações de projeto, mais que outras, necessitam facilidades para introduzir modificações no esquema de objetos ao longo de sua execução, pois nesse tipo de aplicação a definição do esquema é parte da atividade de projeto. Assim, é importante ter um sistema gerenciador da base de objetos no qual modificações de esquema possam ocorrer sem maiores impactos sobre os dados já armazenados.

Interfaces de programação. Em sistemas gerenciadores de banco de dados tradicionais, a manipulação dos dados ocorre através de uma linguagem de consulta, não procedimental, semanticamente distante da linguagem de programação da aplicação. Nas aplicações utilizando um sistema gerenciador da base de objetos, ao contrário, é interessante que haja uma maior integração entre a base de objetos e a linguagem de programação orientada a objetos usada pela aplicação, uma vez que ambas adotam o mesmo paradigma.

Múltiplas versões. Em aplicações de projeto o conceito de versões dos objetos é essencial, tanto para a manutenção da evolução histórica do projeto como para expressar alternativas concorrentes para os objetos de projeto. É importante que um sistema gerenciador da base de objetos permita expressar essa multiplicidade de representações para um mesmo objeto.

Embora tais motivações tenham sido expostas principalmente a partir das necessidades de um sistema gerenciador da base de objetos para aplicações de projeto em engenharia, claramente diversos desses mecanismos são interessantes para outras categorias de aplicação. A importância em se obter uma combinação destes novos mecanismos com os mecanismos tradicionalmente oferecidos por sistemas gerenciadores de banco de dados foi logo observada, motivando diversas linhas de pesquisa com o objetivo de analisar e propor como tal integração poderia ocorrer. Afinal, um sistema gerenciador da base de objetos incorporando tais mecanismos constitui uma ferramenta poderosa para aplicações futuras e para a evolução de aplicações tradicionais.

3.3 Protótipos pioneiros

Nesta seção serão apresentados alguns dos esforços pioneiros no desenvolvimento de sistemas gerenciadores que incorporassem, de algum modo, o conceito de orientação a objetos. Alguns dos primeiros desenvolvimentos em sistemas gerenciadores da base de objetos incluem os sistemas Iris, ORION e GemStone [25].

Iris [17] foi um protótipo desenvolvido na Hewlett-Packard para explorar as novas funcionalidades de um sistema gerenciador da base de objetos. A abordagem adotada em Iris pode ser classificada

como uma extensão de sistemas funcionais para orientação a objetos. Entre os objetivos do projeto buscava-se incorporar novos mecanismos de modelagem e de construção de tipos de dados, suporte a inferências no banco de dados, suporte a transações longas e a múltiplas versões das bases de dados.

Aplicações poderiam interagir com o sistema gerenciador da base de objetos através de várias linguagens, que teriam suas construções mapeadas para as construções internas do gerenciador de objetos. A arquitetura básica do sistema é apresentada na Fig. 3. Para o módulo gerenciador de armazenamento, um sistema gerenciador de banco de dados relacional da própria HP foi utilizado. O sistema foi implementado em C em estações de trabalho Unix (HP-9000).

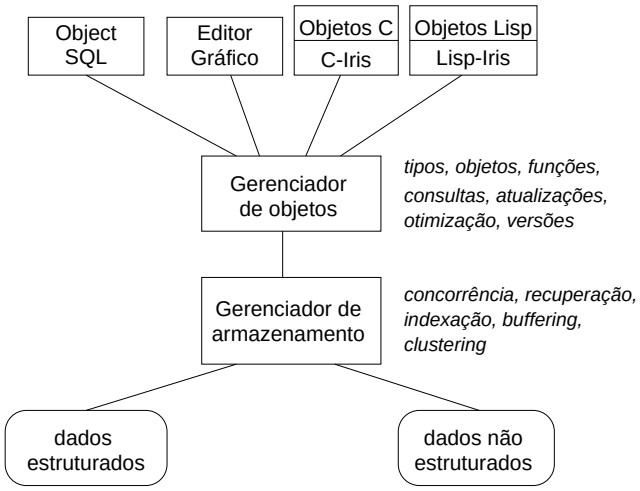

Figura 3: Arquitetura do sistema Iris.

Do ponto de vista da aplicação, três abordagens de interface foram propostas para este sistema. A primeira abordagem refletia a interface usual para sistemas gerenciadores de banco de dados de então — através de comandos de uma linguagem, OSQL, embutidos em alguma linguagem de programação. A segunda abordagem apresentava o sistema gerenciador da base de objetos como um objeto para a aplicação, que interagiria com o sistema através de métodos aplicados ao objeto. Finalmente, a terceira abordagem procurava explorar o conceito de persistência de objetos, onde objetos da aplicação seriam manipulados pelo sistema gerenciador da base de objetos de forma transparente para o programador. Um editor gráfico, implementado em Lisp, oferecia facilidades para a criação e manipulação de esquemas de objetos.

Algumas características particulares de Iris incluíam a possibilidade de examinar esquemas como quaisquer outros dados, a possibilidade de associar um objeto a mais de um tipo, e um mecanismo simples de controle de versões por *checkout* e *checkin.* Iris deu origem posteriormente a um produto da HP, OpenODB.

ORION [24] foi resultado de um projeto desenvolvido no Programa *Advanced Computer Architecture* da MCC *(Microelectronics and Computer Corporation)*, empresa de Austin, Texas. O

objetivo do projeto era inicialmente suportar as necessidades de manipulação de dados de Proteus, uma plataforma para sistemas especialistas.

Foram implementadas em ORION funções tais como controle de versões e notificação de modificações, objetos compostos e BLOBs, herança múltipla, evolução dinâmica de esquemas, gerência de transações, consultas e armazenamento de dados multimídia. Versões de objetos e referências eram suportadas através de *time stamps.*

A arquitetura do sistema é apresentada na Fig. 4. O sistema, inicialmente mono-usuário, foi implementado em Common Lisp para Symbolics (máquina Lisp) e estações de trabalho Unix. Posteriormente o sistema apresentou uma segunda versão com múltiplos clientes e uma terceira versão com múltiplos servidores.

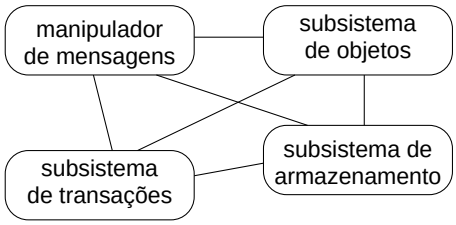

Figura 4: Arquitetura do Sistema ORION.

A interação da aplicação com ORION foi definida através de mensagens, que poderiam ser métodos definidos por usuários, para a definição e armazenamento de dados; mensagens de acesso, para a recuperação e atualização de valores de atributos de uma classe; e funções definidas pelo sistema, para realizar tarefas tais como definição de esquemas, gerência de transações, criação e remoção de instâncias.

As principais ênfases no projeto ORION foram na implementação de bases distribuídas e em evoluções de esquema [4]. Duas formas de evolução de esquema eram suportadas: mudanças na definição de classes (atributos e métodos) e mudanças na estrutura de classes (classes e associações).

O sistema ORION deu posteriormente origem a um sistema de banco de dados comercial, Itasca [3]. Itasca apresenta uma arquitetura de múltiplos clientes e múltiplos servidores, adotando formatos neutros para representação interna de dados. Assim, dados do sistema podes ser compartilhados por aplicações com interface de programação C++, C, CLOS, Lisp e Ada.

GemStone [7, 8, 14] foi um produto da Servio Logic Corporation que estendeu a linguagem Smalltalk para incorporar manipulação de bases de dados. A linguagem resultante, OPAL, implementava um modelo de dados baseado em teoria de conjuntos, STDM *(Set Theoretic Data Model).* Entre outras características, GemStone incorporava o conceito de história de objetos.

Uma visão simplificada da arquitetura do sistema GemStone é apresentada na Fig. 5. *Gem* oferecia compilação de programas em OPAL, verificava autorização de usuários, e oferecia um conjunto pré-definido de classes e métodos em OPAL para uso da aplicação. Caso atuasse no servidor, era

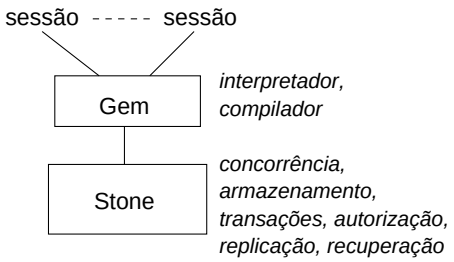

Figura 5: Arquitetura do Sistema GemStone.

também o módulo encarregado por controlar conexões com as sessões clientes, recebendo *bytecodes* para um submódulo interpretador. Opcionalmente, este módulo poderia ser alocado à máquina cliente.

Stone era o gerenciador de dados, módulo onde as funcionalidades de sistemas de banco de dados, tais como entrada e saída de disco, controle de concorrência, serviços de transação e de recuperação, eram efetivamente implementadas.

O protótipo inicial de GemStone, suportando a linguagem OPAL, foi implementado em *hardware* específico. Posteriormente, o sistema também foi oferecido com possibilidade de integração a aplicações em C++. Assim, GemStone foi o primeiro sistema a oferecer integração através de mais de uma linguagem.

4 Manipulação e implementação de gerenciadores de bases de objetos

Nesta seção são descritos os aspectos do gerenciamento de dados que são específicos para sistemas gerenciadores de base de objetos [10]. Inicialmente é apresentada uma visão conceitual dos mecanismos de manipulação de objetos, seguida por uma apresentação dos principais aspectos de implementação.

4.1 Objetos e atributos

Na orientação a objetos, objetos têm uma identificação única que não depende de valores de seus atributos. Esta identificação de objeto (OID) é utilizada para referências a objetos, tanto internas como (possivelmente) externas. Deste modo, limitações relacionadas ao processamento de chaves (primárias, "estrangeiras") em sistemas de banco de dados relacionais não estariam presentes em sistemas de banco de dados orientados a objetos.

Não obstante, o conceito de um atributo chave está presente em muitas aplicações. Deste modo, apesar de não ser um conceito da orientação a objetos, vários sistemas gerenciadores de base de objetos oferecem a definição de atributos chaves como um mecanismo de acesso adicional ao identificador de objeto, mais próximo da compreensão dos usuários e oferecendo as mesmas restrições de

integridade presentes em sistemas gerenciadores de banco de dados relacionais.

Atributos de objetos estão presentes em qualquer sistema gerenciador de banco de dados, sendo que em sistemas tradicionais usualmente tais atributos estão restritos a valores literais e atômicos. Uma diferença em sistemas gerenciadores de base de objetos está na maior flexibilidade oferecida para a definição destes atributos, que podem simples ou complexos.

Atributos simples são aqueles cujos valores são literais. Em muitas aplicações, valores multimídia são considerados atributos simples, onde o BLOB referente ao objeto multimídia é considerado um valor literal, atômico e não-estruturado. Um sistema gerenciador de banco de dados multimídia é essencialmente um sistema gerenciador de base de objetos onde os dados multimídia recebem tratamento diferenciado, não sendo considerados simples BLOBs literais.

Atributos complexos podem ser de três tipos: referências, coleções ou virtuais. Referências ou associações estabelecem relacionamentos entre objetos, tendo como valores identificadores de objetos. Um aspecto fundamental em sistemas gerenciadores de base de objetos é a manutenção da consistência entre referências a objetos.

Atributos complexos do tipo coleção são utilizados para definir conjuntos, listas ou arranjos de valores. Coleções são exemplos de tipos parametrizados presentes em sistemas gerenciadores de base de objetos; os três tipos citados são as construções parametrizáveis fundamentais, porém outras podem estar presentes em distintas implementações.

Atributos virtuais ou derivados são aqueles cujos valores não estão explicitamente armazenados na base de dados, mas ao invés são computados através de procedimentos. Em geral, tais atributos são definidos como *read-only*, sendo raro o oferecimento de mecanismos para atualizar seus valores.

4.2 Aspectos comportamentais

Um sistema gerenciador de base de objetos precisa oferecer mais que simplesmente novos mecanismos de estruturação de dados. Além da orientação a objetos estrutural, é preciso suportar a orientação a objetos comportamental [1], que pode ser obtida através da definição de métodos, procedimentos que permitem encapsular a semântica de um objeto. Em geral, atributos de um objeto são propriedade privada e exclusiva do objeto, sendo que apenas através dos métodos públicos a informação associada ao objeto pode ser manipulada. Este mecanismo de encapsulação é fundamental para atingir independência de dados em sistemas gerenciadores de base de objetos.

A definição estrita de encapsulação exige a satisfação de três critérios:

1. Apenas métodos podem ser públicos;
2. Métodos são definidos em uma linguagem procedimental;
3. Métodos podem manipular apenas dados do próprio objeto.

Esta definição é satisfeita em Smalltalk. Há no entanto diversas outras linguagens de programação orientadas a objetos e sistemas gerenciadores de base de objetos que relaxam uma ou mais destas

restrições para oferecer um melhor desempenho de execução às custas de se aceitar um grau menor de encapsulação.

A representação de comportamento em sistemas gerenciadores de base de objetos permite definir dados ativos, que incluem os atributos derivados, regras e agentes. Atributos derivados já foram apresentados anteriormente. Regras ou *triggers* são definidos como pares padrão-ação, onde as ações são ativadas quando modificações à base de dados tornam verdadeiro o predicado expresso no padrão da regra. Restrições de integridade são casos particulares de regras onde as ações estão limitadas a sinalizações de condições de erro. Agentes são caracterizados por processos associados ao sistema gerenciador de base de objetos, porém independentes, podendo realizar ações que extrapolam o ambiente da base de dados — por exemplo, enviar uma mensagem por correio eletrônico.

4.3 Tipos e herança

Objetos com estrutura similar são agrupados em classes, sendo que o tipo do objeto é definido através de uma declaração de classe. Usualmente, tipos são criados através de uma linguagem de definição de dados ou, mais especificamente em sistemas gerenciadores de base de objetos, através de uma linguagem de definição de objetos. Uma diferença entre uma linguagem de definição de dados e uma linguagem de definição de objetos é que esta última pode permitir, como já observado, a definição de procedimentos associados à estrutura, como no caso de atributos virtuais, incorporando assim aspectos de linguagem de manipulação de dados.

Um conceito avançado que pode estar presente um uma linguagem de definição de objetos é o conceito de herança, que permite definir novos tipos de objetos a partir de tipos já existentes. Herança é um mecanismo de implementação do conceito de generalização, importante para bancos de dados como ressaltado no trabalho de Smith e Smith sobre abstração de dados. Os novos subtipos podem ser diferenciados dos tipos dos quais foram derivados por mecanismos tais como especialização, com o acréscimo de novos atributos, ou sobrecarga *(overloading)*, onde oferece-se uma implementação diferenciada para um método já existente no tipo ancestral. Deste modo, o esquema de tipos de objetos define uma árvore ou hierarquia de tipos relacionados.

Herança múltipla refere-se ao mecanismo de herança através do qual um novo tipo de objeto pode ser derivado a partir de mais de um tipo ancestral. Entretanto, a semântica de herança múltipla nem sempre é clara. Ademais, a implementação de herança múltipla requer cuidados adicionais para lidar com situações como o conflito de resolução para atributos ou métodos. Quando herança múltipla é suportada, o esquema de tipos de objetos passa a ser organizado como uma malha ou grafo de tipos.

Em um sistema gerenciador de base de objetos, a modificação de um tipo implica na modificação do esquema de dados. Em um sistema gerenciador de banco de dados tradicional, modificações no esquema eram ações que afetavam extremamente as aplicações, muitas vezes necessitando de processamento *off-line* para converter os dados de um esquema para o outro. Em um sistema gerenciador de base de objetos, esta evolução de esquemas deve ocorrer de forma mais natural. São três as categorias de modificações que implicam em evolução do esquema: mudanças em componentes (atributos

ou métodos) de um tipo, mudanças na hierarquia (ou malha) de tipos, ou mudanças na definição de tipos, através de modificação de nome, inclusão ou remoção de tipos.

4.4 Consultas

Um dos principais aspectos a se considerar na manipulação de bases de dados é o mecanismo oferecido para expressar operações sobre os dados. Em geral, aplicações usando bases de dados são desenvolvidas em alguma linguagem de programação e integram expressões de manipulação de dados aos programas. A diferença entre estas linguagens de programação e de manipulação de dados dá origem ao problema de "descasamento de impedâncias" *(impedance mismatch)* entre as linguagens, que é endereçado de maneiras distintas em diversos sistemas gerenciadores de base de objetos. Entretanto, nenhuma destas abordagens, descritas a seguir, consegue eliminar completamente o problema do descasamento de impedâncias entre as aplicações.

Uma estratégia de integração de linguagens é o *query download*, onde todos os objetos relevantes para a operação da aplicação são recuperados da base de dados, traduzidos para a representação adequada para a linguagem de programação, manipulados, re-traduzidos e armazenados de volta à base de dados. Uma variante deste esquema limita a operação a um único objeto por interação, com o objetivo de maximizar a concorrência de acesso à base de dados. Outra alternativa ainda é armazenar objetos da linguagem de programação como BLOBs na base de dados, que nesta situação não conteria nenhuma informação sobre a estrutura do objeto.

Em outra direção de integração, alguns sistemas estendem linguagens de manipulação de dados com construções procedimentais. Sybase adotou esta abordagem ao estender SQL com construções de controle de execução para sistemas relacionais. Outros sistemas oferecem novos tipos de dados na linguagem de programação para refletir tipos e operações referentes à base de dados.

Uma abordagem mais transparente de integração entre linguagens de programação orientadas a objetos e sistemas gerenciadores de base de objetos é a incorporação de mecanismos de persistência de objetos à linguagem de programação. Nesta situação, a base de dados é utilizada de forma implícita e transparente como meio de manutenção do estado de objetos da aplicação entre sessões.

Métodos de especificação de persistência em sistemas gerenciadores de base de objetos podem ser por tipo, por invocação explícita ou por referência. Na especificação por tipo, apenas alguns tipos pré-definidos são persistentes. Uma crítica a este modelo é que persistência e especificação de tipos são conceitos ortogonais, e que associar persistência a apenas alguns tipos fere essa ortogonalidade. Na especificação de persistência por invocação explícita, a aplicação solicita explicitamente ao sistema gerenciador de base de objetos o armazenamento persistente do objeto em questão. Na especificação por referência, um ou mais objetos (as raízes) são declarados como persistentes, e todos os objetos por eles referenciados tornam-se automaticamente persistentes. Este é o mecanismo mais transparente, mas que exige mais em termos de processamento da parte do sistema gerenciador de base de objetos.

Outro aspecto importante relativo a consultas em sistemas gerenciadores de base de objetos refere-se ao tipo de resultado que uma consulta gera. Em sistemas relacionais, existe um único tipo

(a relação); portanto, operandos e resultados de consultas são relações. Em sistemas gerenciadores de base de objetos, o grau de liberdade é maior. Alguns sistemas podem limitar a resposta obtida a alguns tipos específicos, embora a tendência seja permitir a manipulação de resultados abertos — literal, objeto, coleções de objetos ou até mesmo relações (arranjos tabulares de dados).

Consultas envolvendo múltiplos objetos necessitam de alguma forma de mecanismo de junção. Duas variantes da operação de junção em sistemas gerenciadores de base de objetos são a junção por referência e a junção por valor. A junção por referência utiliza associações definidas na base de objetos para obter objetos associados. A junção por valor, assim como a junção de sistemas relacionais, é realizada sobre valores de atributos que compartilham um domínio comum, não necessitando portanto de associações explícitas para sua realização.

4.5 Concorrência e recuperação

Como dados em uma base de dados são persistentes e usualmente compartilhados por várias aplicações e/ou usuários, é preciso que haja mecanismos para controle de acesso concorrente, de recuperação de dados em caso de operações falhas e de manutenção de integridade dos dados. Em um sistema gerenciador de base de objetos, diferentemente do que ocorre em sistemas tradicionais, uma aplicação manipula de forma natural dados transientes e dados persistentes, o que pode criar dificuldades adicionais na manutenção de consistência dos dados.

Operações sobre objetos persistentes são usualmente definidas no escopo de transações, assim como ocorre em sistemas gerenciadores de banco de dados tradicionais. Uma possível diferença é que sistemas gerenciadores de base de objetos podem (e geralmente o fazem) incluir transações longas e, em alguns casos, transações aninhadas.

Versões oferecem um mecanismo alternativo para controle de concorrência. Através do mecanismo de versões libera-se o sistema de manter uma única cópia de um objeto na base, o que alivia uma eventual degradação que poderia ocorrer com a incorporação de transações (e consequentemente *locks*) longas. Com versões, evoluções alternativas nos estados dos objetos são permitidas. Um conceito relacionado a este é configuração, um conjunto de versões de objetos mutuamente consistente. Configurações equivalem a *snapshots* (ou visões instantâneas) consistentes da base de objetos.

4.6 Implementação de identificadores

Um dos fatores mais importantes no desempenho de um sistema gerenciador de base de objetos é a forma de implementação da representação de seus objetos — como são armazenados, identificados e recuperados da base de dados.

A representação de identificadores de objetos pode ser física ou lógica. A representação física contém o endereço real do objeto, como ocorre em linguagens de programação orientadas a objetos onde objetos são identificados por sua posição na memória. A representação lógica envolve algum mecanismo de mapeamento entre uma representação interna e o endereço do objeto.

As quatro abordagens básicas de implementação de identificadores são utilização simples do endereço, um identificador físico; utilização de endereços estruturados, que contém uma componente lógica e outra física; utilização de *surrogates*, identificadores lógicos que são mapeados para endereços físicos através de um índice ou de uma tabela; e a utilização de *surrogates* tipados, onde a informação sobre o tipo do objeto é incluída no identificador lógico.

A unicidade de um identificador de objeto é um aspecto fundamental para a manutenção da consistência de uma base de objetos. Identificadores de objetos são utilizados tanto pelas aplicações como pela própria base de objetos, para representar associações entre objetos. O mecanismo para garantir unicidade pode variar entre implementações distintas. Alguns sistemas garantem que identificadores de objetos removidos não são reaproveitados; outros buscam garantir unicidade entre múltiplas bases de objetos interligadas em rede, incorporando o endereço da máquina servidora no identificador de objeto.

Identificadores de objetos puramente físicos, por endereços, oferecem o melhor potencial de desempenho para recuperação de objetos. Porém, fixar a posição do objeto na base torna-a menos flexível, dificultando operações como relocação de objetos. A relocação pode ser necessário para ampliar o espaço de armazenamento associado a um objeto ou realizar uma compactação ou *reclustering* da base de objetos. Por este motivo, endereços físicos não são utilizados na prática. ONTOS e Objectivity/DB utilizam endereços estruturados, GemStone e POSTGRES utilizam *surrogates*, e ORION e Itasca adotam *surrogates* tipados.

Além da referência a objetos por identificadores de objetos, diversos sistemas oferecem a possibilidade de se localizar um objeto através de atributos chaves. Em geral, a implementação deste mecanismo em sistemas gerenciadores de base de objetos é similar àquela de sistemas tradicionais, onde valores de atributos chaves são mapeados a identificadores de objetos através de índices *hash* ou *B-trees*.

4.7 Swizzling

Swizzling é o processo de conversão entre a representação persistente do objeto, na base de objetos, e sua representação dinâmica, na linguagem de programação orientada a objetos adotada pela aplicação, onde o objeto será efetivamente manipulado. Evidentemente, é um processo importante na manipulação de objetos, sendo determinante para o desempenho do sistema gerenciador de base de objetos.

A conversão que deve ocorrer através de *swizzling* envolve tipicamente traduções de referências e, eventualmente, modificações na estrutura e no formato de representação de dados. Referências dinâmicas são tipicamente ponteiros na linguagem de programação, e não identificadores de objetos. A tradução de identificadores de objetos para referências a objetos na linguagem de programação é a principal atividade desempenhada durante o *swizzling.* Mudanças na estrutura podem ocorrer através da incorporação de cabeçalhos e campos para utilização em tempo de execução. Mudanças na representação interna dos dados podem ser necessárias principalmente quando a base de objetos é

compartilhada em ambientes heterogêneos.

Swizzling pode ocorrer em momentos distintos, de acordo com a implementação adotada para este mecanismo. Tipicamente, uma das quatro abordagens a seguir é adotada:

1. A conversão ocorre no momento em que o objeto é transferido da base de objetos para o espaço de memória da aplicação *(checkout)*.

2. A conversão ocorre no momento em que o objeto, já transferido para a memória, é referenciado pela primeira vez pela aplicação.

3. A conversão ocorre no momento em que a aplicação a solicita explicitamente.

4. A conversão ocorre quando não é possível reaproveitar um endereço físico designado anteriormente, o qual é registrado também na base de dados.

Esta última abordagem é adotada em ObjectStore, que busca explorar a grande capacidade de endereçamento virtual dos processadores modernos para alocar um objeto sempre à mesma posição de memória virtual.

4.8 Granularidade de transferência

Bases de objetos podem transferir informação entre o disco e a memória em unidades de dimensão, ou granularidades, distintas. Tipicamente, as granularidades adotadas são páginas, objetos ou resultados de consultas.

A abordagem mais utilizada nas implementações de sistemas gerenciadores de base de objetos é passar para a aplicação uma imagem do disco, adotando granularidade de uma página (Fig. 6). Esta abordagem simplifica o trabalho do servidor de páginas, onde o processamento será mínimo. Entretanto, o tráfego de dados pela rede é ampliado, pois uma página pode conter mais informação que a solicitada pela aplicação.

Quando a granularidade adotada é um objeto ou um resultado de consulta, o tráfego de dados pela rede é reduzido, pois garante-se que a aplicação estará recebendo apenas o que foi solicitado. Entretanto, exige-se nessas situações que o servidor de objetos já realize algum processamento sobre a base de objetos. Se o sistema gerenciador de base de objetos for compartilhado por muitas aplicações, um alto número de requisições simultâneas pode ter impacto significativo no desempenho do servidor.

A abordagem de imagem do disco, embora eficiente, pode apresentar riscos à integridade dos dados presentes na página. Este risco está presente principalmente quando a aplicação utiliza uma linguagem de programação onde é permitido aos programas examinar e alterar qualquer posição de seu espaço de memória — como ocorre em C++, por exemplo. Em tais situações, seria necessário que o sistema gerenciador de base de objetos impusesse mecanismos adicionais de manutenção da integridade dos dados em uma página manipulada pela aplicação, evitando sua degradação através de

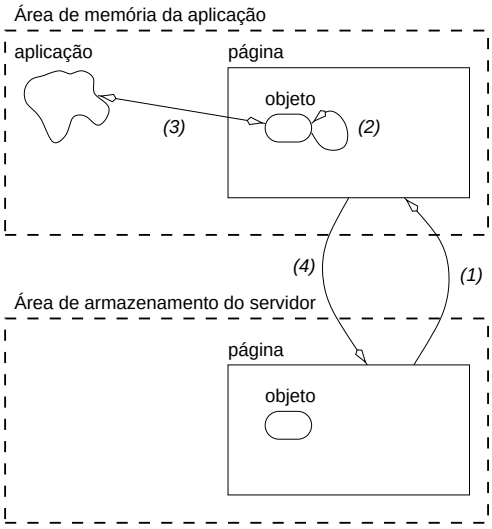

Figura 6: Abordagem de imagem do disco com servidor de páginas: *(1)* transferência de página *(checkout)* do servidor para cliente; *(2) swizzling*; *(3)* manipulação do objeto pela aplicação; *(4)* reconversão para formato do disco e transferência de página *(checkin)* do cliente para o servidor.

ações inadvertidas ou maliciosas. Entretanto, as implementações correntes de sistemas gerenciadores de base de objetos não incorporam mecanismos com este fim.

Citou-se como uma das atividades durante o *swizzling* a incorporação de informação para manipulação de objetos em tempo de execução. Informação típica de *run time* para um objeto inclui o mapeamento de um identificador de objeto para endereço físico, o registro de quais objetos estão na memória da aplicação, manutenção de *dirty bits* para indicar se objetos foram atualizados ou apenas consultados, e contadores de referências para sustentar atividades internas de *swapping* e *garbage collection.* Abordagens para manipulação deste tipo de informação incluem a manutenção de tabelas em memória, utilização de descritores de objetos e incorporação de *status bits* no próprio objeto.

4.9 Atributos

A forma de implementação mais simples para a representação de atributos de um objeto seria utilizar relações, com campos de dimensão fixa, como foi proposto por alguns protótipos iniciais de sistemas gerenciadores de base de objetos. Entretanto, esta alternativa limita severamente a flexibilidade de representação para objetos, não permitindo operar de modo eficiente com atributos de dimensões variáveis, com coleções com quantidade variável de elementos, a evolução de esquema pela inclusão de atributos e o armazenamento eficiente para atributos não utilizados por um objeto.

Listas de propriedades oferecem um mecanismo mais versátil de implementação de atributos de um objeto. Uma lista de propriedades é composta por elementos contendo três campos, representando um identificador de atributo, o tamanho do valor do atributo e o valor. Este mecanismo é flexível,

porém demanda um maior grau de processamento da parte dos métodos que manipulam a estrutura interna do objeto.

Herança é outro aspecto que traz impacto na representação de atributos. Em sistemas onde apenas herança simples é permitida, o mecanismo de lista de propriedades é ainda adequado. Neste caso, a lista de propriedades de um objeto é composta pela concatenação dos atributos do tipo do objeto com os atributos do tipo de objeto ancestral, e assim recursivamente até encontrar os atributos do tipo de objeto que é a raiz da hierarquia. Para lidar com herança múltipla, mecanismos mais elaborados são necessários para estabelecer priridades na concatenação e resolver conflitos entre tipos ancestrais com atributos de mesmo nome.

Manipulação de coleções é outro aspecto crítico em sistemas gerenciadores de base de objetos, sendo esse mecanismo amplamente utilizado na descrição de estrutura de atributos de objetos e no retorno de consultas. Como coleções podem ser demasiadamente extensas e/ou conter objetos grandes, é preciso oferecer mecanismos que permitam manipulá-las de modo mais eficiente. O principal mecanismo adotado é a utilização de objetos iteradores. Um iterador é um objeto temporário que contém referências aos membros da coleção, oferecendo métodos tais como produzir o próximo elemento da coleção (lista) ou buscar o i-ésimo elemento da coleção (arranjo).

A manipulação de atributos do tipo BLOB é usualmente implementada através de interfaces *stream*. Assim como iteradores, um *stream* permite referenciar objetos sem que esses estejam completamente concretizados em memória, com métodos para posicionar, ler e escrever blocos de dados binários. Alternativas para a manipulação de BLOBs incluem a paginação por demanda e utilização de *piece streams*. Na paginação por demanda, a aplicação tem o objeto completo em memória virtual, sendo que páginas do objeto são transferidas para a aplicação à medida que seus *bytes* são referenciados. Um *piece stream* é um mecanismo de *stream* com facilidades adicionais que permitem a atualização de conteúdo do BLOB, tais como inserção e remoção de *bytes* na posição corrente.

4.10 Associações

A implementação de relacionamentos não precisa refletir diretamente a forma como a aplicação os define conceitualmente. A estratégia de implementação pode depender da multiplicidade das associações. Por exemplo, um relacionamento um-para-muitos pode ser representado por contiguidade física, por indexação, por ligação ou através de objetos intermediários criados pelo sistema gerenciador de base de objetos.

Um tipo de associação implícita entre os elementos de uma coleção é a ordenação. Apesar de estar presente em muitas aplicações, nem sempre este mecanismo é suportado em sistemas gerenciadores de base de objetos, onde apenas coleções do tipo conjunto (assim como relações, sem critérios de ordenação entre elementos) são suportadas. A importância da ordenação da coleção é reconhecida de longa data; o modelo CODASYL definia associações de ordenação manual (critério de ordenação estabelecido pela aplicação) e automática (estabelecido pelo sistema sobre algum atributo da coleção).

Outro aspecto importante na implementação de associações é a manutenção da integridade referencial. O mecanismo mais amplamente adotado para este fim é a utilização de pares de atributos inversos. Assim, se a aplicação define uma referência de um objeto a para um objeto b, denotada $a \rightarrow b$, o sistema gerenciador de base de objetos cria internamente a referência $b \rightarrow a$. Alternativamente, o sistema pode implementar um contador de referências para o objeto, só permitindo sua remoção quando o valor deste contador for nulo.

4.11 Clustering

A incorporação do conceito de objeto complexo em sistemas gerenciadores de base de objetos introduz grande flexibilidade de estruturação dos dados. Por outro lado, a recuperação de objetos complexos pode introduzir degradação no desempenho dos sistemas. Afinal, o conteúdo de um objeto complexo é composto por outros objetos, que potencialmente podem estar distribuídos em diversas páginas espalhadas pelo disco. Assim, recuperar o conteúdo de um objeto complexo pode acarretar em várias solicitações de páginas do servidor para o cliente.

Clustering é um mecanismo que pode ser adotado para minimizar esta degradação no desempenho. Simplesmente colocado, o conceito de *clustering* é adotar alguma estratégia de alocação de objetos no sistema de armazenamento de modo a minimizar a quantidade de páginas necessárias para manipular objetos. As estratégias de *clustering* mais utilizadas são:

Objetos complexos: adota o relacionamento de agregação para identificar objetos que devem ser armazenados próximos — preferencialmente, na mesma página ou em páginas contígüas.

Referências: adota a existência de relacionamentos quaisquer entre objetos para direcionar a proximidade de armazenamento. A estratégia de *clustering* por objetos complexos é um caso especial desta estratégia.

Tipos de objetos: adota o critério de aproximar o armazenamento de objetos de uma mesma classe. Este é um critério equivalente ao adotado pela maior parte dos sistemas gerenciadores de banco de dados relacionais, onde tuplas de uma mesma relação são armazenadas contigüamente.

Índices: adota o valor de um atributo como critério de ordenação dos objetos, e aproxima o armazenamento de objetos com valores próximos neste atributo.

Definição do usuário: utiliza um indicativo da aplicação, em tempo de execução, para buscar a melhor posição de armazenamento de um objeto. Um exemplo de implementação utiliza um identificador de um objeto existente como parâmetro na criação de um novo objeto, indicando que os dois devem ser armazenados em posições próximas.

A seleção de um critério de *clustering* deve depender da forma de recuperação mais adotada para o objeto na aplicação. Por exemplo, em sistemas CAD provavelmente a estratégia de objetos

complexos seria a preferencial, enquanto que para uma aplicação comercial, envolvendo seqüências de recuperações em ordem alfabética, a estratégia de índices seria a mais adequada.

4.12 Evolução de esquemas

Em grande parte dos sistemas gerenciadores de base de objetos, o esquema de dados é em si armazenado como um objeto. Entretanto, este objeto nem sempre é diretamente manipulável pela aplicação em tempo de execução; restrições de acesso são impostas como forma de manutenção da integridade da base de objetos. Mudanças no esquema devem obedecer a tais restrições, determinando como esquemas podem evoluir.

Há quatro estratégias básicas que determinam como um esquema pode evoluir: tipo de uma escrita, atualização imediata, atualização posterior, e mapeamento de esquema. Tais estratégias são brevemente apresentadas a seguir.

A estratégia de tipos de uma escrita (ou tipos *write-once*) é a abordagem mais simples, que determina que qualquer tipo que já contenha pelo menos uma instância não pode ser modificado. Modificações no tipo, tal como a inclusão de um novo atributo, deve ser explicitamente desempenhada pela aplicação através da criação de um novo tipo e cópia do conteúdo de objetos já existentes para objetos do novo tipo.

A estratégia de atualização imediata é a mais adotada em sistemas gerenciadores de base de objetos correntes. Nesta estratégia, qualquer modificação no tipo de um objeto é imediatamente refletida para os objetos. Assim, a inclusão de um novo atributo em um tipo faria com que todos os objetos daquele tipo tivessem sua estrutura atualizada oara conter um campo adicional contendo o valor nulo.

Na estratégia de atualização posterior, a estrutura de um objeto cujo tipo tenha sido modificado não é alterada a não ser no primeiro momento em que o objeto é acessado após a alteração do tipo. Nesta abordagem, é preciso que o sistema gerenciador de base de objetos mantenha informação histórica sobre a evolução de esquemas e sobre procedimentos de conversão para poder manter a consistência da base.

A estratégia de mapeamento de esquema equivale a uma estratégia de atualização posterior onde o momento de atualização é postergado indefinidamente, até o momento em que uma reorganização da base de objetos seja explicitamente solicitada. Nesta estratégia é preciso manter os mapas de conversão entre todas as possíveis versões de esquema, o que possibilitaria que usuários pudessem ter visões de um mesmo objeto sob diferentes versões de esquema. Entretanto, a implementação de tal estratégia seria extremamente complexa, não tendo sido implementada em nenhum sistema gerenciador de base de objetos até o momento.

5 Exemplos

Nesta seção serão descritos alguns exemplos de sistemas gerenciadores de base de objetos. Deve-se observar, entretanto, que em se tratando de sistemas comerciais geralmente não é possível obter informação detalhada sobre os mecanismos de implementação do sistema. Por este motivo, apresenta-se inicialmente um protótipo sobre o qual informação sobre a implementação é bem conhecida.

5.1 UniCOSMOS

UniCOSMOS *(University of Campinas Object Storage Management System)* foi um protótipo desenvolvido na UNICAMP no período 1985–1989 para aplicações em engenharia [28, 39]. Efetivamente, UniCOSMOS não era um sistema gerenciador de base de objetos, mas sim um núcleo para tais sistemas onde alguns dos mecanismos básicos de manipulação de objetos persistentes poderiam ser avaliados.

A arquitetura de UniCOSMOS é apresentada na Fig. 7. A parte superior da figura representa módulos relativos ao esquema de objetos, enquanto que a parte inferior representa módulos que manipulam instâncias (ou seja, objetos).

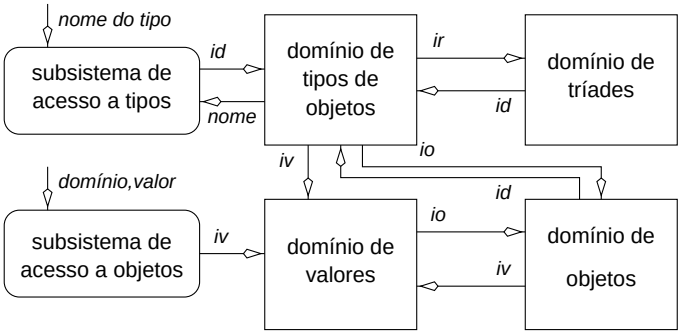

Figura 7: Arquitetura de UniCOSMOS.

Neste sistema, tipos de objetos são identificados por um nome. O subsistema de acesso a tipos mantém uma tabela *hash* que associa os nomes de objetos a *surrogates* tipados (`id`). Um mecanismo de sinônimos é suportado, sendo possível associar mais de um nome para um mesmo tipo de objeto.

O domínio de tipos de objetos registra a informação sobre o esquema da base de dados. Para cada tipo de objeto, cinco listas internas mantêm informação sobre:

1. os nomes do tipo de objeto;
2. os atributos do tipo de objeto;
3. as associações com outros tipos de objeto;

4. os tipos de objetos componentes, caso o objeto seja complexo;

5. os tipos de objetos que contêm este tipo como componente, caso o objeto faça parte de um objeto complexo.

Associações são representadas internamente por tríades, ou associações binárias. A ligação entre dois tipos de objetos é estabelecida por um terceiro tipo de objeto, criado internamente. Assim, uma referência $a \xrightarrow{r} b$ é mapeada internamente para uma tríade $r \rightarrow a \rightarrow b$. Para facilitar a manutenção da integridade referencial, o domínio de tríades armazena todas as permutações de cada associação. Assim, também são armazenadas neste domínio as tríades $a \rightarrow b \rightarrow r$ e $b \rightarrow r \rightarrow a$. Cada tríade registrada neste domínio é identificada por um *surrogate* `ir`.

No nível de instâncias, objetos são representados através de listas de propriedades armazenadas no domínio de objetos. A flexibilidade das listas de propriedades permite manipular adequadamente atributos com valores nulos, multivalorados e de dimensão variável. Cada lista de propriedades é identificada por um *surrogate* de objeto (`io`), e cada elemento da lista é formado por um identificador de atributo (referente ao identificador do tipo de objeto, `id`) e por um identificador de valor (`iv`). Os valores são armazenados de fato no domínio de valores, onde cada domínio (no sentido matemático da palavra, o conjunto de elementos dos quais os valores de atributos são tomados) é composto por uma lista de propriedades para valores. Cada elemento desta lista, identificado por um *surrogate* `iv`, contém a dimensão e a representação interna do valor.

O subsistema de acesso a objetos tem por objetivo apoiar atividades de consultas por valor à base de objetos. Neste módulo são implementados os mecanismos para agilizar acesso à informação, tais como estruturas de índices. Como domínios podem ser compartilhados por vários tipos de objeto, as respostas a consultas são organizadas na forma de listas de *surrogates* `io`. Para restringir a busca a um único tipo de objeto, é preciso combinar este resultado com o resultado de uma busca no nível de esquema, através do nome do objeto. Para agilizar a realização destas combinações, diversos métodos de manipulação de *surrogates* são oferecidos às aplicações de UniCOSMOS.

O sistema foi implementado em linguagem C, em estações de trabalho Unix. Desenvolvimentos relacionados a UniCOSMOS incluíram o tratamento de transações convencionais e longas [35], tratamento de regras e de restrições [30], manipulação de herança [43] e a implementação de um gerenciador de dados para projetos de engenharia manipulando esquemas entidade-relacionamento estendidos com mecanismos para objetos complexos e evolução de esquemas [40].

5.2 O_2

O desenvolvimento do Sistema O_2 [15] iniciou-se como um protótipo de pesquisa, posteriormente convertido para um produto comercial da empresa francesa O_2 Technology. O sistema foi desenvolvido em C++ para plataformas Unix.

O principal módulo do sistema é o EngenhoO_2, estruturado em três camadas:

Gerenciador de esquema, responsável por monitorar criação, recuperação, atualização e remoção de classes, métodos e nomes globais;

Gerenciador de objetos, camada intermediária que fornece identificação de objetos e implementa estratégias de indexação e *clustering*;

Gerenciador de disco, responsável por implementar as estruturas de arquivos (paginados) que armazenam fisicamente os dados dos objetos do sistema.

As camadas mais elevadas do sistema são organizadas na forma de uma aplicação cliente, que pode executar em uma máquina conectada ao servidor ba base de dados, onde a aplicação gerenciador de disco atua como um servidor de páginas.

Identificadores de objetos são registrados internamente sob a forma de endereços estruturados, com campos para identificação de um volume de dados (2 bytes), combinado com uma identifcação de página no volume (4 bytes) e uma posição *(slot)* na página (2 bytes). Para a aplicação, descritores de objetos são utilizados.

Objetos em O_2 podem ser atômicos (inteiros, reais, booleanos, *strings* e bits) ou complexos, criados a partir de construtores de tuplas, conjuntos, *bags* e arranjos. Persistência é definida por referência; objetos que são derivados de uma raiz persistente, direta ou indiretamente, são persistentes.

A interação de uma aplicação com o sistema pode ocorrer através de interfaces de programação em C ou C++, através de uma linguagem declarativa (O_2 C) ou através de O_2 SQL, uma extensão de SQL para lidar com objetos complexos.

5.3 ObjectStore

ObjectStore [32] é um sistema comercial de banco de dados orientado a objetos desenvolvido pela Object Design Inc., de Burlington, Massachusetts. Um dos objetivos iniciais no desenvolvimento deste sistema era fazer de C++ uma linguagem de programação de banco de dados. Por este motivo, ObjectStore é um dos sistemas mais próximos desta linguagem em termos semânticos.

ObjectStore oferece suporte para identificação de objetos, atributos inversos, herança múltipla, acesso paginado a BLOBs e aplicações interativas para visualização de conteúdo e criação de esquemas. Manipulação de coleções é oferecida através de bibliotecas de tipos de coleções, que inclui conjuntos, *bags*, listas, arranjos e dicionários. *Bags* são conjuntos que permitem duplicação de elementos. Dicionários são *bags* onde a cada elemento é associada uma chave. Um tipo genérico, coleção, também é oferecido. A Fig. 8 apresenta uma árvore de decisão associada à definição da melhor forma de coleção para uma aplicação [33].

A interação entre ObjectStore e a aplicação pode ocorrer através de uma linguagem de manipulação de dados ou através de interfaces de programação em C, em C++ e, desde a versão 5.0, em Java. O mecanismo de mapeamento de objetos para o espaço de memória virtual torna a persistência natural em ObjectStore, onde qualquer dado pode se tornar persistente. Assim, o mecanismo

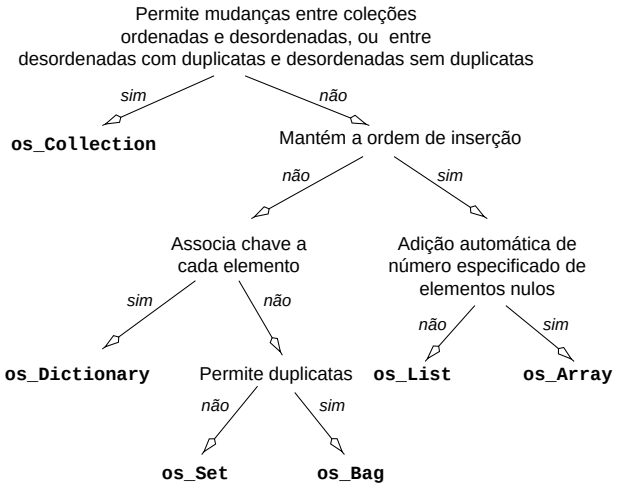

Figura 8: Árvore de decisão para tipos complexos em Object Store.

de persistência é ortogonal à hierarquia de tipos. Um aspecto negativo nesta estratégia é que este mecanismo exige a manutenção de mecanismos de pré-processamento, dificultando a portabilidade do sistema entre plataformas distintas. Por outro lado, uma vez que objetos sejam transferidos para a memória principal após um *page fault*, sua manipulação é tão rápida como a de qualquer objeto não persistente da linguagem de programação.

Aplicações de ObjectStore utilizam dois processos para sua execução. O servidor de páginas é o ObjectStore *Server*, que está executando permanentemente em uma máquina onde os dados estão armazenados. O lado cliente de ObjectStore é implementado através de um *Cache Manager*, que é ativado automaticamente pela aplicação (Fig. 9). Um cliente pode acessar informação de diversos servidores, caracterizando ObjectStore como tendo uma arquitetura múltiplos clientes e múltiplos servidores.

6 Esforços correntes e direções futuras

Nesta seção final é feito um breve apanhado de esforços correntes que poderão afetar o desenvolvimento futuro de sistemas gerenciadores de base de objetos, com ênfase nos aspectos de padronização e futuras aplicações.

6.1 ODMG

ODMG *(Object Database Management Group)* é um consórcio, formado em 1991 por diversos fabricantes de sistemas gerenciadores de base de objetos, que tem por objetivo criar um padrão para bancos de dados orientados a objetos que permita ampliar o grau de portabilidade das aplicações

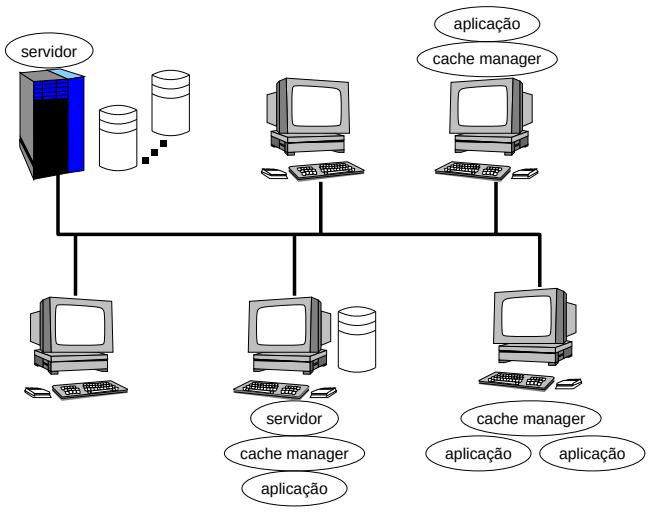

Figura 9: Processos na arquitetura de ObjectStore.

desenvolvidas para os seus diferentes sistemas [9]. Assim como SQL permite oferecer para uma aplicação uma visão comum de bases de dados relacionais, independente do fabricante, busca-se com ODMG atingir um grau de padronização similar para sistemas gerenciadores de base de objetos.

Os participantes do consórcio ODMG incluem Object Design, ONTOS, O_2 Technology, Versant, Objectivity, Digital Equipment, Hewlett-Packard, Itasca, Intellitic, Poet Software, Servio e Texas Instruments. Pelo menos estas empresas concordaram em tornar seus produtos compatíveis com o padrão ODMG para sistemas gerenciadores de base de objetos. Deve ser enfatizado que não há uma padronização da implementação, ou seja, os produtos de cada empresa não são idênticos; apenas oferecem uma interface comum para as aplicações.

O foco deste padrão foi na integração de sistemas gerenciadores de base de objetos a linguagens de programação. Entretanto, para poder definir interfaces padronizadas entre sistemas gerenciadores de base de objetos e linguagens de programação era necessário inicialmente ter uma visão comum de como um sistema gerenciador de base de objetos deveria estar estruturado e de quais seriam os seus componentes. O resultado deste primeiro esforço foi a definição de uma arquitetura comum, apresentada na Fig. 10.

O padrão ODMG está organizado em duas componentes principais, *framework* e *bindings.* Em *framework* definem-se aspectos comuns a todas as linguagens de programação, tais como a arquitetura, o modelo de objetos, a linguagem de declaração (ODL) e uma linguagem declarativa de consulta (OQL). O conjunto de *bindings*, estabelece o mapeamento das construções do padrão para aquelas de linguagens de programação. Atualmente, C++, Smalltalk e Java são contempladas no padrão.

O modelo de objetos de ODMG sumariza diversos conceitos suportados por sistemas gerenciadores de base de objetos. Ele oferece objetos e literais como primitivas de modelamento. Literais

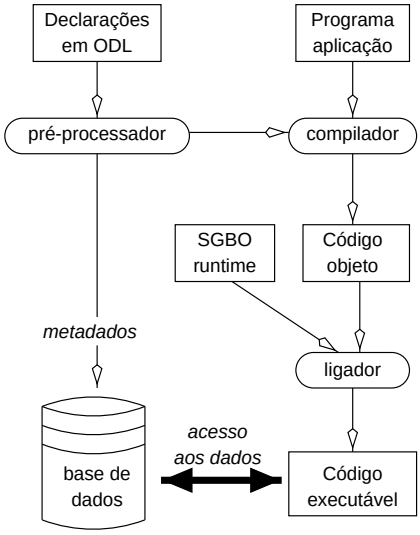

Figura 10: Arquitetura de um sistema gerenciador de base de objetos no padrão ODMG.

podem ser atômicos (Long, Float, Octet,...), coleção (Set, Bag, List e Array) ou estruturados (Date, Time, Timestamp,...). Objetos podem ser atômicos ou coleção. O estado de um objeto é definido por suas propriedades, que podem ser atributos ou relacionamentos. O comportamento do objeto é definido pelo conjunto de operações que podem ser executadas sobre o objeto. Objetos e literais são categorizados por seus tipos; elementos de um mesmo tipo têm estrutura e comportamento comuns. Algumas vezes, um objeto é denominado uma instância de um tipo. Um banco de dados armazena objetos, permitindo que estes sejam compartilhados por múltiplos usuários e aplicações. Um banco de dados é baseado em um esquema que é definido em uma linguagem de definição de objetos e contém instâncias de tipos definidos por seu esquema.

6.2 OMG

OMG *(Object Management Group)* [34] é um consórcio industrial, formado em 1989, cujo objetivo é definir padrões para permitir inter-operabilidade em sistemas distribuídos heterogêneos, adotando para tal o paradigma de objetos distribuídos. Atualmente o consórcio engloba mais de 800 participantes.

O principal resultado do OMG foi a definição de uma arquitetura para o desenvolvimento de aplicações por objetos distribuídos. Esta arquitetura, CORBA *(Common Object Request Broker Architecture)* (Fig. 11), estabelece a funcionalidade de uma camada *middleware*, o ORB *(Object Request Broker)* e as funcionalidades de serviços de objetos padrões, *CORBAservices*, que incluem serviços de busca de objetos por nome *(naming)*, busca por propriedades *(trader)*, serviços de segurança, de transações e de notificação de eventos, entre outros.

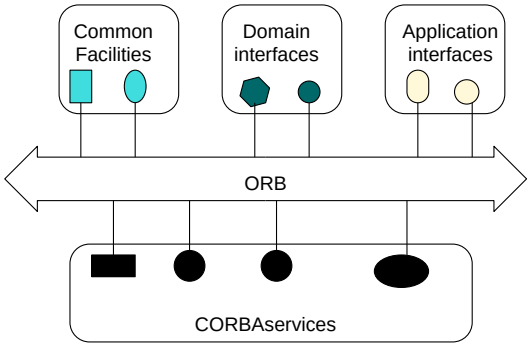

Figura 11: Arquitetura de gerenciamento de objetos (OMA).

Além disto, outros tipos de serviços podem estar agrupados em *Common Facilities, Domain Interfaces* ou *Application Interfaces* [42]. Assim como *CORBAservices*, *Common Facilities* incluem serviços de propósito geral; porém, tais serviços estão mais próximos em nível de abstração do usuário final, enquanto *CORBAservices* apresentam nível de abstração baixo. Um exemplo de uma facilidade nesta categoria é o *Distributed Document Component Facility*, para a manipulação de documentos compostos. As interfaces de domínio são serviços especializados para dadas categorias de aplicações, como em área financeira, telecomunicações e medicina. As interfaces de aplicação são componentes da arquitetura específicos de uma aplicação, não sendo portanto objeto de padronização.

Como mostra a Fig. 12, ORB e bancos de dados aderentes à proposta ODMG apresentam alguma sobreposição de funcionalidades, mas apresentam um alto grau de complementação de serviços oferecidos. Antevê-se que a combinação destas duas linhas de desenvolvimento seja a direção para as novas aplicações de sistemas de informação em ambientes distribuídos.

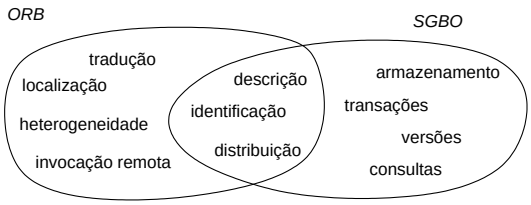

Figura 12: Funcionalidades ORB e ODMG.

Uma análise da integração entre sistema gerenciador de base de objetos e CORBA, particularmente para aplicações multimídia, foi apresentada por Tobar [49]. Similarmente, outras categorias de aplicação poderiam analisar qual plataforma atende melhor às suas necessidades para selecionar a estratégia de implementação.

6.3 Conclusões

A tecnologia de orientação a objetos está permeando praticamente todas as novas aplicações computacionais. O que há algum tempo atrás era apenas uma *buzzword* que dava um "ar de modernidade" para algumas aplicações é atualmente uma tecnologia relativamente madura, com utilização efetiva nas mais diversas categorias de aplicação.

Banco de dados é apenas uma das áreas que vem se beneficiando desta tecnologia. Provavelmente, mais do que simplesmente oferecer a capacidade de ter um repositório orientado a objetos para aplicações isoladas, a importância de sistemas gerenciadores de base de objetos será permitir a integração de aplicações em ambientes de objetos distribuídos. Nesta abordagem, os serviços de um sistema gerenciador de base de objetos poderão ser incorporados a aplicações de forma uniforme e transparente, da mesma maneira que assim o serão serviços de interfaces gráficas com usuários ou invocação dinâmica de serviços distribuídos.

A chave para atingir esta próxima etapa na tecnologia de banco de dados está na generalidade. Ao contrário do paradigma de desenvolvimento que dominou o final da década de 1970 e praticamente toda a década de 1980, quando havia dezenas ou centenas de sistemas gerenciadores de banco de dados especializados para a aplicação x ou para a aplicação y, o objetivo atual é oferecer serviços básicos, possivelmente configuráveis, que possam ser integrados de acordo com as necessidades de cada aplicação.

Esta abordagem de oferecimento de serviços atende de forma muita adequada às novas categorias de aplicação em sistemas de informação, incluindo a manipulação de informação multimídia e hipermídia. Dificilmente seria possível desenvolver um sistema gerenciador de banco de dados especializado que atendesse adequadamente às necessidades de todas aplicações multimídia ou hipermídia, pois estas variam muito de acordo com o caráter da aplicação. No entanto, é perfeitamente viável imaginar que tais aplicações serão atendidas através da combinação de serviços especializados de armazenamento, de consulta, de pesquisa, e assim por diante. O oferecimento de tais serviços torna-se interessante à medida que eles não são de uso exclusivo de um banco de dados ou de uma aplicação. Assim, o mesmo serviço de pesquisa que é utilizado por uma aplicação multimídia poderia ser utilizado em uma outra aplicação em *data mining*, por exemplo.

Em suma, há atualmente produtos em bancos de dados orientados a objetos que utilizam uma tecnologia que já está razoavelmente amadurecida, atendendo a muitas aplicações. Há ainda, no entanto, uma necessidade de se estender as funcionalidades desses serviços de bancos de dados para qualquer aplicação desenvolvidas em *frameworks* de objetos distribuídos. Para atingir este objetivo, as pesquisas nesta área seguem duas diretrizes fundamentais. A primeira pode ser resumida através das palavras generalidade e adaptabilidade. Desenvolvimentos em *design patterns* são emblemáticos desta diretriz. A segunda pode ser resumida pela palavra inter-operabilidade, onde a tecnologia de objetos distribuídos vem se destacando.

Referências

[1] S. Abiteboul e P. Kanellakis. The two facets of object-oriented data models. *IEEE DB Engineering Bulletin*, 14(2), Junho 1991.

[2] M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, , J. N. Gray, P. P. Griffiths, W. F. King, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, e V. Watson. System R: Relational approach to database management. *ACM Transactions on Database Systems*, 1(2):97–137, Junho 1976.

[3] T. R. Ayers, D. K. Barry, J. D. Dolejsi, J. R. Galareneau, e R. V. Zoeller. Development of ITASCA. *Journal of Object-Oriented Programming*, 4(4):46–49, Julho 1991.

[4] Jay Banerjee, Won Kim, Hyoung-Joo Kim, e Henry F. Korth. Semantics and implementation of schema evolution in object-oriented databases. Em *Proceedings of the ACM SIGMOD Conference*, páginas 311–322. 1987.

[5] J. H. ter Bekke. *Semantic Data Modeling.* Prentice Hall, 1992.

[6] M. W. Blasgen, M. M. Astrahan, D. D. Chamberlin, J. N. Gray, W. F. King, B. G. Lindsay, R. A. Lorie, J. W. Mehl, T. G. Price, G. R. Putzolu, M. Schkolnick, P. G. Selinger, D. R. Slutz, H. R. Strong, I. L. Traiger, B. W. Wade, e R. A. Yost. System R: An architectural overview. *IBM Systems Journal*, 20(1):41–62, 1981.

[7] Robert Bretl, David Maier, Allen Otis, Jason Penney, Bruce Schuchardt, Jacob Stein, E. Harold Williams, e Monty Williams. The GemStone data management system. Em Kim e Lochovsky [25], capítulo 12, páginas 283–308.

[8] P. Butterworth, A. Otis, e J. Stein. The GemStone object database management system. *Communications of the ACM*, 34(10):64–77, Outobro 1991.

[9] R. G. G. Cattell, editor. *The Object Database Standard: ODMG–93.* Morgan–Kaufmann Publishers, San Mateo, CA, 1993.

[10] R. G. G. Cattell. *Object Data Management: Object-Oriented and Extended Relational Database Systems.* Addison-Wesley, 1994. Revised edition.

[11] Peter P. Chen. The entity-relationship model — toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, Março 1976.

[12] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, Junho 1970.

[13] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, Dezembro 1979.

[14] George Copeland e David Maier. Making Smalltalk a database system. Em *Proceedings of the SIGMOD Conference*, páginas 316–325. ACM, 1984.

[15] O. Deux, C. Delobel, D. DeWitt, Y. Ioannidis, et al. The story of O2. *IEEE Transactions on Knowledge and Data Engineering*, 2(1), Março 1990.

[16] Ramez Elmasri e Shamkant B. Navathe. *Fundamentals of Database Systems.* Benjamin/Cummings Publishing Company, segunda edição, 1994.

[17] D. H. Fishman, J. Annevelink, D. Beech, E. Chow, T. Connors, J. W. Davis, W. Hasan, C. G. Hoch, W. Kent, S. Leichner, P. Lyngbaek, B. Mahbod, M. A. Neimat, T. Risch, M. C. Shan, e W. K. Wilkinson. Overview of the Iris DBMS. Em Kim e Lochovsky [25], capítulo 10, páginas 219–250.

[18] J. P. Fry e E. H. Sibley. Evolution of data-base management systems. *ACM Computing Surveys*, 8(1):7–42, Março 1976.

[19] Adele Goldberg e David Robson. *Smalltalk-80, The Language*. Addison-Wesley Series in Computer Science. Addison-Wesley, Setembro 1989.

[20] J. Gosling, B. Joy, e G. Steele. *The Java Language Specification.* Java Series. Addison-Wesley, 1996. `http://java.sun.com/doc/language_specification/index.html`.

[21] James Gosling e Henry McGilton. *The Java Language Environment: A White Paper*. Sun Microsystems, Inc., Mountain View, CA, Maio 1996. `http://www.javasoft.com/docs/white/langenv/`.

[22] Richard Hull e Roger King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):201–260, Setembro 1987.

[23] Won Kim, editor. *Modern Database Systems: The Object Model, Interoperability and Beyond.* ACM Press, 1995.

[24] Won Kim, Nat Ballou, Hong-Tai Chou, Jorge F. Garza, e Darrel Woelk. Features of the ORION object-oriented database system. Em Kim e Lochovsky [25], capítulo 11, páginas 251–282.

[25] Won Kim e Frederick H. Lochovsky, editores. *Object-Oriented Concepts, Databases, and Applications.* Frontier Series. ACM Press, 1989.

[26] Henry F. Korth e Abraham Silberschatz. *Database System Concepts.* McGraw-Hill, 1986.

[27] George Lapis, Guy M. Lohman, e Hamid Pirahesh. Starburst is born. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 19(2), Junho 1990.

[28] Léo P. Magalhães, Armando L. N. Delgado, Ivan L. M. Ricarte, Regina C. Ruschel, e Carlos José M. Olguín. Implementação de um banco de dados não-convencional: a experiência GERPAC/UniCOSMOS. Em *IV SBBD — Simpósio Brasileiro de Banco de Dados*, páginas 77–89. Campinas, SP, Abril 1989.

[29] Oscar Nierstrasz. A survey of object-oriented concepts. Em Kim e Lochovsky [25], capítulo 1, páginas 3–21.

[30] Farid Nourani e Léo Pini Magalhães. Um sistema de tratamento de restrições de projeto no contexto GERPAC/UniCOSMOS. Em *VI Simpósio Brasileiro de Banco de Dados*. Manaus, AM, Maio 1991.

[31] Kristen Nygaard. Basic concepts in object oriented programming. *SIGPLAN Notices*, 21(10):128–132, Outobro 1986.

[32] Object Design, Inc., Burlington, MA. *ObjectStore Technical Overview; Release 2.0*, Julho 1992.

[33] Object Design, Inc. *ObjectStore C++ API User Guide*. Burlington, MA, Junho 1995.

[34] Object Management Group. OMG home page. `http://www.omg.org/`, 1998.

[35] Carlos José M. Olguín e Léo Pini Magalhães. Um gerenciador de transações no contexto do GERPAC/UniCOSMOS. Em *V Simpósio Brasileiro de Banco de Dados*. Rio de Janeiro, RJ, Abril 1990.

[36] Kamran Parsaye, Mark Chignell, Setrag Khoshafian, e Harry Wong. *Intelligent Databases: Object-Oriented, Deductive, Hypermedia Technologies*. John Wiley and Sons, 1989.

[37] Joan Peckham e Fred Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, Setembro 1988.

[38] Walter D. Potter e Robert P. Trueblood. Traditional, semantic, and hyper-semantic approaches do data modeling. *IEEE Computer*, páginas 53–63, Junho 1988.

[39] Ivan L. M. Ricarte. MOODS: a modular, object-oriented design database system. Em F. Kimura e A. Rolstadås, editores, *Computer Applications in Production and Engineering*, páginas 225–229. North-Holland, 1989.

[40] Ivan L. M. Ricarte e Armando L. N. Delgado. GERPAC — um SGBD para PAC. Em *II Simpósio Brasileiro de Banco de Dados*, páginas 22–38. Porto Alegre, RS, Maio 1987.

[41] Lawrence A. Rowe e Michael R. Stonebraker. The POSTGRES data model. Em *Proceedings of the 13th VLDB Conference*, páginas 83–96. Brighton, UK, 1987.

[42] Doug Schmidt. Overview of CORBA. `http://www.cs.wustl.edu/~schmidt/corba-overview.html`, Agosto 1998.

[43] Mamede Augusto Machado da Silveira. *Extensão do Núcleo UniCOSMOS para Suporte à Orientação por Objeto.* Tese de mestrado, Faculdade de Engenharia Elétrica, UNICAMP, Abril 1991.

[44] John Miles Smith e Diane C. P. Smith. Database abstractions: Aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):105–133, Junho 1977.

[45] Michael Stonebraker. Retrospection on a database system. *ACM Transactions on Database Systems*, 5(2):225–240, Junho 1980.

[46] Michael Stonebraker, Eugene Wong, Peter Kreps, e Gerald Held. The design and implementation of INGRES. *ACM Transactions on Database Systems*, 1(3):189–222, Setembro 1976.

[47] Bjarne Stroustrup. What is object-oriented programming? *IEEE Software*, páginas 10–20, Maio 1988.

[48] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, segunda edição, 1991.

[49] Carlos M. Tobar e Ivan L. M. Ricarte. Multiware database, a distributed object database system for multimedia support. Em *Open Distributed Processing: Experiences with Distributed Environments, Proceedings of the IFIP International Conference on Open Distributed Processing, ICODP'95*, páginas 439–450. Fevereiro 1995.

[50] M. Ubell. The Montage extensible DataBlade architecture. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):482–482, Junho 1994.