Aspectos de Implementação de Servidores de banco de Dados Orientados ao Objeto*

Renato Campos Mauro Marta Lima de Queirós Mattoso

Programa de Engenharia de Sistemas e Computação
COPPE/UFRJ
Caixa Postal 68511, Rio de Janeiro, RJ, 21945-970
e-eletrônico: { rmauro, marta }@cos.ufrj.br

Resumo:

O objetivo deste trabalho é apresentar o gerente de armazenamento físico de objetos implementado no Sistema de Gerência de Objetos GOA++. Os principais serviços do gerente de armazenamento são a gerência de representação e acesso aos objetos e a gerência dos meios de armazenamento. O gerente de objetos armazenados contém serviços de identificação de objetos, representação de objetos simples e longos, referência entre objetos e agrupamento de objetos no disco. O gerente de meios de armazenamento oferece uma série de serviços para o armazenamento dos objetos em meio secundário e primário. A unidade de transferência é a página que pode ser definida no momento de configuração da base de dados. O servidor de páginas possui diversos algoritmos que cuidam da gerência eficiente de páginas em memória, sendo este outro parâmetro flexível na utilização do gerente de armazenamento. O sistema foi todo desenvolvido em C++ de modo a facilitar a reutilização das classes por outros sistemas que não o GOA++ a fim de obter serviços de persistência para aplicações.

Palavras Chaves: Banco de Dados, Orientação a Objetos, Estrutura de Dados

1. Introdução

Com a larga utilização do paradigma da orientação a objetos no desenvolvimento de sistemas e aplicações, o uso de Sistemas de Banco de Dados Orientados a Objetos (SBDOO) vem se tornando uma solução integradora para fornecer serviços de persistência e gerência de objetos dessas aplicações. Características da orientação a objetos como abstração de dados, identidade de objetos, herança, polimorfismo, objetos complexos, dentre outras, impõem novos requisitos para a gerência do armazenamento desses objetos.

O objetivo deste trabalho é apresentar o gerente de armazenamento físico de objetos implementado no sistema GOA++ [MAUR97]. O GOA++ é um Gerente de Objetos Armazenados cujo objetivo é oferecer persistência e serviços de gerência de coleções e objetos. Dentre os serviços de gerência de coleções está o processamento de consultas sobre a extensão de coleções de objetos. Um dos objetivos do GOA++ é o desempenho que está ligado diretamente ao modelo de armazenamento físico dos objetos apresentado neste trabalho.

O sistema GOA++ é fruto de uma evolução de seu antecessor GOA [MATT94a]. Após uma série de avaliações[MATT94b] do GOA com o *benchmark* OO7 [CARE93], várias técnicas utilizadas no modelo físico de representação de objetos foram revistas dando origem a novas técnicas de armazenamento incorporadas ao sistema GOA++, como a nova política de alocação e remoção de objetos, a gerência do espaço de armazenamento disponível e a gerência de objetos longos.

Embora já existam vários SBDOOs no mercado, como O_2, ObjectStore, GemStone e outros, este trabalho apresenta um gerente de armazenamento que pode ser utilizado tanto no GOA++ como em outras aplicações da linguagem de programação C++ que necessitem de serviços de persistência de objetos simples e longos. Uma característica especial do gerente de armazenamento do GOA++ é sua capacidade de processamento paralelo [MEYE97]. Foram desenvolvidos serviços de gerência de execução paralela para meios de armazenamento distribuídos numa rede Ethernet através do software PVM (Parallel Virtual Machine) [GEIS94].

Os principais serviços do gerente de armazenamento são a gerência de representação e acesso aos objetos e a gerência dos meios de armazenamento. O gerente de objetos armazenados contém serviços de identificação de objetos, representação de objetos simples e longos, referência entre objetos e agrupamento de objetos no disco. O gerente de meios de armazenamento oferece uma série de serviços para o armazenamento dos objetos em meio secundário e primário. A unidade de transferência é a página que pode ser definida no momento de configuração da base de dados. O servidor de páginas possui diversos algoritmos que cuidam da gerência eficiente de páginas em memória, sendo este outro parâmetro flexível na utilização do gerente de armazenamento.

O sistema foi todo desenvolvido em C++ de modo a facilitar a reutilização das classes que oferecem os serviços de persistência para as aplicações. Atualmente é utilizado tanto em plataformas de estações de trabalho Unix como em microcomputadores. No ambiente micro, os serviços do GOA++ estão disponíveis via uma API implementada utilizando bibliotecas dinâmicas (.dll), fazendo a comunicação com o servidor.

Este trabalho está organizado da seguinte forma. Na Seção 2 é apresentada a arquitetura geral do GOA++ onde foram implementados os serviços de armazenamento apresentados neste trabalho. A Seção 3 apresenta as políticas de gerência dos meios de armazenamento. A Seção 4 descreve as técnicas de gerência da persistência dos objetos e finalmente a Seção 5 conclui este trabalho.

2. Arquitetura do GOA++

O GOA++ é um Gerente de Objetos Armazenados cujo objetivo é oferecer persistência e serviços de gerência de coleções e objetos. Dentro dessa gerência estão incluídos o processamento de predicados de consultas sobre coleções, o agrupamento de objetos no disco, além do emprego de técnicas recentes para gerência de cache e armazenamento de objetos. O GOA++ é fruto de uma re-engenharia de código C do servidor GOA [MATT94a] para C++, onde vários serviços foram otimizados, estendidos e adaptados para atender às exigências dos padrões ODMG [CATT94] e CORBA [OMG95]. A Figura 1 ilustra a arquitetura do GOA++.

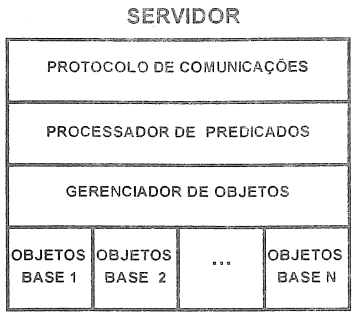

Figura 1 : Arquitetura do GOA++

A camada mais externa é constituída de protocolos de comunicação padrão CORBA. A camada intermediária é composta por um processador de predicados que seleciona objetos de coleções, a partir de uma linguagem intermediária. Esta linguagem é gerada a partir de um processador OQL, linguagem de consulta declarativa para consultas em coleções de objetos. O processador de predicados também é capaz de criar objetos segundo descrições ODL, também pré-processadas. As linguagens ODL e OQL são propostas pelo padrão ODMG. O gerenciador de objetos é a camada mais baixa da arquitetura. Ela tem como objetivo gerenciar a persistência dos objetos de maneira eficiente. As seções subseqüentes tem por objetivo apresentar os aspectos de implementação desta camada. Este gerenciador pode ser utilizado por outras aplicações - persistência em linguagens de programação, por exemplo. Conhecer como esta camada está implementada é fundamental para aqueles que desejam utilizá-la de maneira otimizada. Informações e contato com os pesquisadores do projeto do servidor GOA++ podem ser obtidas em [GOA96].

3. Servidor de Páginas

O Servidor de páginas é a camada responsável em fazer acesso a disco. Este servidor oferece como serviços a seus clientes basicamente leitura e escrita de registros. Registro é um conjunto de bytes que começam e terminam em uma mesma página. Denomina-se página a menor unidade de dados que de uma só vez é copiada do disco para a memória. O tamanho da página é um parâmetro dependente do ambiente onde o banco de dados é instalado. Esse parâmetro costuma variar entre 512 e 4096 bytes. Uma vez inicializada a base de dados com um valor específico, durante todo o ciclo de vida da base o tamanho da página não é alterado.

Quando o servidor de páginas inicializa um novo arquivo que conterá a base de objetos, ele o cria em função de um número inicial de páginas. Quando o arquivo tornar-se pequeno e houver necessidade de expansão, isso deve ser solicitado explicitamente pelo cliente.

3.1. Anatomia do servidor de páginas

O servidor de páginas é composto por uma memória cache e um índice associando um número de página do disco a uma posição no cache. Quando um registro é solicitado pelo cliente, o servidor de páginas checa no índice para ver se a página correspondente se encontra em

memória principal. Em caso afirmativo o registro é rapidamente despachado para o cliente. A ausência da página no cache implica que deverá ser feito um acesso a disco. A página recém acessada é imediatamente armazenada no cache.

O tamanho do cache é especificado (em número de páginas) pelo cliente no momento da inicialização do servidor. Ele é implementado por uma lista duplamente encadeada, ordenada por sua prioridade. No momento em que se solicita um registro cuja página não se encontra no cache, é a página menos prioritária que cede lugar para a nova página. Uma questão que surge é saber onde colocar a nova página. Será que a posição que ela já se encontra é a melhor posição? Não seria interessante esta página ocupar uma posição mais prioritária?

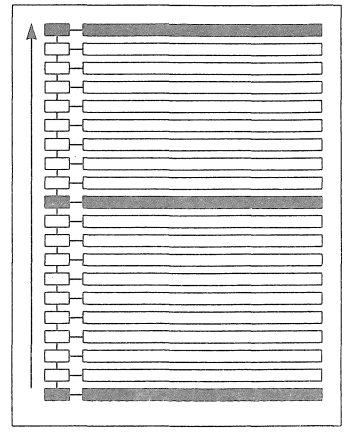

Figura 2 : Cache do Servidor de Páginas

O servidor deixa por conta do cliente sugerir uma "promoção" inicial para a página recém vinda do disco. Este indicador de localidade da página pode ser "alto", "médio" ou "baixo", correspondendo à posição mais prioritária, à posição mediana e à posição menos prioritária respectivamente. Estas posições aparecem em destaque na Figura 2. A seta indica a sentido das páginas mais prioritárias.

Quando um registro cuja página já está no cache é solicitado, a página recém acessada é promovida uma posição na direção das páginas mais prioritárias, trocando de posição com sua vizinha.

Cada elemento da lista do cache é composto por cinco campos conforme ilustrado na Figura 3: um bit indicando se a página foi modificada ou não, o número da página no disco, um ponteiro para o buffer onde os bytes da página correspondente ficam armazenados em memória e 2 ponteiros paras as páginas vizinhas (próxima = mais prioritária e anterior = menos prioritárias), necessários para fazer o encadeamento da lista. O bit indicador é setado na primeira vez que se faz uma modificação, isto é, cada vez que o cliente escrever um registro. Quando a página sair do cache (porque tornou-se a menos prioritária ou está na hora de fechar a base de dados), se o bit estiver ligado, a atualização é feita no disco. Caso contrário, a página é simplesmente descartada, sem ter que fazer nenhum acesso ao disco.

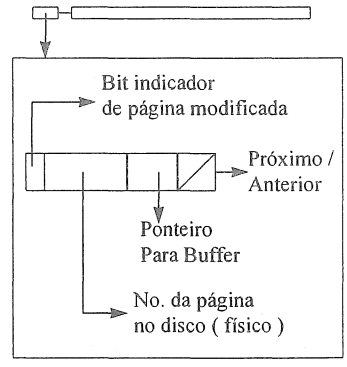

Figura 3: Elemento da Lista do Cache

3.2. *Índice do cache*

Já foi citado que o cache foi implementado utilizando listas duplamente encadeadas. Esta estrutura de dados é particularmente interessante para essa caso, pois economiza esforços de ficar copiando bytes desnecessariamente. Trocar dois elementos quaisquer de posição dentro do cache implica simplesmente em atualização dos ponteiros responsáveis pelo encadeamento.

Sabe-se porém, que a recuperação de elementos em listas duplamente encadeadas só pode ser feito seqüencialmente, o que pode ser desastroso para o desempenho do sistema. Para

contornar essa situação, implementa-se um índice (*hashing*) cuja chave é o número da página e o valor obtido é o endereço de memória do elemento correspondente no cache.

3.3. Gerente de Páginas

O gerente de páginas é uma especialização do servidor de páginas. Ele mantém a informação de que páginas estão sendo utilizadas no arquivo que contém a base de objetos. Denomina-se "alocadas" às páginas que estão sendo utilizadas, em oposição às "desalocadas" (as páginas não utilizadas). Tentativas de Leitura ou escrita em páginas não alocadas provocam um erro no servidor de páginas. Estrutura de dados e algoritmos para gerenciamento de espaços disponíveis já vêm sendo discutido há bastante tempo na literatura. Estes trabalhos, porém, dão enfoque para gerenciamento em memória não persistente, objetivando o principalmente as linguagens de programação. McAULIFFE et. al. [McAu96] é um dos trabalhos mais recente enfocando armazenamento em memória secundária.

As informações sobre o espaço disponível é guardado em listas, utilizando para isso o próprio espaço disponível.

O último byte de cada uma das páginas é utilizado pelo gerente de páginas para "rotular" cada uma delas. Esse byte indicada se a página está sendo utilizada ou não; e se utilizada, ele indica para que.

Este rótulo é bastante interessante, pois além de diferenciar as páginas utilizadas das não utilizadas; conhecendo-se a natureza das utilizadas, podemos reorganizar o arquivo em "batch".

4. Gerência de Persistência de Objetos

4.1. IDO: A identificação dos Objetos

Existem várias abordagens de implementação de identificadores de objetos na literatura [KHOS86]. A versão original do GOA utiliza OID físico. Este tipo de identificador corresponde ao deslocamento do início do objeto no arquivo que a contém. A vantagem da utilização desse método é a velocidade de acesso do objeto. Nenhum cálculo precisa ser feito. Nenhuma tabela precisa ser consultada. A desvantagem é que os objetos não podem mudar de posição. Nem mesmo deslocar um único byte para frente ou para trás.

O GOA++ utiliza o conceito de identificador estruturado. O identificador é composto por dois campos: o número da página e o índice do objeto dentro da página. O número da página corresponde à parte física do identificador. A vantagem é que se garante um acesso rápido à página onde objeto está alocado. Não há necessidade de consultar nenhuma outra informação: simplesmente acessamos a página e podemos ter certeza que o objeto ali está. O índice corresponde à parte lógica do identificador: precisa ser processado para se conheça o deslocamento do objeto. Se o identificador possuir índice nulo, indica que é um "objeto longo", precisando de um tratamento especial. Nas seções que seguem são apresentados a anatomia das páginas contendo objetos curtos e os algoritmos para processamento de objetos longos.

4.2. Objetos Curtos

Objetos curtos são assim chamados por caberem inteiramente dentro de uma única página. A Figura 4 ilustra a anatomia das páginas que contém objetos curtos. Ela contém 3 objetos, identificados pelas duplas (123, 1) , (123, 2) e (123, 3). O número de página "123" foi escolhido ao acaso. No início da Figura 4 (a) existe uma região chamada "área de controle". Esta região armazena a informação de "quantos" objetos estão ali armazenados (# Objs) , o identificador do agrupamento (id Grupo) e uma descrição para cada um dos objetos. O espaço disponível (Figura 4 (b)) fica todo no final da página. Cada objeto é descrito por seu tamanho e por seu índice.

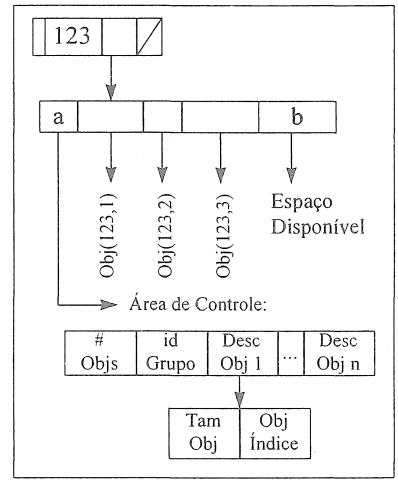

Figura 4: Pagina de Objetos Curtos

Já citamos uma vantagem do identificador estruturado: acesso rápido à pagina que contém o objeto. Uma segunda vantagem é que os objetos podem mudar à vontade de posição dentro da página, sem com isso invalidar seu identificador. Considere o processo de obter um objeto e armazená-lo modificado com um tamanho superior ao tamanho inicial, ilustrado na Figura 5. A página onde o objeto está armazenado é diretamente obtida a partir do seu IDO. O objeto é recuperado dentro da página a partir de seu índice. Aos bytes do objeto, uma pequena quantidade de bytes é anexada e então o objeto é armazenado de volta na página. Como a taxa de crescimento foi menor que o espaço disponível, o objeto modificado simplesmente empurrou os outros para frente, diminuindo apenas o espaço disponível no final da página.

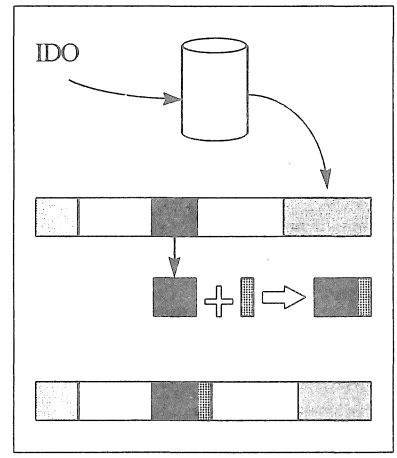

Figura 5: Armazenamento de Objeto Modificado

A desvantagem em utilizar IDO físico a nível de páginas é que o objeto não pode mudar de endereço. Imagine uma situação parecida com a ilustrada pela Figura 5, com a diferença de que a taxa de crescimento do objeto excede o espaço disponível dentro da página. O objeto ainda cabe, porém inteiro dentro de uma página. Não há solução senão o objeto mudar de página. A mudança de página implicaria na mudança de IDO, o que é terminantemente proibido.

Como solução a esse problema, deixamos no seu endereço antigo a informação de que o objeto "mudou-se", indicando também o seu novo endereço. O Objeto passa a ser acessado por (no máximo) dois acessos a disco: o primeiro para pegar os dados na página original, aonde ele pode descobrir que o objeto não está mais ali, tendo que ir buscá-lo no novo endereço. Se uma terceira mudança de endereço for necessária, o endereço original passa apontar diretamente para o mais novo endereço.

4.3. *Objetos Longos*

Objetos longos (OL) são assim caracterizados pelo fato de não caberem completamente em uma única página, em oposição aos objetos curtos. O Servidor deve dedicar-lhe um tratamento especial. Objetos longos já não são novidade em sistemas de banco de dados, incluindo os relacionais, que disponibilizam campos de binários longos e memorandos, que podem armazenar imagens e textos de qualquer tamanho.

Um objeto longo pode ser estruturado ou não estruturado. Os estruturados tem como características serem "conhecidos" pelo servidor. O servidor é capaz de processar as informações nele armazenadas. Os não estruturados são vistos pelo servidor apenas como um conjunto de bytes. A aplicação cliente é que sabe interpretá-lo. Lista de objetos, listas de inteiros, tabelas de strings são exemplos de objetos estruturados. Vídeo AVI, som WAV, imagem GIF são exemplos de objetos não estruturados.

No nível que este artigo está examinando, não há diferença entre objetos estruturados e não estruturados. Ambos tem como características não caberem na página e possivelmente não caberem sequer na memória principal. O processamento dos objetos longos devem ser feito em trechos.

As funcionalidades oferecidas pelos bancos de dados variam de servidor para servidor. Alguns oferecem o processamento similar a uma "*stream*": disponibilizando métodos para posicioná-la, ler e escrever. Uma variação ("*Piece stream*") permite que além das operações citadas, o usuário possa inserir e remover bytes de qualquer parte do OL, o que é conveniente para um grande número de aplicações - processador de textos, por exemplo. Este tipo de implementação é bem mais complicado de ser feita eficientemente. Alguns sistemas, como o EXODUS [CARE88] e ORION utilizam um índices do tipo árvore-B para obter essa funcionalidade. A seguir apresentaremos o gerenciamento de OLs implementado no GOA++. Esse sistema também usa índice para endereçar os bytes, só que não multi-nível (como árvores-B), mas sim de apenas um nível, diminuindo o custo da estrutura de dados.

4.3.1. Implementação dos objetos longos

Os primeiros bytes de um objeto longo correspondem a uma lista contendo informações sobre cada uma das páginas integrantes do OL. Para ilustrar a composição da lista, considere um "pequeno objeto longo" com apenas 2 páginas (Figura 6). Cada elemento dessa lista (exceto o primeiro) é composto pelo número da página e pelo número de bytes nela armazenados. O primeiro elemento armazena o número de páginas que compõe o OL, além do número de bytes da primeira página.

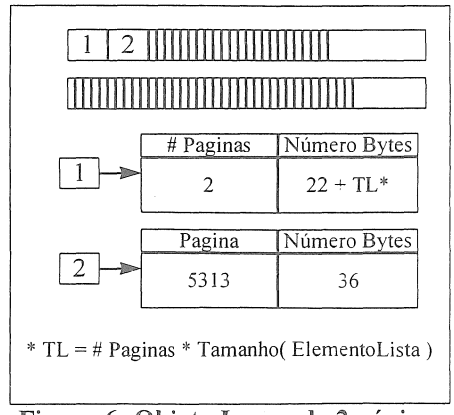

Figura 6: Objeto Longo de 2 páginas

A lista contendo a descrição das páginas do OL podem ter qualquer tamanho. Não está limitada a uma única página. Considere pequenas páginas de 8 bytes cada uma, com elementos da lista de 2 bytes: 1 para identificar a página, outro para saber quantos elementos ela armazena. Considere também um OL de 31 bytes, distribuídos pelas páginas conforme ilustra a Figura 7. Talvez a figura possa dar a impressão que as páginas são contínuas no disco, mas isso é falso. As páginas podem estar distribuídas aleatoriamente no disco. A ordem das páginas é imposta pelos elementos da lista.

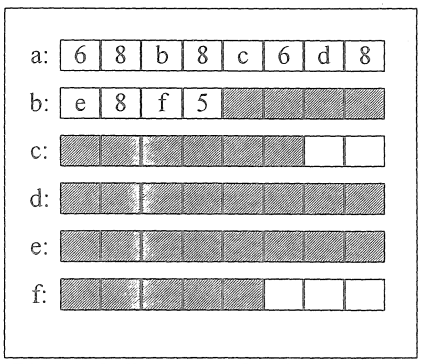

Figura 7: Exemplo de objeto longo

Uma posição de um OL é facilmente descoberta, bastando para isso percorrer a lista que está no início do OL. Uma característica fundamental é que a única página cuja descrição está armazenada nela mesma é a primeira página. É justamente esse fato que garante que a lista pode ter qualquer tamanho. Quando a primeira página terminar de ter sido percorrida, já se conhece a descrição das primeiras páginas do OL. Em particular todas aquelas cuja descrição está na página recém percorrida. Este fato garante obter todas as descrições das páginas, garantido acessibilidade para todas as páginas .

Para garantir uma boa utilização das páginas ocupadas pelo objeto longo, temos a restrição de que todas as páginas (exceto a primeira) deve ter ocupação maior que 50% (semelhante à árvore B). Caso contrário poderíamos ter uma baixa utilização do disco.

4.4. Agrupamento de Objetos

A principal idéia do agrupamento de objetos é permitir que objetos que sejam acessados freqüentemente juntos estejam próximos uns dos outros no disco. O GOA++ permite que o usuário especifique perto de qual objeto, deseja-se armazenar o novo objeto. Com essa abordagem simples, podemos agrupar objetos de diversas formas. É interessante lembrar que não existe uma melhor maneira de agrupamento. Tudo depende da natureza e freqüências das consultas. Esses dados são conhecidos apenas pelo projetista da aplicação, que tem a liberdade de organizar o disco da maneira mais conveniente.

Um primeiro modo de se agrupar objetos é pela extensão da classe. Nessa abordagem objetos de uma mesma classe estarão próximos uns dos outros no disco. Uma outra maneira de agrupar é por faixa de valores. Uma terceira maneira é agrupar pelo "objeto raiz". Nessa abordagem agrupamos todos objetos relacionados a um outro objeto dado. Considere, por exemplo, duas classes: departamento e funcionário. Vários funcionários trabalham em um departamento. Podemos estar interessados em agrupar todos os funcionários de um mesmo departamento.

5. Conclusão

Este trabalho apresentou como o GOA++ gerencia o armazenamento dos objetos em memória secundária. Uma das preocupações foi atacar os problemas relativos a eficiência do acesso de objetos em geral (tanto objetos curtos como objetos longos), tentando obter uma boa utilização do espaço em disco. O módulo responsável por este gernciamento pode ser re-utilizado em outras aplicações que precisem apenas de persistência de seus objetos. O GOA++, por estar implementado em camadas permite que novas funcionalidades sejam inseridas facilmente. Estruturas de dados para indexação, por exemplo, podem ser facilmente implementadas utilizando o servidor de páginas. Aproveitando toda a infra-estrutura do GOA++ novas extensões estão sendo propostas, especializando o gerente de objetos.

6. Referências bibliográficas

[CARE88] CAREY, M., DeWITT D., RICHARDSON, J. SHEKITA, E., "Storage Management for Objects in EXODUS" in "Object Oriented Concepts, Databases and Applications" W. KIM, F.H Lochovsky - Addison Wesley - Massachusetts, 1988.

[CARE93] Carey, M.J. DeWitt, D. Naughton, J. "The 007 Benchmark" Proc ACM SIGMOD International Conference on Management of Data, junho 1993, pp.12-21.

[CATT94] CATTEL, R.G.G, 1994, "Object Data Management", Addison-Wesley Publishing Company, Revised Ed.

[CATT94] CATTELL R.G.G., "The Object Database Standard: ODMG-93 - Release 1.1, San Francisco, California, Morgan Kaufmann Publishers, Inc., 1994

[ELMA94] ELMASRI, R., NAVATHE, S.B., 1994 "Fundamentals of Database Systems", The Benjamin/Cummings Publish Company, Inc., 2ª. edição

[GEIS94] GEIST AL. et al; "PVM: Parallel Virtual Machine - A User's guide and Tutorial for Networked Parallel Computing; The MIT Press-Cambridge, Massachusetts, Londres; 1994

[GOA96] Baião, F. A.. url: http://www.cos.ufrj.br/~goa/ , 1996

[KHOS86] KHOSHAFIAN, S., COPELAND, G., "Object Identity" in Proceedins of the conference on Object-Oriented Programming Systems and Languages (OOPSLA), Portland, Oregon, September 1986.

[KHOS94] KHOSHAFIAN, S. 1994 , "Object Oriented Databases", John Wiley & Sons, inc.

[KIM95] KIM, W., 1995 "Modern Database System", ACM Press.

[MATT94a] MATTOSO, M.L.Q., SOUZA, J.M. "GOA: Um Servidor de Objetos Persistentes para Sistemas de Banco de Dados Orientados a Objetos", XX Conferência Latinoamericana de Informárica, Cidade do México, México, setembro de 1994.

[MATT94b] MATTOSO,M.L.Q., SILVA,G.Z.1994, "Explorando o Processamento de Consultas no Servidor de Objetos GOA", In: *Anais 9 Simpósio Brasileiro de Banco de Dados*.

[MAUR97] MAURO R.C. et. al. "GOA++: Tecnologia, Implementação e Extensões aos Serviços de Gerência de Objetos". Trabalho a ser publicado nos anais do XII Simpósio Brasileiro de Banco de Dados da Sociedade Brasileira de Computação, Fortaleza, Brasil, outubro 1997.

[McAU96] McAULIFFE M.L., CAREY, M. J., SOLOMON, M.H., "Toward an Effective and Efficient Free Space Management", Proceedings of the SIGMOD'96 pp. 389-400 Monteal Canada, Junho de 1996.

[MEYE97] Meyer L.V. , 1997, *Paralelismo em SGBDOO com memória distribuída: Uma implementaçao no PARGOA,* Trabalho a ser publicado nos anais do XII Simpósio Brasileiro de Banco de Dados da Sociedade Brasileira de Computação, Fortaleza, Brasil, outubro 1997.

[OMG95] Object Management Group, "The Commom Object Request Broker Architecture and Specification", Revisão 2.0, julho de 1995.