UMA METODOLOGIA E ALGORITMOS PARA O PROJETO DE DISTRIBUIÇÃO DE BASES DE DADOS USANDO REVISÃO DE TEORIAS

Fernanda Araujo Baião Amorim

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO

Aprovada por:

Profª. Marta Lima de Queirós Mattoso, D.Sc.

Prof. Gerson Zaverucha, Ph.D.

Profª. Ana Carolina Salgado, D.Sc.

Profª. Sheila Regina Murgel Veloso, D.Sc.

Profª. Maria Luiza Machado Campos, Ph.D.

Prof. Jano Moreira de Souza, Ph.D.

Prof. Marco Antônio Casanova, Ph.D.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2001

AMORIM, FERNANDA ARAUJO BAIÃO

Uma Metodologia com Algoritmos para o Projeto de Distribuição de Bases de Dados usando Revisão de Teorias [Rio de Janeiro] 2001

X, 57 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 2001)

Tese - Universidade Federal do Rio de Janeiro, COPPE

1. Projeto de Distribuição
2. Revisão de Teorias
3. Bancos de Dados

I. COPPE/UFRJ II. Título (série)

A meu marido

Eduardo Amorim,

e a minha avó

Ana

AGRADECIMENTOS

À Professora Marta Mattoso, pela orientação exemplar, carinho, seriedade e incentivo constantes durante o desenvolvimento deste trabalho.

Ao Professor Gerson Zaverucha, pela orientação e incentivo, e pelo seu acompanhamento e questionamentos constantes, fundamentais para o desenvolvimento deste trabalho.

Ao Professor Jude Shavlik, por suas sugestões e comentários valiosos durante o período do doutorado sanduíche em que trabalhamos juntos na Universidade de Wisconsin - Madison.

Aos membros da banca, professores Carol, Luiza, Sheila, Jano e Casanova, pela apreciação do trabalho, críticas construtivas e comentários incentivadores.

Ao Professor Jano Moreira de Souza, pela preocupação constante em fornecer condições favoráveis ao desenvolvimento dos trabalhos de todos os alunos, e pelas oportunidades que me foram dadas.

Ao Professor Marcos Roberto da Silva Borges, pelo primeiro incentivo à pesquisa através da orientação em Projetos de Iniciação Científica.

Ao Blaschek, professor e "guru", por sua amizade, pelas lições e experiências que me foram passadas, pela confiança depositada, por todas as oportunidades de crescimento profissional, e pela preocupação com o bem estar e motivação de todos a sua volta.

Aos demais professores e funcionários da COPPE Sistemas, em especial à Professora Sheila Veloso, pelo incentivo desde a época da graduação, às meninas da secretaria - Claudia, Solange, Sueli, Mercedes e Lúcia - e ao pessoal do laboratório - Inês, Guilherme, Fred, Carlos e Júlio Assunção - e carinhosamente à Patty e à Marilene, pela amizade e por contribuírem pacientemente na resolução de problemas e fornecerem condições favoráveis ao desenvolvimento deste trabalho e dos projetos COPPETEC em que participei.

À Melise e ao Márcio, pelo apoio e carinho constantes, fundamentais nos momentos mais angustiantes, à Renata e Yoko, pela força que me deram e por seus conselhos extremamente valiosos. Em especial à Carlete, por sua alegria, competência e disposição contagiantes, e pelo seu exemplo de força.

À todas as pessoas que trabalham nos projetos COPPETEC em que participei, por contribuírem para um ambiente de trabalho agradável, competente e motivador.

A todos os amigos da COPPE que acompanharam, incentivaram e colaboraram com esta minha empreitada desde a época do mestrado (Nivinha, Aninha, Rômulo, Gustavo Guedes, Fal, Gustavo Pinto, Leonardo Guerreiro e Rodrigo Salvador), em especial à Sandra Lino, velha companheira, pela sua amizade, ao Paulo Pires cujo companheirismo, clareza de idéias e alegria tornaram boa parte desta empreitada bastante agradável, e à Gabriela, Kate e André, pelo trabalho em conjunto.

Às pessoas que contribuíram durante a nossa estadia na Universidade de Wisconsin, em especial aos Professores David Page e Jeffrey Naughton pela disponibilidade para apreciação deste trabalho, experiência passada e ricos comentários. À Jussara e Luciana pela amizade e momentos de descontração, à Natassa pelo desprendimento, apoio e discussões sobre o trabalho, à Adriana, Zizi, Biel e Juju pela agradável convivência, amenizando o frio e a saudade.

Ao tio Gilson, pelo exemplo de competência no meio acadêmico, e aos meus avôs e avós por terem criado uma raiz forte.

À tia Wilma, por acreditar e pelas palavras carinhosas de incentivo, e pelas mensagens profundas e festas alegres que a cada ano fortalecem ainda mais a união da família, criando um ambiente de extrema alegria e harmonia.

Ao S. Walter, D. Adi, Nã e Roberto, por sempre acreditarem e confiarem. À Kamilla e Karina, por sua espontaneidade e carinho.

A meus pais, João Vianey e Maria das Graças, por todos os valores que me foram passados, pelo eterno e carinhoso apoio, pela confiança e incentivo, e por criarem um ambiente de amor, tranqüilidade e harmonia.

À minha irmã Sandra, pelo seu exemplo de competência e responsabilidade.

Ao CNPq, à FAPERJ e à Fundação COPPETEC, por financiarem este trabalho e outros eventos e trabalhos relacionados que muito contribuíram para a minha formação.

Ao meu marido e grande amor Eduardo, pelo seu amor, companheirismo e compreensão constantes, por todo incentivo e confiança, pela paciência nos momentos difíceis e por compartilhar durante toda a jornada, em todos os sentidos; e às minhas "filhas" Milla e Nini, pela fidelidade e por todos os momentos de alegria.

A Deus, por tudo.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

UMA METODOLOGIA COM ALGORITMOS PARA O PROJETO DE DISTRIBUIÇÃO DE BASES DE DADOS USANDO REVISÃO DE TEORIAS

Fernanda Araujo Baião Amorim

Dezembro/2001

Orientadores: Marta Lima de Queirós Mattoso
Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

O projeto de distribuição de bases de dados é responsável pela fragmentação e alocação de dados e programas aos nós de um sistema distribuído. A etapa de fragmentação agrupa, em fragmentos, informações que são acessadas simultaneamente pelas aplicações. O presente trabalho propõe uma infra-estrutura para a etapa de fragmentação do projeto de distribuição de bases de dados, utilizando o modelo orientado a objetos no nível conceitual de forma a captar a semântica da aplicação representada pelo usuário. A infra-estrutura proposta integra três módulos. O módulo heurístico define um conjunto de heurísticas para direcionar a fragmentação de classes, e as incorpora em uma metodologia detalhada que leva em conta um número abrangente de fatores relevantes. A metodologia proposta contempla um algoritmo de análise que escolhe a técnica de fragmentação mais adequada para cada classe, algoritmos para a fragmentação horizontal e vertical, além da formalização de conceitos e definições importantes para o modelo de objetos. O bom desempenho dos esquemas de fragmentação obtidos com a aplicação da metodologia foi comprovado por estudos experimentais. O módulo de revisão de teorias define uma estratégia para refinamento de algoritmos utilizando a técnica de inteligência artificial denominada revisão de teorias, que altera automaticamente o algoritmo de análise de forma a adequá-lo para a geração de novos esquemas de fragmentação apresentados como exemplos. Resultados experimentais mostram a aplicação desta estratégia obtendo um algoritmo de análise revisado que produz esquemas de fragmentação com desempenho superior. Finalmente, o módulo *branch-and-bound* apresenta uma estratégia alternativa para a fragmentação de classes, baseada na técnica de otimização de mesmo nome. Esta estratégia explora o espaço de soluções para o problema de forma inteligente, na busca do esquema de fragmentação que maximize o desempenho do sistema distribuído como um todo. Os esquemas de fragmentação obtidos pelo módulo *branch-and-bound* podem ainda representar exemplos para o módulo de revisão de teorias no refinamento do algoritmo de análise, incorporando desta forma informações adicionais ao módulo heurístico.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

A METHODOLOGY AND ALGORITHMS FOR THE DESIGN OF DISTRIBUTED DATABASES USING THEORY REVISION

Fernanda Araujo Baião Amorim

December/2001

Advisors: Marta Lima de Queirós Mattoso
Gerson Zaverucha

Department: Computer Science and Systems Engineering

The design of distributed databases involves making decisions on the fragmentation and placement of data and programs across the sites of a computer network. The fragmentation phase clusters in fragments the information accessed simultaneously by frequently executed applications. This work presents a framework to handle the class fragmentation problem during the design of distributed databases. The framework works in the conceptual level, and uses the object data model to capture the application semantics represented by the user. The proposed framework integrates three modules. The heuristic module defines a set of heuristics to drive the fragmentation of object databases, and incorporates them in a methodology that takes a large number of parameters into account. The proposed methodology includes an analysis algorithm that automatically chooses the most adequate fragmentation technique to be applied in each class, horizontal and vertical class fragmentation algorithms, and also several definitions and formalizations to handle the problem in the object model. Experiments using our methodology resulted in fragmentation schemas with better performance results when compared to other fragmentation schemas proposed in the literature. The theory revision module defines a knowledge-based approach to improve algorithms through the use of an artificial intelligence technique called theory revision, which automatically improves the analysis algorithm using fragmentation schemas with previously known performance presented as examples. Experimental results show the effectiveness of the approach in finding better fragmentation schemas with improved performance. Finally, the branch-and-bound module presents an alternative approach for class fragmentation through the use of an optimization technique. This approach performs an intelligent search for a fragmentation schema through the space of hypotheses. The fragmentation schema output from the branch-and-bound module may further represent examples for the theory revision module to improve the analysis algorithm, thus incorporating additional information to the heuristic module.

ÍNDICE DO TEXTO

ÍNDICE DE FIGURAS

Capítulo 1

INTRODUÇÃO

Este capítulo apresenta a motivação desta tese, a definição do problema tratado, os principais objetivos pretendidos e a organização do texto nos próximos capítulos.

1.1 MOTIVAÇÃO

Distribuição de dados e paralelismo em Sistemas Gerenciadores de Bases de Dados (SGBD) são duas das técnicas mais eficientes para o aumento de desempenho em aplicações que manipulam um grande volume de dados. No entanto, para maximizar o aumento de desempenho que pode ser alcançado, é essencial que a distribuição de dados seja realizada de forma adequada, visando reduzir ao máximo o acesso das aplicações às informações irrelevantes e a troca de dados entre os nós de um sistema distribuído, que são os objetivos do projeto de distribuição (ÖZSU e VALDURIEZ, 1999). O projeto de distribuição de bases de dados orientadas a objetos (PDBDOO) é um problema complexo, pelas seguintes razões: (i) a não existência de uma formalização consensual do problema na literatura, em especial do modelo de dados orientado a objetos; (ii) a existência de muitos parâmetros de entrada ao problema que devem ser levados em consideração; (iii) a existência de muitas diferenças entre os modelos relacional e orientado a objetos, que limitam uma adaptação direta dos algoritmos existentes; (iv) a existência de passos intermediários no problema com objetivos conflitantes; e (v) a necessidade de se obter estimativas e heurísticas para direcionar a solução do problema, que muitas vezes também são conflitantes quando combinadas.

A abordagem descendente para o projeto de distribuição de bases de dados divide-se em duas fases, denominadas fragmentação e alocação. A fragmentação de dados, escopo do presente trabalho, é responsável por agrupar, em fragmentos, informações que são acessadas simultaneamente pelas aplicações. A etapa de alocação, por sua vez, é responsável pelo armazenamento físico dos fragmentos de classe que foram gerados pela etapa de fragmentação entre os nós do sistema distribuído, e pela replicação de dados. Vale lembrar que o problema da replicação de dados em sistemas distribuídos relacionais comerciais ainda se apresenta resolvido de forma insatisfatória, utilizando *snapshots* ou transações de atualização de múltiplas cópias dos dados para o gerenciamento da replicação de dados, como dito por MOLINA e HSU (1995). Já nos sistemas OO este problema se encontra ainda mais incipiente.

Para a fragmentação de uma classe podem ser utilizadas duas técnicas básicas (fragmentação horizontal e fragmentação vertical). No modelo orientado a objetos, a fragmentação horizontal distribui as instâncias da classe entre os fragmentos, os quais

vão conter um subconjunto das instâncias da classe, compartilhando a mesma estrutura lógica (atributos e métodos). Por outro lado, a fragmentação vertical particiona a estrutura lógica da classe e a distribui entre os fragmentos, que vão portanto conter as mesmas instâncias, mas com estruturas lógicas diferentes. A fragmentação horizontal de uma classe pode ainda ser subdividida em primária e derivada. A fragmentação horizontal primária (PHF) visa otimizar operações sobre um conjunto de dados (como a busca na extensão de uma classe) através da redução dos dados que precisam ser acessados e do aumento do grau de paralelismo que pode ser aplicado durante a execução da operação. A fragmentação horizontal derivada, no entanto, visa otimizar operações de navegação entre os dados através do agrupamento no disco de objetos de classes diferentes que estejam relacionados. Também é possível aplicar as técnicas de fragmentação vertical e horizontal simultaneamente na mesma classe (fragmentação híbrida) ou em classes diferentes (fragmentação mista) do esquema.

Existem diversos trabalhos na literatura que endereçam o PDBDOO, incluindo BELLATRECHE *et al.* (2000), BELLATRECHE *et al.* (1998), BELLATRECHE *et al.* (1996), CHEN e SU (1996), EZEIFE e BARKER (1995), EZEIFE e BARKER (1998), KARLAPALEM *et al.* (1994), MAIER *et al.* (1994), SAVONNET *et al.* (1998). Entretanto, devido à complexidade do problema, a maioria destes trabalhos baseia-se em um subconjunto limitado de estimativas e heurísticas. Além disso, algumas abordagens trabalham sobre a base de dados instanciada, o que limita a sua aplicabilidade. É importante ressaltar também que os trabalhos relacionados limitam a sua análise a apenas uma das técnicas de fragmentação existentes (vertical ou horizontal) em todas as classes do esquema, resultando em esquemas de fragmentação puramente horizontal ou puramente vertical.

Nossos trabalhos anteriores (BAIÃO, 1997, BAIÃO e MATTOSO, 1997) evidenciaram os benefícios alcançados com as técnicas de fragmentação mista e híbrida no aumento de desempenho das aplicações avaliadas. Estes trabalhos também motivaram a análise de diversos parâmetros durante o projeto de distribuição que não são considerados nos trabalhos relacionados na literatura.

1.2 OBJETIVOS

O presente trabalho propõe uma infra-estrutura para tratar o problema da fragmentação de classes durante o projeto de distribuição de bases de dados. A infra-estrutura proposta trabalha no nível conceitual, utilizando o modelo orientado a objetos a fim de captar a semântica das aplicações como representadas pelo usuário, e integra três módulos: o módulo heurístico, o módulo de revisão de teorias e o módulo *branch-and-bound* (figura 1.1).

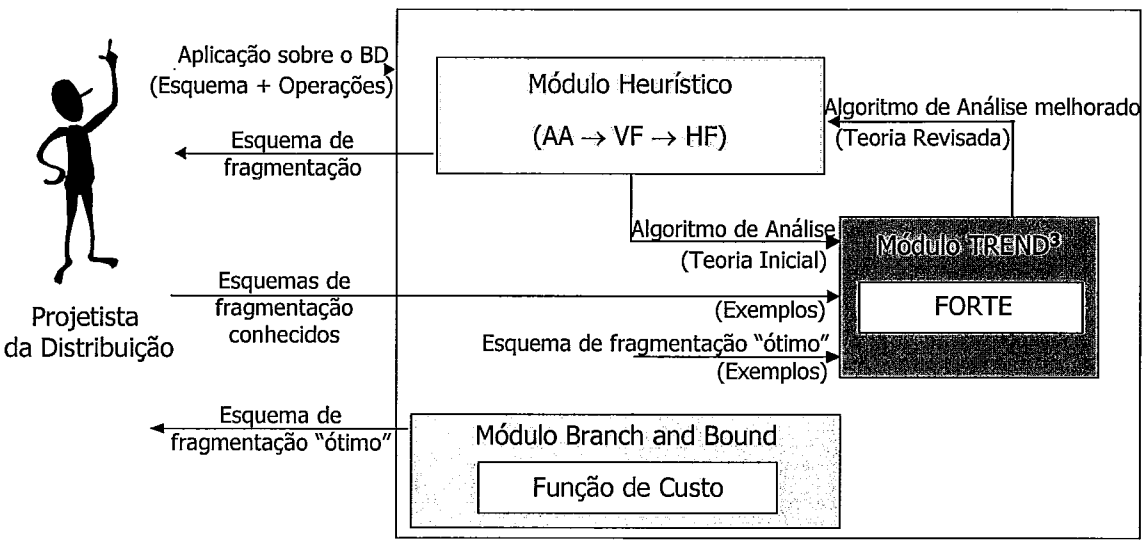


Figura 1.1: Infra-estrutura para o projeto de distribuição de bases de dados

O módulo heurístico define um conjunto de heurísticas para direcionar a fragmentação de classes, e as incorpora em uma metodologia detalhada que inclui um algoritmo de análise, um algoritmo de fragmentação vertical e um algoritmo de fragmentação horizontal. A metodologia reúne idéias e resultados obtidos em estágios anteriores desta tese (BAIÃO, 1997, BAIÃO e MATTOSO, 1997, 1998, BAIÃO *et al.*, 1998, 2000, 2001), define uma classificação "dono-membro" para as classes de um esquema e diversas características do modelo orientado a objetos relevantes para o problema. Além disso, a metodologia considera muitas características do modelo orientado a objetos, e parâmetros e heurísticas adicionais quando comparado a nosso trabalho anterior (BAIÃO, 1997). Uma característica importante da metodologia é a existência da etapa de análise para auxiliar o projetista da distribuição durante o processo de fragmentação. A etapa de análise considera informações sobre o esquema do banco de dados e sobre as aplicações a fim de indicar a técnica de fragmentação mais adequada para cada classe do esquema. Nossos resultados experimentais anteriores

(BAIÃO *et al.*, 2000) mostram situações de conflito na escolha entre fragmentação horizontal primária ou derivada para determinadas classes, no entanto outros trabalhos relacionados na literatura não tratam este problema. No presente trabalho, cada classe a ser fragmentada é cuidadosamente analisada levando-se em conta seus relacionamentos e cardinalidades, assim como a freqüência de execução das aplicações. Desta forma, a fragmentação escolhida reflete a estrutura de caminhos de navegação acessados pelas aplicações mais freqüentes, que nem sempre são intuitivos. A eficiência dos esquemas de fragmentação resultantes foi comprovada em resultados experimentais (BAIÃO *et al.*, 2000) sobre o benchmark 007 (CAREY *et al.*, 1993).

O módulo de revisão de teorias define uma abordagem baseada em conhecimento para refinamento de algoritmos, em especial do algoritmo da etapa de análise do módulo heurístico. Esta abordagem utiliza técnicas de inteligência artificial (revisão de teorias) com o objetivo de automaticamente aumentar a eficiência do algoritmo revisado, através de um procedimento de refinamento. A estratégia foi implementada em uma ferramenta denominada Theory REvisioN on the Design of Distributed Databases (TREND3), cujo objetivo é automaticamente modificar o algoritmo de análise de forma que ele passe a obter soluções com desempenho ainda melhor do que o encontrado inicialmente. O procedimento de revisão de teorias é então aplicado apresentando-se uma implementação em Prolog do algoritmo de análise (como teoria inicial do domínio) e um conjunto de esquemas de fragmentação com desempenhos previamente conhecidos (como conjunto de exemplos). Resultados preliminares comprovaram a viabilidade desta abordagem proposta, obtendo um algoritmo de análise revisado que produz esquemas de fragmentação com melhor desempenho que o algoritmo original (BAIÃO, 2002).

O módulo *branch-and-bound* apresenta uma abordagem alternativa para o projeto de distribuição, utilizando técnicas de otimização para realizar uma busca inteligente, através do espaço de soluções viáveis, pelo esquema de fragmentação ótimo, ou seja, que maximize o desempenho da aplicação como um todo.

1.3 ORGANIZAÇÃO DO TEXTO

O restante deste trabalho está organizado da seguinte forma. O capítulo 2 apresenta definições existentes na literatura relevantes para o problema do projeto de distribuição

e um breve resumo de trabalhos relacionados. O capítulo 3 relaciona os principais aspectos que devem ser levados em conta no PDBDOO, define a classificação "dono-membro" de classes e propõe heurísticas para direcionar o PDBDOO. No capítulo 4 é apresentado o módulo heurístico, através da proposta de uma metodologia e seus algoritmos para o PDBDOO. O capítulo 5 apresenta o módulo de revisão de teorias, propondo uma abordagem baseada em conhecimento para refinamento do algoritmo de análise. O capítulo 6 apresenta o módulo *branch-and-bound*, uma abordagem alternativa para o PDBDOO através de técnicas de otimização. Finalmente, o capítulo 7 conclui este trabalho.

Para maiores detalhes relacionados à proposta desta tese recomendamos a leitura de (BAIÃO, 2002).

Capítulo 2

PROJETO DE DISTRIBUIÇÃO DE BASES DE DADOS ORIENTADAS A OBJETOS

Este capítulo apresenta uma caracterização do problema tratado nesta tese, e descreve o estado da arte na área.

2.1 APRESENTAÇÃO DO PROBLEMA

O projeto de distribuição, segundo a definição de ÖZSU e VALDURIEZ (1999), define como será o armazenamento dos dados entre os nós da rede do sistema distribuído. Na estratégia descendente, o projeto é composto por duas fases: fragmentação e alocação.

A etapa de fragmentação é responsável pelo agrupamento, em fragmentos, das informações que são acessadas por uma consulta ou transação executada sobre a base de dados, enquanto que a etapa de alocação distribui os fragmentos gerados entre os nós do sistema, e trata da replicação de dados.

No modelo de dados relacional, a preocupação do projetista se limita em distribuir os atributos das relações pertencentes ao esquema de dados. Com a utilização do modelo orientado a objetos, além dos atributos das classes, outros fatores devem ser levados em consideração, como: a existência de métodos, relacionamentos entre as classes e suas cardinalidades, atributos multi-valorados e relacionamentos de herança.

Existem dois tipos básicos de fragmentação de uma classe (ÖZSU e VALDURIEZ, 1999): fragmentação vertical (que divide a estrutura lógica – atributos e métodos – da classe, distribuindo-a entre os fragmentos, de modo que os fragmentos conterão os mesmos objetos, mas com estruturas diferentes) e fragmentação horizontal (que distribui as instâncias de uma classe em diversos fragmentos, os quais terão a mesma estrutura mas conteúdo diferente). Há também a fragmentação híbrida, que combina as fragmentações vertical e horizontal na mesma classe. A importância da fragmentação híbrida foi ressaltada em (NAVATHE *et al.*, 1995, ÖZSU e VALDURIEZ, 1999, SU *et al.*, 1998) e é tratada nesta tese, assim como em nossos trabalhos anteriores (BAIÃO, 1997, BAIÃO e MATTOSO, 1997, 1998, BAIÃO *et al.*, 1998, 2000, 2001)

A fragmentação horizontal pode ser ainda subdividida em primária e derivada. Para isso, geralmente definem-se as classes do esquema em "dono" ou "membro". A fragmentação horizontal primária (PHF) é aplicada sobre as classes "dono", enquanto a fragmentação horizontal derivada (DHF) é aplicada sobre as classes "membro" de acordo com a fragmentação da sua classe "dono". A classificação "dono-membro" para as classes de um esquema orientado a objetos é um problema importante e complexo, embora ainda não tenha sido tratada com profundidade na literatura, que deve ser

considerado durante a fragmentação horizontal de uma base de dados. A definição de classes "dono" e "membro" é discutida em maiores detalhes em nossos trabalhos relacionados (BAIÃO, 2002, BAIÃO *et al.*, 2001). Existe uma outra técnica de fragmentação mencionada na literatura, denominada "partição de caminhos" (*path partitioning*). No entanto, no contexto desta tese, esta técnica será tratada como um caso especial da fragmentação horizontal derivada, aplicável em situações em que existem relacionamentos "todo-parte" entre as classes do esquema.

Uma das maiores dificuldades do projeto de distribuição de bases de dados no modelo orientado a objetos é a existência de operações tanto sobre conjuntos quanto de navegação entre as classes, as quais definem padrões de acesso bastante distintos aos dados armazenados (acesso associativo *versus* acesso navegacional). Esta diferença nos padrões de acesso das operações dificulta muito a tarefa de agrupamento de objetos que deve ser realizada pelo SGBDOO, e considerada pelo projetista da distribuição. Isto porque, em SGBDOOs, é muito comum combinar várias políticas de agrupamento de objetos em uma mesma base de dados, uma para cada classe, e esta é uma decisão complexa (MATTOSO *et al.*, 2001). Um agrupamento representa o armazenamento de objetos em áreas contíguas no disco, segundo um determinado critério, de modo a minimizar o número de páginas ocupadas e aumentar o desempenho das operações que acessam a base (SHRUFI, 1994). Duas das políticas mais comuns de agrupamento são o *agrupamento por extensão* (que reúne objetos de uma mesma classe em áreas contíguas no disco) e o *agrupamento por referência* (que armazena juntos os objetos fisicamente relacionados entre si).

Alguns trabalhos na literatura utilizam técnicas de agrupamento de objetos para aumentar o desempenho de aplicações sobre bancos de dados centralizados (GARDARIN *et al.*, 1995). Uma contribuição importante do presente trabalho é ressaltar a ligação que pode ser estabelecida entre as políticas de armazenamento existentes e o processo de fragmentação de classes, desta forma possibilitando a aplicação de técnicas advindas desta área relacionada. Enquanto DHF privilegia o acesso navegacional (partindo de um objeto composto para seus componentes, por exemplo), a fragmentação vertical favorece o acesso associativo a uma extensão de classe e o uso individualizado de atributos e métodos, através da redução de dados irrelevantes acessados pelas operações. Desta forma, a fim de evitarmos a geração de esquemas de fragmentação que podem levar à execução de consultas com baixo

desempenho, ambas as técnicas (horizontal e vertical) devem ser consideradas na etapa de fragmentação, e combinadas em classes distintas do esquema (o que chamamos de fragmentação mista) ou na mesma classe (o que chamamos de fragmentação híbrida). No entanto, esta escolha entre diferentes técnicas não é simples de ser tomada pelo projetista da distribuição, por isso é importante a proposta de uma metodologia que auxilie esta escolha, como ressaltado em (ÖZSU e VALDURIEZ, 1999).

2.2 TRABALHOS RELACIONADOS

Existem na literatura muitos trabalhos que tratam do projeto de distribuição no modelo de dados relacional, incluindo (CERI e NAVATHE, 1983, MOLINA e HSU, 1995, NAVATHE e RA, 1989, NAVATHE *et al.*, 1995, ÖZSU e VALDURIEZ, 1999). No modelo de dados orientado a objetos, muitos trabalhos também evidenciam a importância do projeto de distribuição para o aumento de desempenho de aplicações que manipulam um grande volume de dados (KARLAPALEM *et al.*, 1994, KARLAPALEM e LI, 2000, MAIER *et al.*, 1994). Um breve resumo destes trabalhos pode ser encontrado em (BAIÃO, 2002).

No contexto da fragmentação horizontal, os trabalhos de (SAVONNET *et al.*, 1998, BELLATRECHE *et al.*, 1998, 2000 e EZEIFE e BARKER, 1995) apresentam propostas distintas para obtenção de esquemas de fragmentação horizontal para uma base de dados.

Os trabalhos mais relevantes que abordam a fragmentação vertical de classes são os de BELLATRECHE *et al.* (1996) e EZEIFE e BARKER (1998), cuja análise também pode ser encontrada em (BAIÃO, 2002).

A combinação das duas técnicas básicas de fragmentação aplicadas em um mesmo esquema de uma base de dados orientada a objetos foi inicialmente proposta em (BAIÃO, 1997, BAIÃO e MATTOSO, 1997), que apresentaram uma estratégia com uma etapa de análise para escolha da técnica de fragmentação mais adequada a cada classe do modelo, além de etapas de fragmentação horizontal e fragmentação vertical. Uma segunda versão da etapa de análise foi apresentada em (BAIÃO *et al.*, 1998), e mais recentemente a etapa de fragmentação horizontal foi estendida e validada em (BAIÃO *et al.*, 2000) através de estudos experimentais adicionais. O presente trabalho é uma evolução de todos os nossos trabalhos anteriores, e apresenta uma metodologia

completa para o PDBDOO, assim como novas abordagens para o problema. Maiores detalhes sobre a metodologia proposta podem ser encontrados no capítulo 4 e em (BAIÃO *et al.*, 2001), enquanto que o capítulo 6 e o trabalho (BAIÃO, 2002) descrevem mais detalhadamente a nova abordagem proposta para o problema.

As principais características das estratégias de fragmentação horizontal e/ou vertical mencionadas neste capítulo são detalhadamente analisadas em (BAIÃO, 2002) e sumarizadas nas tabelas 2.1 e 2.2

Tabela 2.1: Trabalhos relacionados sobre fragmentação horizontal de classes

Características	Savonnet *et al.* (1998)	Bellatreche *et al.* (1998)	Ezeife & Barker (1995)	Baiao *et al.* (1998)	Baiao *et al.* (2000)
Trata PHF	√	√	√	√	√
Trata DHF	√		√	√	√
Analisa adequação da classe para fragmentação		√		√	√
Considera os relacionamentos definidos no esquema da base	√		√	√	√
Considera navegações embutidas nos métodos	√		√	√	√
Considera navegações das consultas definidas pelo usuário			√	√	√
Considera acesso associativo a uma extensão de classe	√	√	√	√	√
Considera freqüência de execução de aplicações	√		√	√	√
Trabalha independentemente da implementação do SGBD, e não requer uma base de dados instanciada	√	√		√	√
Considera definição "dono-membro" para classes no modelo de dados OO					√
Apresenta algoritmos		√	√	√	√
Apresenta resultados experimentais					√

Tabela 2.2: Trabalhos relacionados sobre fragmentação vertical de classes

Características	Bellatreche *et al.* (1996)	Ezeife & Barker (1998)	Baiao *et al.* (1998)
Analisa adequação da classe para fragmentação			√
Considera chamada de métodos a outros métodos		√	√
Considera consultas definidas pelo usuário	√	√	√
Considera tanto atributos quanto métodos da classe		√	√
Considera freqüência de execução de aplicações	√	√	√
Trabalha independentemente da implementação do SGBD, e não requer uma base de dados instanciada	√		√
Considera os relacionamentos definidos no esquema da base			√
Apresenta algoritmos		√	√

Capítulo 3

HEURÍSTICAS PARA O PDBDOO

Este capítulo apresenta os principais aspectos que são considerados nesta tese para o projeto de distribuição de bases de dados orientadas a objetos, e define as heurísticas que vão direcionar a etapa de análise da metodologia.

3.1 ASPECTOS QUE DEVEM SER CONSIDERADOS

Existem muitos aspectos que devem ser considerados durante o projeto de distribuição, a fim de escolher a técnica de fragmentação mais adequada para cada classe e, desta forma, obter o melhor esquema de fragmentação possível. Este capítulo classifica tais aspectos em três grandes grupos que merecem uma atenção especial do projetista de distribuição, pois exercem grande influência na qualidade da distribuição dos dados e podem, desta forma, degradar o desempenho do sistema como um todo.

- **características das aplicações**, ou seja, as operações que são executadas sobre os dados distribuídos, e suas freqüências de execução;
- **semântica do modelo de dados**, representada pelo conjunto de classes do esquema (incluindo a definição de seus atributos e métodos) e relacionamentos entre elas;
- **informação quantitativa** como a existência de coleções de objetos definidas pelo projetista lógico e suas cardinalidades estimadas.

Estes três grandes grupos representam um conjunto abrangente de aspectos relevantes que são necessários para a obtenção de um bom esquema de fragmentação, e que por isso são tratados nesta tese. É importante ressaltar que os trabalhos relacionados na literatura limitam-se à análise de apenas um subconjunto destes aspectos. A seção 3.2 apresenta cada grupo de aspectos tratado nesta tese mais detalhadamente, enquanto que a seção 3.3 evidencia a influência exercida por cada grupo no projeto de distribuição, através da definição de heurísticas para a fragmentação de classes que direcionarão a etapa de análise da metodologia proposta no capítulo 4.

3.2 INFORMAÇÕES RELEVANTES

A etapa de aquisição de informações relevantes para a realização do PDBDOO é encapsulada no que denominamos "Módulo de Interface" (*Interface Module*). O objetivo do Módulo de Interface é adquirir as informações relevantes e torná-las disponíveis para as etapas seguintes da metodologia. A aquisição de informações pode ser realizada tanto através de consultas ao esquema conceitual global da base de dados (que armazena definições das classes e relacionamentos, existência de extensões de classes e aplicações definidas durante a fase de projeto lógico do banco de dados), quanto através de consultas ao próprio projetista lógico do banco de dados (já que

informações como tamanho estimado das extensões de classe e freqüência de execução das aplicações nem sempre estarão disponíveis no esquema conceitual). As informações relevantes que devem ser adquiridas relacionadas às aplicações, à semântica do modelo de dados e dados quantitativos são definidas a seguir.

3.2.1 APLICAÇÕES

Para aumentar o desempenho das aplicações executadas sobre a base de dados distribuída, o projetista da distribuição deve ter acesso a informações como as freqüências de execução, os atributos e métodos de classes acessados e os predicados inseridos em cada uma delas.

No modelo relacional, estas informações são representadas por um conjunto de predicados simples e completos, que podem ser extraídos com a aplicação do algoritmo COM_MIN (ÖZSU e VALDURIEZ, 1999). No modelo orientado a objetos, no entanto, esta decomposição das aplicações em predicados simples não é tão fácil. Primeiro, porque objetos não são estruturas simples, pois podem referenciar recursivamente outros objetos. Segundo, porque um componente de um objeto pode ser uma estrutura aninhada. Finalmente, porque as consultas e transações orientadas a objetos podem conter também chamadas de métodos entre as classes.

Desta forma, as aplicações devem ser decompostas em estruturas menores, denominadas operações, que contenham toda a informação necessária para a escolha da fragmentação das classes acessadas. Sobre estas operações serão definidas heurísticas para as etapas de fragmentação. Em (BAIÃO, 2002), uma operação é definida como "um acesso a uma ou mais classes de um banco de dados durante uma transação", onde uma transação pode ser composta por consultas e/ou chamadas de métodos. O trabalho apresenta uma técnica de análise e decomposição de uma aplicação em três tipos de operação possíveis: predicados simples, expressões de caminho ou acessos isolados à uma classe (projeções de atributos ou chamadas de métodos existentes na formação do resultado da consulta). De acordo com as idéias apresentadas, a consulta:

```
select tuple(s.name, s.grade, s.department.location) from s in Students where s.GPA > 2.5
```

pode ser decomposta nas seguintes operações:

i. predicado simples: `GPA > 2.5`

ii. expressão de caminho: `s.department.location`

iii. acessos isolados: s.name, s.grade

Ainda de acordo com (BAIÃO, 2002), uma operação é denominada **operação de seleção** (no caso de predicados simples), **operação de projeção** (no caso de acessos isolados), ou **operação de navegação** (no caso de expressões de caminho).

As informações relevantes às aplicações reúnem, para cada operação extraída:

- sua classificação (seleção, projeção ou navegação);
- as classes acessadas;
- uma estimativa da freqüência de execução (número entre 0 e 100), que determinará a prioridade da análise da operação no processo de fragmentação.

3.2.2 SEMÂNTICA DO MODELO DE DADOS

Para realizar o projeto de distribuição de uma base de dados, é essencial que o projetista da distribuição tenha conhecimento sobre o esquema conceitual do banco. No modelo de dados orientado a objetos que é assumido no contexto deste trabalho (definido em (BAIÃO, 2002)), o esquema conceitual contém informações como a definição das classes – atributos e métodos – e dos relacionamentos entre as classes.

A visualização das classes, seus atributos e métodos complexos, e dos relacionamentos entre as classes pode ser feita através de uma adaptação do **grafo de dependências entre classes (GDC)**, definido em (CERI e NAVATHE, 1983) e mais recentemente tratado em (EZEIFE e BARKER, 1995). No GDC, os nós representam as classes do esquema e as arestas representam ligações físicas (definidas pelos atributos complexos, e implementadas através de ponteiros) ou lógicas (definidas pelas chamadas de métodos) entre duas classes. A cardinalidade das ligações também é representada ("1:1", "1:N", "N:1" ou "N:M"), e reflete a cardinalidade dos relacionamentos e/ou o número de objetos acessados pelas chamadas dos métodos. Desta forma, a representação de uma ligação L no GDC é dada por:

L = (A, B, name, cardinality)

Onde A e B são as classes relacionadas, name é uma *string* identificando a ligação, e cardinality é a cardinalidade da ligação. Maiores detalhes sobre o GDC podem ser encontrados em (BAIÃO, 2002)

Tradicionalmente, a maioria dos trabalhos na literatura utiliza a classificação "dono-membro" (*owner-member*) entre as classes do esquema para direcionar a escolha entre

fragmentação horizontal primária e fragmentação horizontal derivada de cada classe (ÖZSU e VALDURIEZ, 1999), onde a PHF é aplicada sobre as classes "dono", enquanto a DHF é aplicada sobre as classes "membro" de acordo com a fragmentação da sua classe "dono". No modelo relacional esta classificação é bastante simples e pode ser realizada de forma direta a partir das ligações definidas pelos relacionamentos entre as tabelas. No modelo orientado a objetos, no entanto, a classificação "dono-membro" não é tão simples devido a fatores como a representação direta de relacionamentos muitos-para-muitos (N:M). Em (BAIÃO, 2002, BAIÃO *et al.*, 2001) é proposta uma nova representação para a classificação de classes "dono-membro" e uma estratégia para a sua construção a partir do GDC. A representação proposta trata aspectos do modelo OO como: relacionamentos de herança, relacionamentos N:M e N:1, de forma a minimizar o número de ligações que devem ser consideradas e evitar o compartilhamento de objetos da classe "membro" em relação a sua classe "dono", o que poderia dificultar a posterior definição de fragmentos horizontais derivados. A partir desta classificação, é proposta uma lista denominada *frag(owner,member)*, a fim de que sejam selecionadas as classes "dono" e "membro" para a etapa de fragmentação, onde para cada classe "membro" será definida apenas uma classe "dono". A lista *frag(owner,member)* é uma lista de pares da forma (X, Y) ou (X, null) (onde X e Y são classes do esquema), que será utilizada para direcionar a fragmentação de classes na metodologia proposta no capítulo 4. A definição completa encontra-se em (BAIÃO, 2002).

3.2.3 INFORMAÇÕES QUANTITATIVAS

Pelo fato de, no modelo de dados orientado a objetos, os conceitos de "classe" e "extensão de classe" possuírem semânticas distintas e complementares (descritas mais detalhadamente em (BAIÃO, 2002), informações quantitativas sobre a existência de coleções que implementam a extensão de uma classe tornam-se bastante relevantes para o PDBDOO.

As informações quantitativas relevantes reúnem, para cada classe:

- a existência ou não de uma coleção que implementa a extensão da classe
- o tamanho estimado da coleção (pequena, média, grande) quando comparada a outras classes do esquema

A classificação das coleções em "pequena", "média" ou "grande" pressupõe a existência de conceitos já definidos ou de métricas que direcionem esta definição. De fato, não há um consenso na literatura para tais conceitos, que são extremamente relacionados às características das aplicações e do esquema conceitual que está sendo considerado, e podem evoluir com o tempo. Este, portanto, é um problema em aberto na literatura. Em (BAIÃO *et al.*, 2001), há uma proposta de definição de tais conceitos levando em conta o tamanho da maior coleção, que será utilizada neste trabalho.

3.3 HEURÍSTICAS PARA O PDBDOO

Baseando-se nas informações relevantes para o PDBDOO descritas na seção 3.2 e em resultados experimentais previamente obtidos (LIMA e MATTOSO, 1996, MEYER e MATTOSO, 1998, TAVARES *et al.*, 2000), é possível definir heurísticas para reduzir o espaço de busca para o problema do PDBDOO. Esta seção apresenta um breve resumo das heurísticas propostas em (BAIÃO, 2002), onde cada uma delas é analisada e justificada.

3.3.1 APLICAÇÕES

A classificação das operações apresentada na seção 3.2.1 nos permite definir as seguintes heurísticas, que seguem os resultados experimentais obtidos em (LIMA E MATTOSO, 1996, MEYER e MATTOSO, 1998, TAVARES *et al.*, 2000):

No caso de uma operação de projeção sobre uma classe com cardinalidade grande ou média, esta classe é indicada para fragmentação vertical (VF)

No caso de uma operação de seleção sobre uma classe, esta classe é indicada para fragmentação horizontal primária (PHF)

No caso de uma operação de navegação, classes "não-raiz" do caminho de navegação são indicadas para fragmentação horizontal derivada (DHF) em relação à classe que a precede no caminho

3.3.2 SEMÂNTICA DO MODELO DE DADOS

A lista *frag(owner,member)* proposta em (BAIÃO *et al.*, 2001) é utilizada para refinar as heurísticas propostas na seção anterior através de um novo grupo de heurísticas:

Classes "apenas-dono" na lista frag(owner, member) são indicadas para fragmentação horizontal primária

Um par (X,null) na lista frag(owner, member) indica a classe X a fragmentação vertical, fragmentação horizontal primária ou fragmentação híbrida, de acordo com a classificação da operação que gerou o par (X,null): operações de projeção direcionam VF, operações de seleção direcionam PHF, e a existência dos dois tipos de operação direciona fragmentação híbrida

3.3.3 INFORMAÇÕES QUANTITATIVAS

A influência de informações quantitativas consideradas em estudos experimentais resultaram nas seguintes heurísticas:

Classes com cardinalidade média ou grande que possuam uma coleção implementando a sua extensão devem ser fragmentadas.

Classes com cardinalidade pequena não devem ser fragmentadas verticalmente.

Classes abstratas (que não possuam instâncias) não devem ser fragmentadas

Todas as heurísticas definidas para direcionar a fragmentação de classes durante o projeto de distribuição de bases de dados orientadas a objetos estão detalhadamente apresentadas e analisadas em (BAIÃO, 2002).

Capítulo 4

UMA METODOLOGIA E ALGORITMOS PARA O PROJETO DE DISTRIBUIÇÃO DE BASES DE DADOS ORIENTADAS A OBJETOS

Este capítulo apresenta o módulo heurístico da infra-estrutura proposta, contribuição principal desta tese, representado por uma metodologia para o projeto de distribuição de bases de dados orientadas a objetos.

4.1 MOTIVAÇÃO E VISÃO GERAL

A maioria dos trabalhos da literatura sobre projeto de distribuição no modelo orientado a objetos tratam apenas de uma técnica de fragmentação (horizontal ou vertical). Por causa disso, não abordam a escolha da melhor técnica de fragmentação a ser aplicada a uma classe, e não analisam a adequação de cada classe ao processo de fragmentação. Isto leva à aplicação da mesma técnica em todas as classes. Apesar de tais trabalhos mostrarem resultados (não experimentais) de melhoria de desempenho da base de dados distribuída quando comparada à base de dados centralizada, esta melhoria de desempenho é limitada na medida em que as consultas executadas não se beneficiam das vantagens das duas técnicas de fragmentação.

Desta forma, existe a necessidade de uma estratégia completa para o projeto de distribuição de bases de dados orientadas a objetos, como evidenciado em (ÖZSU e VALDURIEZ, 1999, MOLINA e HSU, 1995). Esta estratégia se faz ainda mais importante porque a decisão entre todos os tipos possíveis de fragmentação não é trivial em um ambiente genérico, levando-se em conta uma série de aplicações que executarão sobre os dados. Isto porque a combinação entre estas diversas operações e todos os parâmetros relevantes ao problema não levam o projetista da distribuição diretamente ao esquema de fragmentação com melhor desempenho. Especialmente em aplicações complexas do mundo real, a dificuldade na tomada das decisões de projeto é ainda maior. A maior dificuldade é decidir quais classes devem ser fragmentadas horizontalmente e/ou verticalmente. Ainda, na fragmentação horizontal as classes podem seguir para fragmentação horizontal primária ou derivada, e esta combinação com a fragmentação vertical pode, em alguns casos, ser conflitante e tornar inviável a implementação do esquema de fragmentação.

De modo a tratar este problema, este trabalho propõe uma metodologia completa para o projeto de distribuição de bases de dados orientadas a objetos que incorpora uma etapa de análise direcionando a escolha da técnica de fragmentação para cada classe, e três algoritmos. Estes algoritmos geram automaticamente um esquema de fragmentação adequado e eficiente a partir das informações fornecidas como entrada (descritas no capítulo 3). O esquema resultante pode combinar fragmentação horizontal, vertical e híbrida em diferentes classes, a partir das heurísticas definidas na seção 3.3.

Este capítulo descreve a metodologia proposta e os algoritmos que foram desenvolvidos para as etapas de análise, de fragmentação vertical e de fragmentação horizontal. Os algoritmos de fragmentação vertical e horizontal apresentados nas seções 4.3 e 4.4 são garantidamente corretos, ou seja, geram esquemas de fragmentação que respeitam as regras de completude, reconstrução e disjunção definidas originalmente em (ÖZSU e VALDURIEZ, 1999) e estendidas em (BAIÃO *et al.*, 2001a) para o modelo OO. A prova disto encontra-se em (BAIÃO, 2002).

A figura 4.1 apresenta uma visão geral da metodologia e ilustra o fluxo de dados (representados pelas elipses) entre as suas etapas (representadas pelos retângulos).

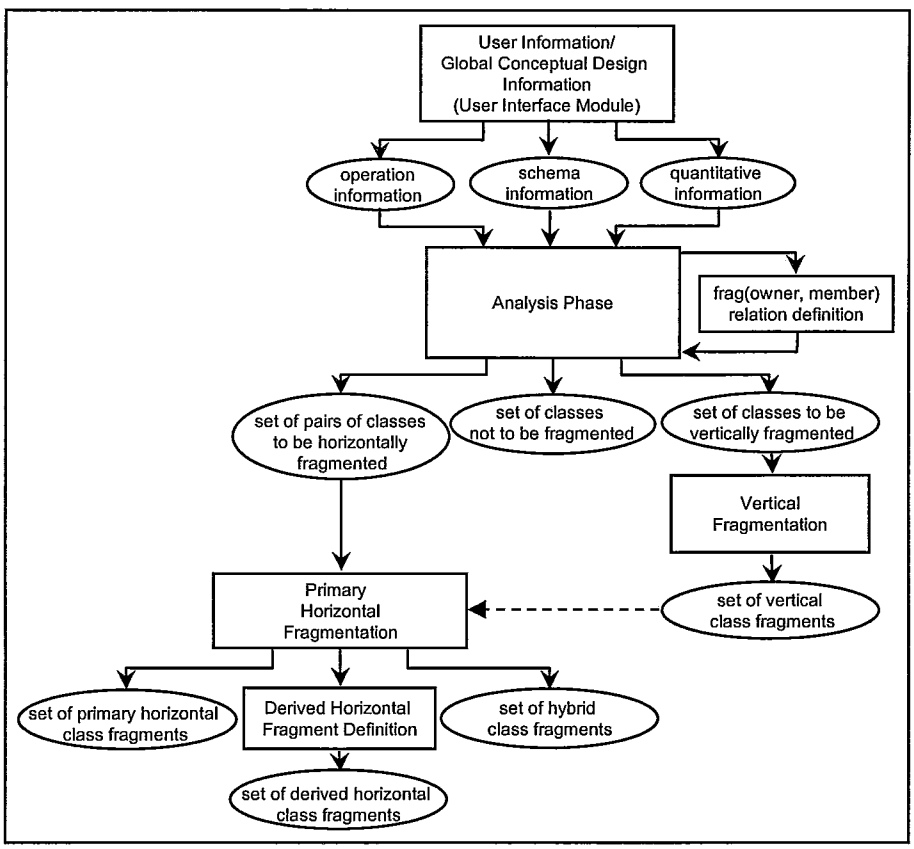


Figura 4.1: Visão geral da metodologia para PDBDOO

4.2 ETAPA DE ANÁLISE

A primeira etapa da metodologia é considerada por nós a mais importante, já que direciona a execução das outras duas. Esta etapa analisa informações adquiridas pelo módulo de interface (descrito em 3.2) e usa as heurísticas da seção 3.3 para decidir a técnica de fragmentação mais adequada (nenhuma, horizontal – primária ou derivada – e/ou vertical) para cada classe, conforme detalhado em (BAIÃO, 2002).

A idéia geral do algoritmo apresentado na figura 4.2 é que operações de seleção sobre classes com cardinalidade grande/média direcionam fragmentação horizontal primária, enquanto operações de projeção direcionam fragmentação vertical. Para as operações de navegação, a lista *frag(owner,member)* é utilizada a fim de decidir entre fragmentação horizontal primária ou derivada: classes "apenas-dono" seguem para fragmentação horizontal primária e classes "membro" serão fragmentadas de forma derivada em relação ao seu "dono". A definição completa da lista *frag(owner,member)* é apresentada em (BAIÃO, 2002) e sua implementação ilustrada na figura 4.3.

O resultado do algoritmo de análise são os conjuntos Ch – com os pares de classes para fragmentação horizontal primária e derivada, Cv – com as classes para fragmentação vertical, e Cn – com as classes que não foram indicadas para as fragmentações anteriores. Ainda, as classes que estiverem tanto em Ch quanto em Cv seguirão para fragmentação híbrida. Neste caso, a fragmentação horizontal será aplicada sobre os fragmentos verticais adequados da classe.

```
function AnalysisPhase ( C : set of (className, classcardinality),
                         O : set of (operationName, operationFrequency,
                                     operationClassification, accessedClass),
                         CDG: class dependency graph )
returns Ch : set of pairs (owner, member) of classes to be horizontally fragmented
        Cv : set of classes to be vertically fragmented
        Cn : set of classes not to be fragmented
var
  fragOwnerMember: set of pairs (owner, member) of classes with no replicas
begin
  Cn = C; fragOwnerMember = empty set                                    (1)
  sort O in descending order according to the operation frequency        (2)
  for each (Oi, freqOi, classificationOi, C(Oi)) in O do                 (3)
    case classificationOi of
      "projection":
          if ( C(Oi) is not an abstract class ) and                      (4)
            ( C(Oi) is not a member in fragOwnerMember ) and             (5)
            ( ((C(Oi), "large") ∈ C) or ((C(Oi), "medium") ∈ C) ) then   (6)
              fragOwnerMember += (C(oi), null)                           (7)
              Cv += C(Oi) ; Cn  -= (C(Oi), "large")                      (8)
        end if
      "selection":
          if ( C(Oi) is not an abstract class ) and                      (9)
            ( C(Oi) is not a member in fragOwnerMember )  then           (10)
              fragOwnerMember += (C(oi), null)                           (11)
              Ch += (C(Oi), null) ; Cn  -= (C(Oi), cardOi)               (12)
          end if
      "navigation":
          for each pair (X,Y) of consecutive classes in C(Oi) do         (13)
              if (X is not an abstract class) and (Y is not an abstract class) then   (14)
                (own, mem) = DefineOwnerMember (Oi, X, Y, Cv, CDG)       (15)
                if (own, mem) is not null then                           (16)
                    fragOwnerMember += (own, mem)                        (17)
                    Ch += (own, mem); Cn -= (own, cardO); Cn -= (mem, cardM)   (18)
                end if
              end if
          end for
    end case
  end for
  return Ch, Cv, Cn
end
```

Figura 4.2: Etapa de análise

```
function DefineOwnerMember ( Oi    : current operation,
                             X, Y  : classes in the schema accessed by Oi,
                             Cv    : current set of classes to be vertically fragmented,
                             CDG   : class dependency graph )
returns (own, mem) : pair of classes in the frag(owner, member) relation
begin
  (own, mem) = null;
  Li = the link (X, Y, name, card) in CDG through which Oi navigates from X to Y   (1)
  case card of
    "1:1", "1:N":                                                                  (2)
        if Y is not in Cv then (own, mem) = (X, Y)  end if                         (3)
    "N:1":                                                                         (4)
        if X is not in Cv then (own, mem) = (Y, X)  end if                         (5)
    "M:N":                                                                         (6)
        (own, mem) = null                                                          (7)
  end case
end
```

Figura 4.3: Definição da lista *frag(owner, member)*

4.3 ETAPA DE FRAGMENTAÇÃO VERTICAL

A fragmentação vertical contribui para aumentar o desempenho de sistemas sobre bases distribuídas na medida em que reduz o acesso das aplicações às informações irrelevantes, agrupando no mesmo fragmento todos os atributos e métodos da classe que são acessados simultaneamente pelas operações mais freqüentes. Desta forma, a estrutura lógica da classe é particionada em fragmentos distintos.

A segunda etapa da metodologia realiza a fragmentação vertical em uma classe indicada pela etapa de análise. O algoritmo utilizado para a fragmentação vertical, ilustrado na figura 4.4, é uma extensão de (NAVATHE *et al.*, 1984, NAVATHE e RA, 1989).

O algoritmo recebe como entrada informacões sobre as operações que acessam a classe a fim de identificar os atributos e métodos da classe que são acessados simultaneamente pelas operações mais freqüentes. A idéia geral do algoritmo é definir os fragmentos verticais da classe através da construção de um grafo de afinidades entre os atributos e métodos, onde cada ciclo resultante no grafo representa um fragmento.

As principais vantagens deste algoritmo, mais detalhado em (BAIÃO, 2002), são: (i) a geração de todos os fragmentos verticais em uma única iteração ($O(n^2)$, onde n é o número de elementos da classe), tornando-o mais eficiente que outros algoritmos da literatura (MCCORMICK *et al.*, 1972); (ii) não requer procedimento de partição binária iterativa como os algoritmos de EZEIFE e BARKER (1998), NAVATHE *et al.* (1984) e CORNELL e YU (1987), o que reduz a sua complexidade computacional; (iii) é adaptável para o modelo de dados orientado a objetos; e (iv) é adaptável para a geração de fragmentos horizontais da classe, conforme mostrado na seção 4.4 a seguir. Maiores detalhes sobre o algoritmo podem ser consultados em (BAIÃO, 2002).

```
function VerticalFragmentation( Cv: set of classes to be vertically fragmented, Oproj: the set of projection operations)
returns Fv : set of vertical class fragments
begin
  for each Ck that is in Cv do                                                    (1)
    M = BuildElementAffinityMatrix(Ck, Oproj)                                     (2)
    fragmentsOfCk = BuildAndPartitionElementAffinityGraph(Ck, M)                  (3)
    Fv += fragmentsOfCk                                                           (4)
  end for
  return Fv
end

function BuildElementAffinityMatrix ( Ck: class to be vertically fragmented, Oproj: the set of projection operations and
                                      their execution frequencies)
returns M : element affinity matrix of Ck
begin
  for each Oi that is in Oproj do                                                 (5)
    for each element ei of Ck that is accessed by Oi do                           (6)
      for each element ej of Ck that is accessed by Oi do                         (7)
        freqOi = execution frequency of Oi
        if M(ei, ej) is not null then M(ei, ej) += freqOi                         (8)
                                 else create M(ei, ej), set M(ei, ej) = freqOi    (9)
        end if
      end for
    end for
  end for
  return M
end

function BuildAndPartitionElementAffinityGraph ( Ck: class to be vertically fragmented, M : element affinity matrix of Ck)
returns fragmentsOfCk : set of vertical fragments of Ck
begin
  N = empty set of nodes; A = empty set of links; G = (N, A)
  N += any element of Ck
  while there is an element of Ck that is not in N do                             (10)
    M(ei,ej) = highest element from M such that ei is a graph extremities and ej is the new node to be inserted (11)
    a := link between ei and ej
    N += ei; N += ej                                                              (12)
    if M(ei,ej) forms a cycle in the graph G then                                 (13)
      let cp be this cycle                                                        (14)
      if cp can be an affinity cycle then                                         (15)
        mark cp as a fragment candidate                                           (16)
      end if
    else                                                                          (17)
      if there is a fragment candidate then                                       (18)
        let cf be this candidate                                                  (19)
        if cf cannot be extended then                                             (20)
          mark cf as a fragment                                                   (21)
          A += a
          fragmentsOfCk += cf                                                     (22)
        end if
      end if
    end if
  end while
  return fragmentsOfCk
end
```

Figura 4.4: Etapa de Fragmentação Vertical

4.4 ETAPA DE FRAGMENTAÇÃO HORIZONTAL

A terceira etapa da metodologia realiza a fragmentação horizontal nas classes indicadas pela etapa de análise. Esta etapa recebe uma lista de pares de classes da forma (*owner*, *member*), onde fragmentação horizontal primária é aplicada sobre a classe *owner* e fragmentação horizontal derivada é aplicada sobre a classe *member*.

4.4.1 FRAGMENTAÇÃO HORIZONTAL PRIMÁRIA

O algoritmo utilizado para a fragmentação horizontal primária, ilustrado na figura 4.5, é similar ao algoritmo utilizado na etapa de fragmentação vertical apresentado na seção 4.3.

O algoritmo também recebe como entrada informações sobre as operações que acessam a classe, a fim de identificar grupos de objetos com características comuns que tenham alta probabilidade de serem acessados simultaneamente pelas operações mais freqüentes. Os fragmentos horizontais primários da classe são definidos através da construção de um grafo de afinidades entre os predicados simples sobre a classe, onde cada ciclo resultante no grafo representa um fragmento.

As principais vantagens de compartilharmos a mesma abordagem entre os dois algoritmos de fragmentação é a redução da dificuldade de implementação e a utilização de um paradigma comum tanto para fragmentação horizontal primária quanto para fragmentação vertical.

Maiores detalhes sobre o algoritmo podem ser consultados em (BAIÃO, 2002).

4.4.2 FRAGMENTAÇÃO HORIZONTAL DERIVADA

A definição de fragmentos horizontais derivados sobre uma classe indicada pela etapa de análise é bastante direta. Nesta fase, a lista *frag(owner, member)* definida anteriormente serve como um guia com o objetivo de agrupar num mesmo fragmento objetos de classes distintas que estejam relacionados entre si e que sejam referenciados pelos caminhos de navegação das operações mais freqüentes. Cada classe "membro" será fragmentada de forma derivada em relação à sua classe dono na lista *frag(owner, member)*. Conforme descrito em (BAIÃO, 2002), os fragmentos horizontais derivados que são definidos podem ser implementados com um operador de semi-junção (*semi-join*).

```
function PrimaryHorizontalFragmentation( Ch: set of pairs of classes to be horizontally fragmented,
                                         Osel: set of selection operations,
                                         Fv: set of vertical class fragments already defined)
returns Fp : set of primary horizontal class fragments, Fh : set of hybrid class fragments
begin
  Cowner = empty set;                                                                                    (1)
  for each owner class Ci in Ch that is not a member in any other pair (own, mem) in Ch do Cowner += Ci  end for   (2)
  for each Ck in Cowner do          //PHF is applied on "owner-only" classes                             (3)
     M = BuildPredicateAffinityMatrix(Ck, Osel)                                                          (4)
     (primaryHorizontalFragmentsOfCk, hybridFragmentsOfCk) = BuildAndPartitionPredicateAffinityGraph(Ck, M, Fv)   (5)
     Fp += primaryHorizontalFragmentsOfCk; Fh += hybridFragmentsOfCk                                     (6)
  end for
  return Fp, Fh
end

function closelyRelatedPredicate ( oi, oj: selection operations, currentOsel: the set of selection operations)
returns c: boolean
begin
  c := false                                                                                             (7)
  pi := attribute accessed by oi; pj := attribute accessed by oj;                                        (8)
  if pi = pj then              // oi and oj access the same attribute                                    (9)
    if oi and oj are jointly used with some common selection operation ok in other queries then          (10)
      pk := attribute accessed by ok;                                                                    (11)
      if pk != pi then         // ok accesses an attribute other than the one accessed by oi and oj     (12)
        c := true                                                                                        (13)
      end if
    end if
  end if
  return c
end

function BuildPredicateAffinityMatrix ( Ck: class to be horizontally fragmented, Osel: the set of selection operations and their execution
                                        frequencies)
returns M : predicate affinity matrix of Ck
begin
  for each pair of selection operations oi, oj in Osel extracted from the same query Q such that C(oi)=C(oj)=Ck do   (14)
    currentOsel += oi; currentOsel += oj                                                                 (15)
    freqQ = execution frequency of Q
  end for
  for each operation oi that is in currentOsel do                                                        (16)
    for each operation oj that is in currentOsel do                                                      (17)
      if M(oi, oj) is not null then M(oi, oj) += freqQ                                                   (18)
                               else create M(oi, oj), set M(oi, oj) = freqQ                              (19)
      end if
      if oi => oj then M(oi, oj).logicLink = true  end if    // logic implication                       (20)
      if closelyRelatedPredicate(oi, oj, currentOsel) then M(oi, oj).closeLink = true    end if         (21)
    end for
  end for
  return M
end

function BuildAndPartitionPredicateAffinityGraph ( Ck: class to be horizontally fragmented,
                                                   M : predicate affinity matrix of Ck,
                                                   Fv: set of vertical class fragments already defined)
returns primHorizontalFragmentsOfCk : set of primary horizontal fragments of Ck,
        hybridFragmentsOfCk : set of hybrid fragments of Ck
begin
  N = empty set of nodes; A = empty set of links; G = (N, A)                                             (22)
  N += any operation of currentOsel                                                                      (23)
  while there is an operation of currentOsel that is not in N do                                        (24)
    M(oi,oj) = highest element from matrix M such that oi or oj are one of the current graph extremities (25)
    N += oi; N += oj                                                                                     (26)
    if M(oi, oj) forms a cycle in the graph G then                                                       (27)
      let cp be this cycle                                                                               (28)
      if cp can be an affinity cycle then mark cp as a fragment candidate                                (29)
    else                                                                                                 (30)
      if there is a fragment candidate then                                                              (31)
        let cf be this candidate                                                                         (32)
        if cf cannot be extended then                                                                    (33)
          mark cf as a group of operations                                                               (34)
        end if
      end if
    end if
  end while
  // Cycles in the predicate affinity graph result in a set of groups of operations, which must be processed to create the fragments
  for each group of operations g = (O1, O2, .., Oq) do   //reduce the number of operations in the group using the logical predicates (35)
    for each pair of operations Oi, Oj that is in g do                                                   (36)
      if (Oi != Oj) and (M(Oi, Oj).logicLink = true) then g -= Oi   end if                               (37)
    end for
    mark g as an operation term; TO += g                                                                 (38)
  end for
  TO = {t1, t2, ..., tt}          // TO is the set of operation terms previously defined               (39)
  Tab = empty table               // Tab is a table of operation terms on Ck                           (40)
  while there is a Ck element in TO which is not in Tab do                                               (41)
    let e be the less frequent element of Ck in TO such that e is not in Tab yet                         (42)
    create a new column c in Tab                                                                         (43)
    fill each element of c with operations over e such that the combination of elements in the same row defines a term in TO (44)
  end while
  for each row r in Tab do                    //Apply the operation terms to define horizontal or mixed fragments on Ck   (45)
    if there are vertical fragments of Ck in Fv then                                                     (46)
       // Hybrid Fragmentation!
       Fh += combination of all elements in row r applied to the vertical fragments of Ck containing these elements   (47)
    else                                                                                                 (48)
       // Primary Horizontal Fragmentation!
       Fp += the combination of all elements in row r applied to class Ck                                (49)
    end if
  end for
  primHorizontalFragmentsOfCk := Fp ; hybridFragmentsOfCk := Fh
  return Fh, Fm
end
```

Figura 4.5: Etapa de Fragmentação Horizontal Primária

Capítulo 5

UMA ABORDAGEM BASEADA EM CONHECIMENTO PARA REFINAMENTO DO ALGORITMO DE ANÁLISE DO PDBDOO

Este capítulo apresenta o módulo de revisão de teorias da infra-estrutura proposta, representado por uma estratégia baseada em conhecimento para refinamento de algoritmos utilizando uma técnica de inteligência artificial. A estratégia apresentada é aplicada sobre um dos algoritmos do módulo heurístico, e os resultados complementam-se no sentido de obter um algoritmo ainda mais eficiente para o projeto de distribuição.

5.1 MOTIVAÇÃO

Os resultados obtidos com a aplicação da metodologia apresentada no capítulo 4 (detalhados em BAIÃO, 2002) evidenciaram a eficiência dos algoritmos propostos. No entanto, seria extremamente interessante que tais algoritmos, uma vez propostos, pudessem ser progressivamente melhorados no sentido de refletir heurísticas adicionais definidas por novos experimentos práticos da literatura. Esta constante melhora dos algoritmos requer, no entanto, uma análise detalhada de cada um destes novos experimentos, assim como a realização manual de alterações nos algoritmos adicionais. Neste sentido, as versões preliminares do algoritmo de análise originalmente proposto em (BAIÃO, 1997, BAIÃO e MATTOSO, 1998) foram manualmente modificadas considerando os resultados obtidos por (LIMA e MATTOSO, 1996, MEYER e MATTOSO, 1998). Entretanto, a formalização de novas heurísticas e sua incorporação ao algoritmo já existente, de tal forma a manter a consistência das heurísticas definidas previamente, mostrou-se um problema complexo, e de dificuldade crescente.

Desta forma, este capítulo propõe uma abordagem baseada em conhecimento para refinamento automático dos algoritmos de PDBDOO através de uma técnica de inteligência artificial denominada revisão de teorias. No contexto deste capítulo, esta estratégia será aplicada sobre o algoritmo de análise apresentado no capítulo 4. O objetivo da estratégia de refinamento do algoritmo de análise é a incorporação automática das mudanças que se façam necessárias ao algoritmo de forma a obter novos esquemas de fragmentação com melhor desempenho de acordo com resultados experimentais, consequentemente refletindo as heurísticas implícitas em tais resultados.

Existem na literatura alguns trabalhos propondo a aplicação de técnicas de aprendizado de máquina no contexto de bancos de dados. BLOCKEEL e DE RAEDT (1996, 1998) apresentam uma abordagem para o projeto indutivo de bancos de dados dedutivos, baseando-se nas instâncias existentes no banco de dados para a definição de predicados. GETOOR *et al.* (2001) utiliza modelos probabilísticos originados da área de redes Bayesianas a fim de estimar a seletividade de consultas em um processador de consultas. No entanto, não existem na literatura abordagens que consideram o projeto de distribuição como uma aplicação para o processo de revisão de teorias. Neste contexto, a estratégia baseada em conhecimento apresentada nesta tese é inédita.

A descrição completa da modelagem do problema do PDBDOO como um domínio para a aplicação de um sistema de revisão de teorias está presente em (BAIÃO, 2002), onde também são apresentados resultados experimentais de revisão do algoritmo de análise, obtendo um algoritmo de análise revisado com desempenho superior ao original.

5.2 A TÉCNICA DE REVISÃO DE TEORIAS

A abordagem baseada em conhecimento proposta para o refinamento do PDBDOO implementa uma técnica de inteligência artificial denominada revisão de teorias (WROBEL, 1996). O processo de revisão de teorias é responsável por alterar automaticamente o algoritmo original (denominado teoria inicial, ou conhecimento preliminar) de forma a adequá-lo para a geração do melhor esquema de fragmentação encontrado pela busca. O resultado do procedimento de refinamento é o algoritmo alterado (denominado teoria revisada) capaz de obter a solução ótima que lhe foi apresentada como entrada.

A técnica de revisão de teorias tem o objetivo de encontrar uma modificação mínima em uma teoria inicial de forma a classificar corretamente um conjunto de exemplos de treinamento. A descrição formal do processo é apresentada na figura 5.1:

Dados:	um conceito-alvo C
	um conjunto P com exemplos positivos de C
	um conjunto N com exemplos negativos de C
	uma linguagem de hipóteses L
	uma teoria inicial T expressa em L descrevendo C
Encontre:	uma teoria revisada TR expressa em L que corresponda à modificação mínima de T tal que TR é correta nos exemplos de P e de N

Figura 5.1: Especificação formal do processo de revisão de teorias

Uma teoria é um conjunto de cláusulas de programa definidas, sem símbolos funcionais. Uma cláusula de programa definida é uma cláusula da forma:

$$\alpha \leftarrow \beta 1 \ldots \beta n$$

onde α, $\beta 1 \ldots \beta n$ são fórmulas atômicas (CASANOVA e GIORNO, 1987, LLOYD, 1987).

Um conceito-alvo é um predicado em uma teoria para o qual existam exemplos no conjunto de treinamento.

Um exemplo é uma instanciação (não necessariamente básica) de um conceito-alvo. Por exemplo, um exemplo do conceito "cardinality" é:

cardinality(Connection, large)

Cada exemplo i possui um conjunto de fatos associados F_i, o qual reúne todos os exemplos de um conceito em um conjunto de treinamento. Um exemplo positivo deve ser derivável da teoria e de seus fatos associados, enquanto que exemplos negativos não.

No domínio do PDBDOO, o conjunto de fatos representa uma definição particular de um esquema de banco de dados e de suas consultas:

class(atomicPart)
cardinality(atomicPart, large)
operation(query1, 100)
accessedClasses(query1, [atomicPart])

A corretude de uma teoria é definida a seguir: dados um conjunto P de exemplos positivos e um conjunto N de exemplos negativos, uma teoria T é correta com relação ao conjunto de exemplos sse:

$$\forall p \in P: T \cup F_p \models p$$
$$\forall p \in N: T \cup F_p \not\models p$$

O processo de revisão de teorias aplica um conjunto de modificações na teoria inicial com o objetivo de obter uma teoria revisada correta. As modificações realizadas em uma teoria resultam da aplicação de operadores de revisão, os quais fazem pequenas modificações sintáticas na teoria. Uma teoria revisada correta obtida através de uma modificação mínima da teoria inicial é alcançada minimizando o número de operadores aplicados. Por requerer um conjunto mínimo de modificações, o processo assume implicitamente que a teoria inicial encontra-se aproximadamente correta e, portanto, a teoria revisada deve ser semanticamente e sintaticamente tão parecida quanto possível com a teoria inicial.

Trabalhos anteriores (BRUNK e PAZZANI, 1995, BRUNK, 1996) apresentaram uma análise e comparações detalhadas entre os diversos sistemas de revisão de teoria existentes na literatura. Esta análise incluiu sistemas como o FORTE (RICHARDS e MOONEY, 1995), A_3 (WOGULIS, 1994) e PTR+ (KOPPEL *et al.*, 1994). As comparações apresentadas resultaram em uma classificação e uma metodologia de avaliação de sistemas de revisão de teorias quanto à capacidade de identificação e localização de erros na teoria inicial, independentemente da capacidade de consertá-los.

De acordo com BRUNK (1996), a análise e comparação do sistema FORTE frente aos outros sistemas demonstraram uma maior abrangência no seu espaço de busca de teorias revisadas, em diferentes domínios, desta forma obtendo teorias revisadas potencialmente melhores, ou seja, com maior acurácia. Em decorrência de ter um espaço de busca maior, o sistema FORTE trata um número maior de pontos de revisão, pois gera e avalia mais candidatos durante a busca.

No contexto do PDBDOO, todos os resultados mostrados em BRUNK (1996) nos levaram a escolher o sistema FORTE para a realização do procedimento de refinamento do algoritmo de análise. O procedimento de refinamento receberá como entrada uma implementação baseada em regras de primeira ordem (i.e., uma implementação em Prolog) do algoritmo de análise, um conjunto de exemplos e fatos associados. Mais detalhes sobre o sistema FORTE e a sua utilização para revisar o algoritmo de análise do PDBDOO podem ser encontrados em (BAIÃO *et al.*, 2001).

5.3 REVISÃO DE TEORIAS NO PROJETO DE BANCOS DE DADOS DISTRIBUÍDOS (TREND^3)

A abordagem baseada em conhecimento para o refinamento do algoritmo de análise do PDBDOO proposta nesta seção é denominada "Revisão de Teorias no Projeto de Bancos de Dados Distribuídos" (Theory REvisioN on the Design of Distributed Databases - TREND3).

A abordagem é composta das seguintes fases: definição e construção da teoria inicial do domínio, escolha do sistema de revisão de teorias, escolha do conjunto de exemplos e realização do processo de revisão de teorias.

5.3.1 DEFINIÇÃO E CONSTRUÇÃO DA TEORIA INICIAL DO DOMÍNIO

O objetivo desta fase é extrair e explicitar todo o conhecimento existente sobre o problema, e representá-lo de forma adequada. O conhecimento a ser extraído pode estar tanto sob o domínio de especialistas no assunto, quanto implícito em resultados experimentais de trabalhos apresentados anteriormente na literatura. No escopo do presente trabalho, este conhecimento foi representado sob a forma de heurísticas para direcionar a fragmentação de classes, apresentadas no capítulo 3.

Todo este conhecimento extraído deve então ser estruturado e representado de forma adequada à realização do processo de revisão de teorias. O algoritmo de análise apresentado no capítulo 4, que incorpora todo o conjunto de heurísticas para a fragmentação de classes, foi então implementado como um conjunto de cláusulas de Horn de primeira ordem (i.e., um programa Prolog). A estrutura geral do programa é ilustrada na figura 5.2. (BAIÃO, 2002) mostra a implementação completa em Prolog do algoritmo de análise, e a representação da aplicação definida pelo benchmark 007 (CAREY *et al.*, 1993) que pode ser utilizada como parâmetro de entrada para este algoritmo.

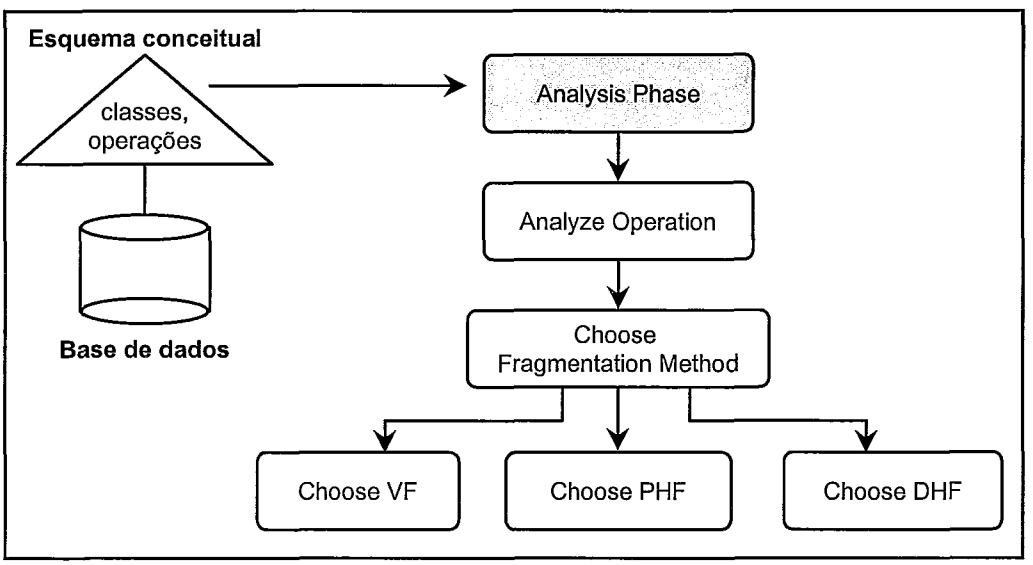


Figura 5.2: A estrutura geral do conjunto de regras para fragmentação de classes

5.3.2 ESCOLHA DO SISTEMA DE REVISÃO DE TEORIAS

O objetivo desta fase é escolher o ambiente que será utilizado para realizar o processo de revisão de teorias do módulo TREND3, incluindo a escolha do sistema que será aplicado, a axiomatização do problema, além de adaptações e extensões que possam ser necessárias.

Como descrito na seção 5.2, o sistema FORTE foi o escolhido para o módulo de revisão de teorias no escopo deste trabalho. No entanto, para que o sistema FORTE pudesse ser utilizado na abordagem TREND3, foi necessário realizar a axiomatização do problema, a fim de modelar e representar toda a informação relevante no domínio do projeto de distribuição (teoria inicial de domínio e conjunto de exemplos) de forma adequada como requerido pelo sistema FORTE.

A teoria inicial de domínio foi então dividida em dois arquivos: a "teoria fundamental de domínio" (fundamental domain theory, arquivo .FDT) e a "teoria inicial" (initial theory, arquivo .THY). O conteúdo completo destes arquivos encontra-se em (BAIÃO, 2002).

TEORIA FUNDAMENTAL DE DOMÍNIO

A teoria fundamental de domínio é composta dos predicados da teoria de domínio inicial que não devem ser revisados pelo sistema FORTE, ou seja, a parte d algoritmo de análise que é assumida previamente como correta. Por exemplo, predicados que definem quando uma operação navega de uma classe X para outra classe Y em uma expressão de caminho:

```
navigatesFromTo( O, X, Y ) :-
    operationAccess( O, ClassPath ),
    member( X, ClassPath ),
    member( Y, ClassPath ),
    navigates( O, X, Y ).

navigatesFromTo( O, X, Y ) :-
    operationAccess( O, ClassPath ),
    member( X, ClassPath ),
    member( Y, ClassPath ),
    navigates( O, Y, X ).
```

TEORIA INICIAL PARA SER REVISADA

Por outro lado, a teoria inicial é representada pelos predicados da teoria de domínio inicial que devem ser revisados pelo FORTE. Deve ser um programa em Prolog puro, e todas as relações e atributos devem ser precedidos por uma referência explícita ao módulo da teoria fundamental de domínio. Por exemplo, o predicado de escolha da técnica de fragmentação vertical para uma classe X é definido como:

```
chooseVerticalFragmentationMethod( Oi, X ) :-
    fdt:classification( Oi, projection ),
    fdt:operationAccess( Oi, [X|_] ),
    fdt:cardinality( X, large ),
    fdt:isNotDerivedFragmented( X ).

chooseVerticalFragmentationMethod( Oi, X ) :-
    fdt:classification( Oi, projection ),
    fdt:operationAccess( Oi, [X|_] ),
    fdt:cardinality( X, medium ),
    fdt:isNotDerivedFragmented( X ).
```

5.3.3 ESCOLHA DO CONJUNTO DE EXEMPLOS

Outra informação essencial ao sistema FORTE durante o processo de revisão de teorias é o conjunto de exemplos. A representação de um exemplo no FORTE é um termo Prolog com 4 argumentos:

example(PositiveInstances, NegativeInstances, Objects, Facts)

onde PositiveInstance (NegativeInstance) é uma lista de fatos positivos (negativos) do conceito a ser aprendido, objects são a representação do contexto da aplicação (no problema de fragmentação de classes, objects são representados pelas classes e operações da aplicação corrente), e facts são fatos previamente conhecidos, representados como fatos dos predicados da teoria fundamental de domínio.

Para a abordagem TREND3, as instâncias positivas representam boas escolhas da técnica de fragmentação a ser aplicada sobre uma das classe do esquema, que em conjunto levam à definição completa do esquema de fragmentação que tenha bom desempenho, e que por isso deve ser o resultado do algoritmo de análise revisado. Já as instâncias negativas correspondem à escolhas que são descartadas pelo algoritmo de análise, pois levam a esquemas de fragmentação com desempenho ruim.

Um exemplo possível sobre o benchmark 007 de quando a técnica de fragmentação vertical é escolhida para a classe AtomicPart durante a análise de uma operação de projeção, de acordo com os predicados definidos anteriormente nos arquivos FDT e THY, é o seguinte:

```
example( /* 007 */
      [ chooseVerticalFragmentationMethod( o1, atomicPart) ],
      [ ],
      [ class([ [designObject,  none,    none],
                [baseAssembly,  small,   none],
                [compositePart, small,   none],
                [atomicPart,    medium,  none],
                [connection,    large,   none]
              ]),
        relationship([ [componentsShared,  'N:N'],
                       [componentsPrivate, '1:N'],
                       [rootPart,          '1:1'],
                       [parts,             '1:N'],
                       [from,              '1:N'],
                       [to,                '1:N']
                     ]),
        operation([ [o1,  100, projection] ])
      ],
      facts([ relationshipAccess( componentsShared,  baseAssembly,  compositePart ),
              relationshipAccess( componentsPrivate, baseAssembly,  compositePart ),
              relationshipAccess( rootPart,          compositePart, atomicPart ),
              relationshipAccess( parts,             compositePart, atomicPart ),
              relationshipAccess( from,              atomicPart,    connection ),
```

```
            relationshipAccess( to,          atomicPart,    connection ),
            query( q1, 100, [o1] ),
            operationAccess( o1, [atomicPart] ),
        ])
    ).
```

No exemplo acima, a instância positiva é dada pelo termo chooseVerticalFragmentationMethod(o1, atomicPart). Não existem instâncias negativas definidas. Os objetos são dados pelo conjunto de classes, relacionamentos e operações da aplicação, enquanto os fatos são definidos pelas relações existentes entre os objetos (por exemplo, o acesso do relacionamento componentsShared às classes baseAssembly e compositePart). Os exemplos do TREND3 são passados ao FORTE em um arquivo de dados (.DAT). O arquivo DAT contém o conjunto de exemplos e também define parâmetros de execução para direcionar o processo de aprendizado do FORTE.

5.3.4 REALIZAÇÃO DO PROCESSO DE REVISÃO DE TEORIAS

A execução do processo de revisão de teorias é o próximo passo no contexto da abordagem TREND3. Este processo recebe como parâmetros principais a teoria inicial de domínio e um conjunto de exemplos. Esta seção mostra os resultados conseguidos com a abordagem TREND3, obtendo um algoritmo de análise que produz esquemas de fragmentação mais eficientes para a aplicação do benchmark 007.

O processo de revisão de teorias foi aplicado utilizando uma versão preliminar do algoritmo de análise (BAIÃO e MATTOSO, 1998) como teoria inicial de domínio. A execução deste algoritmo de análise preliminar retorna os conjuntos Ch e Cv da figura 5.3. Maiores detalhes desta versão preliminar do algoritmo (implementação em Prolog, detalhes da sua execução) estão em (BAIÃO, 2002).

```
Ch = {(atomicPart,connection),(baseAssembly,compositePart),(compositePart,atomicPart)}
Cv = { }
```

Figura 5.3: Resultado da execução da versão preliminar do algoritmo de análise

A execução do algoritmo de análise atual (como apresentado no capítulo 4) resultou nos conjuntos Ch e Cv da figura 5.4. Outros detalhes desta versão também podem ser encontrados em (BAIÃO, 2002).

```
Ch = {(AtomicPart,null),(BaseAssembly,CompositePart),(AtomicPart,Connection),(BaseAssembly, null) }
Cv = { AtomicPart }
```

Figura 5.4: Resultado da execução da versão atual do algoritmo de análise

Os custos dos esquemas de fragmentação resultantes destas duas versões do algoritmo de análise (após a aplicação dos algoritmos de fragmentação vertical e horizontal definidos na metodologia do capítulo 4) foram então comparados segundo a função de custo proposta em (RUBERG, 2001), como ilustrado na figura 5.5.

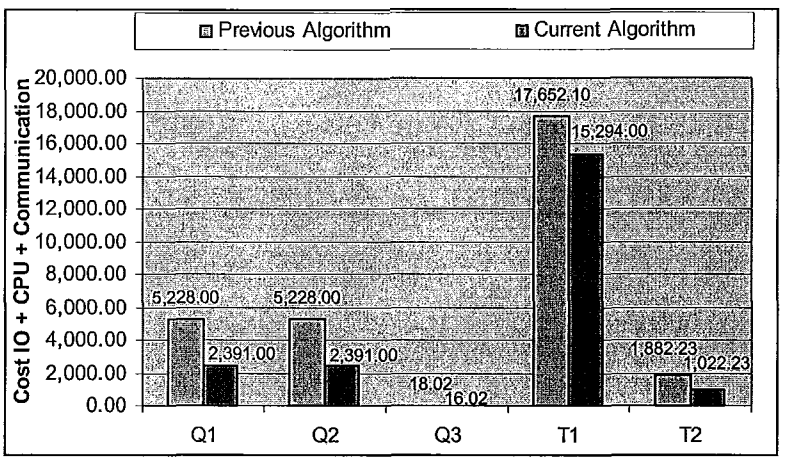


Figura 5.5: Custos das consultas do benchmark 007 segundo a aplicação dos esquemas de fragmentação obtidos com os algoritmos de análise preliminar e atual

A figura 5.5 mostra o custo de execução de cada consulta do benchmark 007. O custo total do benchmark 007 pode ser calculado como uma soma ponderada do custo de cada consulta de acordo com suas freqüências de execução:

Custo(007) = 100*custo(Q1) + 50*custo (Q2) + 30*custo (T1) + 30*custo (T2) + 10*custo (Q3)

Os custos totais de cada versão do algoritmo são:

CustoVersaoPreliminar(007) = 1,370,410.10

CustoVersaoAtual(007) = 848,297.10

Estes resultados identificam a oportunidade de melhoria da versão preliminar do algoritmo de análise. Embora isto tenha sido realizado manualmente através da análise de resultados experimentais de MEYER e MATTOSO (1998) e TAVARES *et al.* (2000), as modificações necessárias para alcançar tal melhoria não foram identificadas e implementadas trivialmente. Esta seção evidencia a importância da abordagem TREND3 proposta nesta tese em modificar o algoritmo de análise preliminar de forma automática, incorporando as modificações mais relevantes que foram necessárias para obtenção de um algoritmo de análise revisado equivalente ao algoritmo de análise atual. Detalhes sobre a execução do sistema FORTE são descritos em (BAIÃO, 2002).

O ALGORITMO DE ANÁLISE REVISADO

Finalmente, o algoritmo de análise revisado (teoria revisada) resultante da abordagem TREND3 é ilustrada na figura 5.6.

```
chooseDerivedHorizontalFragmentationMethod(A,B,C):-
    fdt:classification(A,navigation),
    fdt:navigatesFromTo(A,C,B),
    fdt:relationshipAccess(D,B,C),
    fdt:relationship(D),
    fdt:relationshipType(D,N:1),
    fdt:isNotVerticallyFragmented(B),
    fdt:isNotDerivedFragmented(B).
chooseDerivedHorizontalFragmentationMethod(A,B,C):-
    fdt:classification(A,navigation),
    fdt:navigatesFromTo(A,C,B),
    fdt:relationshipAccess(D,C,B),
    fdt:relationship(D),
    fdt:relationshipType(D,1:N),
    fdt:isNotVerticallyFragmented(B),
    fdt:isNotDerivedFragmented(B).
chooseDerivedHorizontalFragmentationMethod(A,B,C):-
    fdt:classification(A,navigation),
    fdt:navigatesFromTo(A,C,B),
    fdt:relationshipAccess(D,C,B),
    fdt:relationship(D),
    fdt:relationshipType(D,1:1),
    fdt:isNotVerticallyFragmented(B),
    fdt:isNotDerivedFragmented(B).

choosePrimaryHorizontalFragmentationMethod(A,B):-
    fdt:classification(A,selection),
    fdt:operationAccess(A,[B]).

chooseVerticalFragmentationMethod(A,B):-
    cardinality(B,medium),
    classification(A,projection).
chooseVerticalFragmentationMethod(A,B):-
    fdt:classification(A,projection),
    fdt:operationAccess(A,[B|C]),
    fdt:cardinality(B,large),
    fdt:isNotDerivedFragmented(B).
```

Figura 5.6: O algoritmo de análise resultante do processo de revisão de teorias

Pode ser verificado que a maior parte das modificações necessárias na versão preliminar a fim de obter o algoritmo atual como descrito no capítulo 4 foram executadas pelo sistema FORTE. Tais modificações incluem a regra adicionada:

```
chooseVerticalFragmentationMethod(A,B):-
    cardinality(B,medium),classification(A,projection).
```

e o antecedente fdt:cardinality(B,large) excluído da regra:

```
choosePrimaryHorizontalFragmentationMethod(A,B):-
    fdt:classification(A,selection),
    fdt:operationAccess(A,[B]),
    fdt:cardinality(B,large).
```

Maiores detalhes sobre as modificações aplicadas pela execução do sistema FORTE, incluindo detalhes sobre a sua limitação em trabalhar com literais negativos, estão presentes em (BAIÃO, 2002).

Capítulo 6

UMA ABORDAGEM ALTERNATIVA PARA O PDBDOO UTILIZANDO A TÉCNICA *BRANCH-AND-BOUND*

Este capítulo apresenta o módulo de branch-and-bound da infra-estrutura proposta, representado por uma abordagem baseada em técnicas de otimização para o projeto de distribuição de bases de dados.

6.1 MOTIVAÇÃO

No capítulo 4 foi proposta uma metodologia para o PDBDOO, cujo objetivo é obter um esquema de fragmentação de classes que minimize o custo do sistema distribuído (considerando para isso o custo de armazenamento dos dados, de processamento de transações e consultas sobre os dados armazenados e de comunicação entre os nós, conforme elucidado por ÖZSU e VALDURIEZ, 1999).

A metodologia proposta baseia-se em heurísticas obtidas através de estudos experimentais, ao invés de realizar uma busca exaustiva no espaço de todos os esquemas de fragmentação. Uma das desvantagens de abordagens heurísticas é que elas não garantem encontrar o ótimo global no espaço de hipóteses para o problema, enquanto que outras abordagens, como a busca exaustiva, garantidamente resultariam em uma solução ótima segundo um determinado critério. No entanto, como apontado por (ÖZSU e VALDURIEZ, 1999), a fragmentação de classes é um problema NP-difícil, o que impossibilita a aplicação de métodos de busca exaustiva, justificando desta forma a abordagem heurística seguida pela nossa metodologia.

Neste contexto, uma outra abordagem viável para o problema de fragmentação de classes seria a utilização de técnicas de otimização (como a técnica *branch-and-bound*), que de forma geral exploram todo o espaço de soluções potenciais, incluindo as que são encontradas em abordagens heurísticas. A técnica *branch-and-bound* realiza uma procura inteligente por soluções ótimas dentro de um espaço potencial de soluções para um dado problema (LAWLER e WOOD, 1966, MEIRA *et al.*, 1996, OPITZ e SHAVLIK, 1995), e tem sido bastante utilizada para a resolução de problemas NP-difíceis.

A existência de uma abordagem alternativa baseada na técnica *branch-and-bound* para tratar o problema do PDBDOO torna bastante interessante a sua avaliação, e a comparação de seus resultados com os resultados da abordagem heurística apresentada no capítulo 4. Esta comparação pode, inclusive, apontar um potencial de melhora e necessidade de refinamento dos algoritmos da metodologia proposta.

Neste sentido, este capítulo propõe uma estratégia alternativa para o PDBDOO utilizando técnicas de otimização (*branch-and-bound*). O objetivo da estratégia é obter

um esquema de fragmentação que maximize o custo do sistema distribuído como um todo.

6.2 VALIDAÇÃO DOS ALGORITMOS DO PDBDOO

O procedimento de validação aplicado ao PDBDOO explora, através da técnica *branch-and-bound* (LAWLER e WOOD, 1966, MEIRA *et al.*, 1996), o espaço potencial de esquemas de fragmentação viáveis para uma base de dados.

Genericamente, algoritmos *branch-and-bound* realizam uma busca por soluções ótimas dentro de um espaço potencial de soluções para um dado problema. Uma vez que esta técnica é geralmente aplicada a problemas com espaço de soluções muito grande, o objetivo é determinar a solução ótima através da ramificação do espaço de busca em subconjuntos menores (*branch*) enquanto minimiza o número de soluções a ser considerada explicitamente (*bound*).

Algoritmos de *branch-and-bound* caracterizam-se por quatro regras básicas que definem o seu comportamento:

- **Regra de Seleção**: utilizada para selecionar um estado do espaço de busca para avaliação;
- **Regra de Expansão**: o estado selecionado é avaliado de forma a verificar se ele leva a uma solução viável para o problema. Se a solução é viável então o estado é considerado para divisão, a fim de avaliar a possibilidade de melhora da solução;
- **Regra de Divisão**: define como os estados no espaço de busca são divididos em sub-problemas;
- **Regra de Limite**: define como descartar estados que não levam à solução ótima.

Resumidamente, a técnica avalia recursivamente cada subconjunto de estados, selecionando a cada passo o estado que deve ser avaliado (regra de seleção). Se o subconjunto leva a uma solução viável então ele é expandido para avaliar uma possível melhora na solução (regra de expansão). Os subconjuntos são expandidos através da divisão em sub-problemas, gerando um novo subconjunto de estados para avaliação (regra de divisão). Durante a avaliação de um estado, se ele leva a uma solução pior que o limite corrente então ele é descartado (regra de limite). Além disso, a avaliação de um estado pode estabelecer um novo limite para a solução ótima do problema. O processo

continua até que todos os estados tenham sido divididos ou descartados. Ao final, as soluções do problema estão associadas a cada estado não descartado, e a solução com a melhor avaliação é considerada a solução ótima.

No contexto do PDBDOO, o espaço de soluções contém os esquemas de fragmentação possíveis para uma aplicação (esquema do banco de dados e consultas), e cada subconjunto de estados é formado pela escolha de uma técnica de fragmentação para uma classe não fragmentada. Ainda, cada estado no espaço de busca representa uma classe do esquema da base de dados, e a divisão ocorre gerando-se todos os fragmentos possíveis da classe a partir da técnica de fragmentação escolhida.

A **regra de seleção** de um estado para avaliação é escolher sempre a próxima classe ainda não fragmentada, a qual é acessada pela operação mais freqüente que ainda não tenha sido analisada. Por exemplo, para o conjunto de operações e suas classes acessadas definidos a seguir:

```
operation( o1, 100 )
accessedClasses( o1, [AtomicPart] )
operation( o2, 50 )
accessedClasses( o2, [BaseAssembly, CompositePart, AtomicPart, Connection] )
```

a lista ordenada de classes selecionadas pela regra de seleção é:

```
[AtomicPart, BaseAssembly, CompositePart, Connection]
```

Para cada estado selecionado, são gerados subconjuntos de estados através da regra de expansão. A **regra de expansão** de uma classe é aplicar sobre ela as técnicas de fragmentação (vertical, horizontal primária, horizontal derivada, híbrida ou nenhuma) que sejam viáveis em relação às fragmentações já definidas em estados anteriores. Por exemplo, os seguintes subconjuntos de estados poderiam ser formados a partir da classe BaseAssembly:

```
chooseFragmentationMethod( BaseAssembly, NONE )
chooseFragmentationMethod( BaseAssembly, PHF )
chooseFragmentationMethod( BaseAssembly, VF )
chooseFragmentationMethod( BaseAssembly, HybridF )
chooseFragmentationMethod( BaseAssembly, DHF )
```

A **regra de divisão** dos subconjuntos de estados é gerar todos os fragmentos possíveis de uma classe para uma determinada técnica de fragmentação. Neste ponto, para o PDBDOO, esta geração é realizada da seguinte forma:

a. Para a fragmentação NONE, nenhum fragmento é gerado;

b. Para a fragmentação PHF, consideram-se todos os conjuntos de fragmentos que podem ser formados com a fragmentação horizontal primária da classe, obtidos através de cada combinação possível entre os predicados de seleção das operações sobre a classe;
c. Para a fragmentação VF, consideram-se todos os conjuntos de fragmentos que podem ser formados com a fragmentação vertical da classe, obtidos através da distribuição dos atributos e métodos da classe em partições distintas;
d. Para a fragmentação HybridF, consideram-se todos os conjuntos de fragmentos que podem ser formados com a fragmentação híbrida da classe, aplicando-se cada definição de fragmentação vertical em (c) sobre cada definição de fragmentação horizontal primária de (b); e
e. Para a fragmentação DHF, consideram-se todos os conjuntos de fragmentos que podem ser formados com a fragmentação horizontal derivada da classe, levando-se em conta cada uma das classes relacionadas como classe primária;

Por exemplo, considerando uma definição da classe BaseAssembly:

```
Class BaseAssembly
        x: integer;
        y: integer;
method getcompPrivate(): set(CompositePart);
```

então a divisão de chooseFragmentationMethod(BaseAssembly, VF) vai produzir os seguintes possíveis conjuntos de fragmentos verticais da classe:

```
verticalFragments( BaseAssembly, [[x],[y],[getcompPrivate]] )
verticalFragments( BaseAssembly, [[x,y],[getcompPrivate]] )
verticalFragments( BaseAssembly, [[x],[y,getcompPrivate]] )
verticalFragments( BaseAssembly, [[x,getcompPrivate],[y]] )
```

Os estados gerados (cada possível conjunto de fragmentos de classe) vão ser avaliados quanto ao seu desempenho e comparados ao limite corrente. A **regra de limite** utilizada é a seguinte: inicialmente, o valor do limite corrente corresponde ao desempenho do esquema de fragmentação resultante do algoritmo de análise, calculado através de uma chamada à função de avaliação escolhida. A cada passo de avaliação de um estado, se o conjunto de fragmentos de classe representado pelo estado tiver um desempenho pior que o limite corrente então ele é descartado. O limite corrente estabelecido inicialmente permanece constante durante toda a busca, já que estamos interessados em todos os esquemas de fragmentação que tenham um desempenho superior a este limite.

O desempenho de um esquema de fragmentação corresponde ao custo do sistema distribuído, e pode ser calculado da seguinte maneira: dado um conjunto de operações { o1, ..., on } a serem executadas sobre uma base de dados, onde cada operação oi tem uma freqüência de execução wi (1<=wi<=100), o custo do sistema é dado por

$$custo(SISTEMA) = SUM_{i=1 \text{ to } n} (wi * custo(oi))$$

onde custo(oi) é calculado utilizando a função de custo proposta em (RUBERG, 2001, RUBERG *et al.*, 2001). A função de custo utilizada fornece estimativas para o custo do processamento de consultas em bases de objetos distribuídos, levando em conta operações de entrada e saída (I/O), instruções de CPU e comunicação entre os nós.

Os conjuntos de estados gerados pela regra de expansão que não forem descartados pela regra de limite são divididos até que todos os fragmentos possíveis de todas as classes tenham sido gerados. A solução ótima do problema é dada pelo esquema de fragmentação de custo mínimo que não tenha sido descartado.

O procedimento de busca é ilustrado pela chamada BFS_Search (*"best fragmentation schema search"*) da figura 6.1.

```
procedure BFS_Search ( app: Application )
begin
    BFS_Set = {}                                                              (1)
    currentBFS = {}                                                           (2)
    allOperations = all operations extracted from app, ordered in a descendent way according   (3)
                    to their frequencies
    allClasses  = all classes extracted from app                              (4)
    CDG = class dependency graph for app                                      (5)
    analysisAlgorithmFragSchema := AnalysisPhase( allClasses, allOperations, CDG )   (6)
    analysisAlgorithmCost := getCostFunctionCost( analysisAlgorithmFragSchema )      (7)
    BranchBoundBFSSearch (allOperations, currentBFS, 0, analysisAlgorithmCost, BFS_Set )   (8)
    BFS = fragmentation schema in BFS_Set with minimum cost                   (9)
    return BFS
end.

procedure BranchBoundBFSSearch(   OpSet: set(Operation),
                                  curFS: set(Class,Fragmentation), // representation of a fragmentation schema
                                  FSCost: int
                                  AnalysisAlgorithmCost: int
                                  BFS_Set: set(set(Class,Fragmentation), cost) )
begin
    if OpSet is empty then      // base case of recursion                     (10)
        if (not curFS.discard) then BFS_Set += curFS      end if              (11)
        return
    else                                                                      (12)
        o1 = first element from OpSet                                         (13)
        remove o1 from OpSet                                                  (14)
        for each class c that is referenced by o1                             (15)
            possibleFragmentations(c) = GetAllPossibleClassFragmentations(c, o1)   (16)
            for each f in possibleFragmentations(c)                           (17)
                currentCost = frequency(oi) * cost(oi, f)                     (18)
                if FScost + currentCost < AAOFScost then                      (19)
                    curFS.discard = false                                     (20)
                    curFS += {c, f}                                           (21)
                    FScost += currentCost                                     (22)
                    BranchBoundBFSSearch( OpSet, curFS, FSCost, BFS_Set )     (23)
                else
                    OpSet = {}                                                (24)
                    curFS.discard = true  // Current fragmentation schema is discarded   (25)
                end if
```

```
            end for
        end for
    end if
    return
end.
```

Figura 6.1: O procedimento de busca através da técnica *branch-and-bound* para o PDBDOO

Inicialmente, nenhuma das classes está fragmentada, e as variáveis de trabalho do algoritmo são inicializadas (linhas 1 – 5 da figura 6.1). Particularmente, a variável analysisAlgorithmCost representa o limite corrente, que inicialmente recebe o custo do esquema de fragmentação analysisAlgorithmFragSchema, resultante do algoritmo de análise (linhas 6 – 7 da figura 6.1). Ocorre então a chamada inicial do procedimento de busca em si (BranchBoundBFSSearch, linha 8 da figura 6.1), cujo parâmetro de saída BFS_Set vai conter o conjunto de esquemas de fragmentação não descartados pela busca. Ao final, a função retorna a solução ótima do problema (variável BFS), que é o esquema de fragmentação BFS_Set que tem o custo mínimo.

O procedimento de busca *BranchBoundBFSSearch* é um procedimento recursivo, que a cada iteração avalia um conjunto de estados (possíveis fragmentos de classes) que levam a uma solução do problema (esquema de fragmentação), como descrito anteriormente. O caso base da recursão é quando o conjunto de operações recebido como parâmetro está vazio. Quando isto acontece, se o esquema de fragmentação corrente que está sendo avaliado (curFS) não está marcado para ser descartado, ele é então inserido no conjunto de melhores esquemas de fragmentação (linhas 10 – 11 da figura 6.1). Por outro lado, se o conjunto de operações não estiver vazio então passa-se ao próximo passo da recursão de escolher a próxima classe para avaliação, segundo a regra de seleção (linhas 13 – 15 da figura 6.1). Expande-se e divide-se cada estado pela função GetAllPossibleClassFragmentations (linha 16 da figura 6.1), que retorna todos os conjuntos de fragmentos possíveis de cada técnica de fragmentação aplicada à classe, segundo as regras de expansão e divisão. O custo de cada estado gerado pela regra de divisão é calculado (cost(oi, f)) e adicionado ao custo completo do esquema de fragmentação corrente (currentCost), que leva em conta todas as operações já analisadas e suas freqüências (linha 18 da figura 6.1). A regra de limite é então aplicada (linha 19 da figura 6.1) e o conjunto de fragmentos da classe é descartado caso leve a um esquema de fragmentação de custo superior a analysisAlgorithmCost (linhas 24 e 25 da figura 6.1). Se o conjunto de fragmentos da classe não for descartado, então ele passa a compor o

esquema de fragmentação corrente e chama-se o procedimento *BranchBoundBFSSearch* recursivamente, até que todos os estados não descartados tenham sido expandidos e divididos.

A vantagem do procedimento de busca *BranchBoundBFSSearch* sobre os algoritmos heurísticos para a resolução do problema do PDBDOO é a existência de um espaço de hipóteses muito grande que é percorrido na procura de uma solução ótima, uma vez que os algoritmos apresentados no capítulo 4 representam algoritmos gulosos que varrem um espaço de soluções limitado pelo conjunto de heurísticas que implementam. No entanto, podemos argumentar que as heurísticas implementadas nestes algoritmos (principalmente no algoritmo de análise) foram obtidas através de estudos experimentais (LIMA e MATTOSO, 1996, MEYER e MATTOSO, 1998, TAVARES *et al.*, 2000), e não apenas através de estimativas fornecidas pela função de custo que é utilizada na fase de avaliação das soluções percorridas. Além disso, a técnica *branch-and-bound* requer grande esforço computacional e, com isso, o procedimento de busca pode se tornar muito custoso. O alto custo do procedimento de busca é decorrente do grande número de parâmetros de entrada para a busca, já que se dispõe a varrer o maior número possível de esquemas de fragmentação viáveis para o PDBDOO. Por outro lado, o algoritmo de análise possui um custo muito menor por implementar um método heurístico de busca pela melhor técnica de fragmentação para cada classe, e os algoritmos para fragmentação vertical e horizontal são eficientes conforme mostrado em (NAVATHE *et al.*, 1984). O alto custo do procedimento de busca pode tornar a sua aplicação inviável em situações como no projeto de distribuição dinâmico de bases de dados (DURAN, 1999), onde o esquema de fragmentação implementado em uma base de dados instanciada é constantemente avaliado e, caso o seu desempenho seja inferior a um limite mínimo, um novo esquema de fragmentação com desempenho melhor que o anterior deve ser rapidamente encontrado e implementado de forma transparente às aplicações que executam sobre a base de dados distribuída.

A avaliação de cada estado durante o procedimento *branch-and-bound* tem ainda o potencial de gerar informações sobre as soluções que foram exploradas durante a busca. Tais informações geradas podem ser utilizadas pelo procedimento de refinamento descrito anteriormente. Especificamente no contexto do PDBDOO, os estados descartados poderiam ser fornecidos ao processo de revisão de teorias como exemplos de esquemas de fragmentação com desempenho ruim, enquanto o esquema de

fragmentação ótimo encontrado pela busca representaria um bom esquema de fragmentação.

Capítulo 7

CONCLUSÕES

Este capítulo encerra a apresentação desta tese, apresenta as conclusões e contribuições mais relevantes que foram obtidas durante o seu desenvolvimento, além de apontar direções para trabalhos futuros.

Os dois objetivos principais do projeto de distribuição de bases de dados são o aumento do desempenho das aplicações e a maximização do acesso aos dados armazenados localmente. Estes dois objetivos são alcançados através da redução do acesso a dados irrelevantes pelas aplicações e da redução da transferência de dados entre os nós de um sistema distribuído. No entanto, para que se possa tirar máximo proveito da distribuição de dados e aumentar o desempenho do sistema tanto quanto possível, é essencial que a distribuição dos dados seja projetada apropriadamente, requerendo portanto a aplicação de um bom projeto de distribuição.

No modelo de dados relacional, o foco do projetista da distribuição era apenas em dividir os atributos de uma relação. Com a riqueza semântica do modelo de dados orientado a objetos, no entanto, existem diversos fatores adicionais que precisam ser considerados, o que aumenta e muito a dificuldade desta tarefa. Podemos citar, por exemplo, atributos multi-valorados, relacionamentos - de associação, de herança e inversos - além de métodos que podem realizar acessos a todas as classes do esquema. Por esses fatores, a adaptação direta de algoritmos relacionais para o modelo orientado a objetos não é viável, e deixa muitos aspectos em aberto como a classificação "dono-membro" de classes de um esquema para direcionar a fragmentação horizontal derivada. No modelo relacional, a escolha das classes "dono" e "membro" de um relacionamento é simples e direta, enquanto que no modelo orientado a objetos há relacionamentos N:M em que esta escolha não é clara, além de relacionamentos de herança que devem ser tratados de forma especial.

Para tratar as dificuldades existentes no modelo orientado a objetos, este trabalho propõe uma infra-estrutura para tratar o PDBDOO composta de três módulos: o módulo heurístico, o módulo de revisão de teorias e o módulo de *branch-and-bound*. A infra-estrutura proposta utiliza o modelo orientado a objetos no nível conceitual, a fim de captar de forma mais completa a semântica da aplicação.

O módulo heurístico contempla uma metodologia completa, descrições e algoritmos para auxiliar o projetista da distribuição durante a fragmentação dos dados. A metodologia também define a classificação "dono-membro" no contexto orientado a objetos e outras características, e a forma como elas podem ser utilizadas para direcionar a etapa de fragmentação horizontal primária e derivada. A principal contribuição do módulo heurístico é a etapa de análise, que escolhe automaticamente a técnica de fragmentação mais adequada a cada classe do esquema baseando-se em heurísticas

obtidas através de resultados experimentais. Com a utilização de outras estratégias e algoritmos existentes na literatura, o projetista da distribuição é levado a aplicar uma única técnica de fragmentação a todas as classes do esquema do banco de dados. Mesmo se o projetista decidir combinar a utilização de duas técnicas na mesma base de dados, não existe nenhum trabalho na literatura que o auxilie a decidir a melhor forma de realizar tal combinação.

Uma característica importante da metodologia proposta é a incorporação de fragmentação mista e híbrida, além da fragmentação vertical e horizontal - primária e derivada. O algoritmo de análise automaticamente detecta, em um mesmo esquema, classes que devem ser fragmentadas horizontalmente devido a um padrão de acesso navegacional, classes que devem ser fragmentadas verticalmente devido a um padrão de acesso associativo ou às características de utilização de seus atributos e métodos, ou mesmo classes em que nenhuma técnica de fragmentação é indicada. Além de levar em consideração um grande número de aspectos para direcionar esta escolha, a metodologia proposta também apresenta características inéditas, como: análise criteriosa da semântica do modelo de dados (incluindo cardinalidade dos relacionamentos) e de informações quantitativas (existência e tamanho estimado de coleções que implementam extensões de classes), e redução da complexidade e do número de fragmentos no esquema de fragmentação resultante (através das heurísticas implementadas).

Através da utilização de fragmentação mista e híbrida, o sistema distribuído resultante beneficia-se da redução dos dados irrelevantes acessados pelas aplicações, e do aumento do potencial de paralelismo durante a execução das consultas pelo SGBDOO. Como mostrado em nossos trabalhos anteriores (BAIÃO, 1997, BAIÃO *et al.*, 2000, 2001), a metodologia proposta obtém esquemas de fragmentação com melhor desempenho quando comparados a outros esquemas de fragmentação da literatura.

Nossos resultados também evidenciam que parâmetros como o padrão de acesso e freqüência de execução das aplicações, e a cardinalidade das classes e dos relacionamentos exercem grande influência no desempenho de operações muito custosas em aplicações sobre dados distribuídos. A etapa de análise se mostrou essencial e bastante eficiente no auxílio ao projetista da distribuição, já que a análise de todos estes parâmetros e a combinação das diversas heurísticas não é trivial. Este auxílio é ainda mais eficiente através do algoritmo de análise, que automaticamente decide a técnica de fragmentação mais adequada dentre todas as possíveis, o que não é abordado em

trabalhos relacionados. Nossos resultados mostraram que a metodologia contribuiu para o aumento do desempenho de aplicações que acessam fragmentos horizontais e/ou verticais da base de dados (fato bastante comum em aplicações do mundo real), e desta forma vai ao encontro da necessidade ressaltada por ÖZSU e VALDURIEZ (1999) de "uma metodologia de projeto de distribuição que englobe algoritmos de fragmentação horizontal e vertical e os utiliza como parte de uma estratégia mais geral".

No contexto da infra-estrutura para o projeto de distribuição apresentada neste trabalho, o módulo de revisão de teorias propõe uma estratégia de refinamento do algoritmo de análise do módulo heurístico, através de uma abordagem baseada em conhecimento. A estratégia proposta utiliza uma técnica de programação em lógica indutiva (revisão de teorias) que modifica automaticamente o algoritmo de análise de forma a obter a solução ótima apresentada, e incorporar resultados experimentais adicionais ao conjunto de heurísticas representado. Estudos experimentais mostraram a viabilidade do procedimento de revisão de teorias.

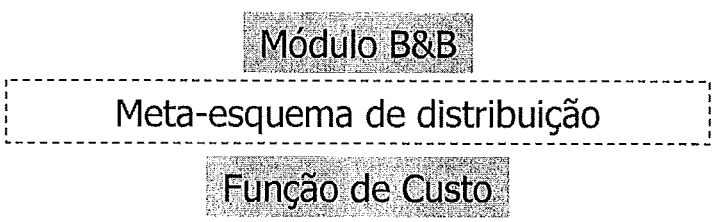
Finalmente, o módulo *branch-and-bound* propõe uma abordagem alternativa para o PDBDOO através de técnicas de otimização. O procedimento de *branch-and-bound* proposto realiza uma busca inteligente pelo esquema de fragmentação que maximize o desempenho da aplicação, dentre o espaço de esquemas de fragmentação potencialmente bons. Para lidar com o alto custo de execução deste algoritmo o espaço de busca é reduzido, descartando-se as ramificações do espaço de busca que levam a esquemas de fragmentação com um custo maior do que o custo do esquema de fragmentação obtido pelo módulo heurístico. Adicionalmente, o esquema de fragmentação resultante do módulo *branch-and-bound* (assim como os esquemas de fragmentação descartados durante a busca) podem ser representados como exemplos positivos (negativos) ao módulo de revisão de teorias, desta forma incorporando os resultados do módulo *branch-and-bound* ao módulo heurístico através das modificações realizadas no algoritmo de análise. Maiores detalhes sobre a proposta deste tese podem ser encontrados em (BAIÃO, 2002).

7.1 TRABALHOS FUTUROS

De acordo com os resultados e contribuições apresentados nesta tese, foram identificadas algumas oportunidades de trabalhos futuros em cada um dos módulos

apresentados, que incluem modificação dos algoritmos do módulo heurístico para avaliar ordens alternativas de aplicação das técnicas de fragmentação vertical e horizontal, avaliação e adaptação da infra-estrutura proposta para trabalhar diretamente com SGBDs que utilizam o modelo relacional-objeto, modificação do sistema FORTE para tratamento de literais negativos, geração automática de exemplos para o módulo de revisão de teorias, além de adaptações no algoritmo de *branch-and-bound* para aumentar ainda mais o espaço de busca percorrido:

- regra de seleção da classe a ser fragmentada que gere um maior número de alternativas possíveis;
- geração do conjunto de fragmentos da classe que se aproxime da geração exaustiva, quando possível;
- cálculo do parâmetro da função de custo que indica os fragmentos de uma classe que podem ser eliminados pelo otimizador de consultas, dentro do contexto de um "meta-esquema" de distribuição que representaria a interface entre o módulo *branch-and-bound* e a função de avaliação do custo de esquemas de fragmentação, como no esquema a seguir:



REFERÊNCIAS

BAIÃO, F., 1997, *Uma Estratégia para o Projeto de Distribuição de Bases de Dados Orientadas a Objetos*, Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Abril

BAIÃO, F., MATTOSO, M., 1997, "A Mixed Fragmentation Strategy for Distributed OO Databases". In: *Proceedings of the Second Workshop on CSCW in Design (CSCWID'97)*, International Academic Publishers, Bangkok, Tailândia, Novembro, pp. 42-48

BAIÃO, F., MATTOSO, M., 1998, "A Mixed Fragmentation Algorithm for Distributed Object Oriented Databases". In: *Proceedings of the International Conference on Computing and Information (ICCI'98)*, Winnipeg, Canadá, Junho, pp. 141-148. Also in *Special Issue of the Journal of Computing and Information* (JCI), vol. 3(1), ICCI 98, Março 2000, ISSN 1201-8511, pp. 141-148

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G., 1998, "Towards an Inductive Design of Distributed Object Oriented Databases". In: *Proceedings of the Third IFCIS Conference on Cooperative Information Systems (CoopIS'98)*, IEEE CS Press, Nova York, EUA, Agosto, pp. 88-197

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G., 2000, "Horizontal Fragmentation in Object DBMS: New Issues and Performance Evaluation". In: *Proceedings of the 19th IEEE International Performance, Computing and Communications Conference (IPCCC 2000)*, IEEE CS Press, Phoenix, Fevereiro, pp.108-114

BAIÃO, F., 2002, "A Methodology and Algorithms for the Design of Distributed Databases using Theory Revision", Relatório Técnico ES-565/02, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Brasil, Janeiro

BAIÃO, F., MATTOSO, M., ZAVERUCHA, G., 2001b, "A Distribution Design Methodology for Object DBMS", submetido em Agosto 2000; manuscrito revisado enviado em Novembro 2001 à *International Journal of Distributed and Parallel Databases*, Kluwer Academic Publishers

BELLATRECHE, L., KARLAPALEM, K., SIMONET, A., 2000, "Algorithms and Support for Horizontal Class Partitioning in Object-Oriented Databases", *International Journal of Distributed and Parallel Databases*, Kluwer Academic Publishers, vol. 8(2), pp. 155-179

BELLATRECHE, L., KARLAPALEM, K., BASAK, B., 1998, "Query-Driven Horizontal Class Partitioning in Object-Oriented Databases". In: *Proceedings of the 9th International Conference on Databases and Expert Systems (DEXA'98)*, Lecture Notes in Computer Science, vol. 1460, Vienna, Áustria, pp. 692-701

BELLATRECHE, L., SIMONET, A., SIMONET, M., 1996, "Vertical Fragmentation in Distributed Object Database Systems with Complex Attributes and Methods". In: *Proceedings of the "7th International Workshop on Database and Expert Systems Applications" (DEXA'96)*, IEEE Computer Society, Zurich, Suíça, pp. 15-21

BENZAKEN, V., DELOBEL, C., HARRUS, G., 1992, "Clustering Strategies in O_2: An Overview". In: BANCILHON, F., DELOBEL, C., KANELLAKIS, P. (eds.), *Building an Object Oriented Database System. The Story of O_2*, capítulo 17, pp. 385-410, Morgan Kaufman Publishers Inc., São Francisco, EUA

BERTINO, E., FOSCOLI, P., 1997, "On Modeling Cost Functions for Object-Oriented Databases", *IEEE Trans. Knowledge and Data Engineering*, vol. 9(3), pp. 500-508

BLOCKEEL, H, DE RAEDT, L., 1996, "Inductive Database Design", In: RAS, Z., MICHALEWICZ, M. (eds.), *Proceedings of the 10th International Symposium on Methodologies for Intelligent Systems* (ISMIS'96), LNAI 1079, pp. 376-385, Springer-Verlag

BLOCKEEL, H, DE RAEDT, L., 1998, "Isldd: an Interactive System for Inductive Database Design", *Applied Artificial Intelligence*, 12(5), pp. 385-420

BOOCH, G., RUMBAUGH, J., JACOBSON, I., 1999, *The Unified Modeling Language User Guide*, Addison Wesley Longman, Inc., USA

BRUNK, C., PAZZANI, M., 1995, "A Linguistically-Based Semantic Bias for Theory Revision". In: *Proceedings of the 12th International Conference of Machine Learning*, São Francisco, EUA, pp. 81-89

BRUNK, C., 1996, *An Investigation of Knowledge Intensive Approaches to Concept Learning and Theory Refinement*, PhD Thesis, University of California, Irvine, EUA

CAREY, M., DEWITT, D., NAUGHTON, J., 1993, "The OO7 Benchmark". In: *Proceedings of the 1993 ACM SIGMOD,* vol. 22(2), Washington DC, pp. 12-21

CASANOVA, M, GIORNO, F., 1987, *Programação em Lógica e a Linguagem PROLOG*, Ed. Edgard Blucher, Brasil

CATTEL, R. *et al.*, 2000, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann Publishers Inc., São Francisco, EUA

CERI, S., NAVATHE, S., 1983, "A comprehensive approach to fragmentation and allocation of data in distributed databases". In: *Proceedings of the IEEE COMPCON Conference*, pp. 426-431

CHEN, Y., SU, S., 1996, "Implementation and Evaluation of Parallel Query Processing Algorithms and Data Partitioning Heuristics in Object Oriented Databases", *International Journal of Distributed and Parallel Databases*, Kluwer Academic Publishers, vol. 4(2), pp. 107-142

CLUET, S., DELOBEL, C., 1992, "A General Framework for the Optimization of Object-Oriented Queries". In: *Proceedings of the 1992 ACM SIGMOD*, vol. 21(2), San Diego, California, pp. 383-391

CORNELL, D., YU, P., 1987, "A Vertical Partitioning Algorithm for Relational Databases". In: *Proceedings of the 3rd International Conference on Data Engineering (ICDE'87),* Los Angeles, EUA, Fevereiro, pp. 30-35

DURAN, M., 1999, *Alocação Dinâmica de Objetos em Sistemas de Bancos de Dados Paralelos*. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Março

EISENBERG, A., MELTON, J., 1999, "SQL 1999, formerly known as SQL 3". In:

Proceedings of the 1999 ACM SIGMOD, vol. 28(1), pp. 131-138

EZEIFE, C., BARKER, K., 1995, "A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object Based System", *International Journal of Distributed and Parallel Databases*, Kluwer Academic Publishers, vol. 3(3), pp. 247-272

EZEIFE, C., BARKER, K., 1998, "Distributed Object Based Design: Vertical Fragmentation of Classes", *International Journal of Distributed and Parallel Databases*, Kluwer Academic Publishers, vol. 6(4), pp. 317-350

GARDARIN, G., GRUSER, J., TANG, Z., 1995, "A Cost Model for Clustered Object-Oriented Databases". In: *Proceedings of the 21st VLDB Conference*, Suíça, pp.323-334

KARLAPALEM, K., NAVATHE, S., MORSI, M., 1994, "Issues in Distribution Design of Object-Oriented Databases". In: ÖZSU, M. *et al.* (eds.), *Distributed Object Management*, Morgan Kaufmann Publishers Inc., São Francisco, EUA, pp. 148-164

KARLAPALEM, K., LI, Q., 2000, "A Framework for Class Partitioning in Object Oriented Databases", *International Journal of Distributed and Parallel Databases*, Kluwer Academic Publishers, vol. 8(3), pp. 333-366

KHOSHAFIAN, S., COPELAND, G., 1986, "Object Identity". In: *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'86)*, Portland, Oregon, pp. 406-416 – also In: *SIGPLAN Notices*, vol. 21(11)

KIM, W. (ed.), 1995, *Modern Database Systems,* ACM Press, USA

KOPPEL, M., FELDMAN, R., SEGRE, A., 1994, "Bias-Driven Revision of Logical Domain Theories", Journal of Artificial Intelligence Research, 1, AI Access Foundation and Morgan Kaufmann, pp. 159-208

LAWLER, E., WOOD, D., 1966, "Branch-and-bound methods: A survey", *Operations Research*, vol. 14, pp. 699-719

LAVRAC, N., DZREROSKI, S., 1994, *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood

LIMA, F., MATTOSO, M., 1996, "Performance Evaluation of Distribution in OODBMS: a Case Study with O2". In: *Proceedings of the IX International Conference on Parallel & Distributed Computing Systems (PDCS'96)*, ISCA – IEEE, Dijon, França, pp.720-726

LLOYD, J. W., 1987, *Foundations of Logic Programming*, 2nd extended edition, Berlin, Springer-Verlag

MAIER, D., *et al.*, 1994, "Issues in Distributed Object Assembly". In: M. ÖZSU *et al.* (eds.), *Distributed Object Management*, Morgan Kaufmann Publishers Inc., São Francisco, EUA

MATTOSO, M., RUBERG, G., BAIÃO, F., VICTOR, A., 2001, "Processamento de Consultas Orientadas a Objetos", Relatório Técnico ES-547/01, Programa de Engenharia de Sistemas e Computação, COPPE/UFRJ, Brasil, Abril

MAURO, R., MATTOSO, M., 1997, "Aspectos de Implementação de Servidores de

Banco de Dados Orientados ao Objetos". In: *Anais da XXIII Conferência Latino-Americana de Informática*, Valparaiso, Chile, Novembro, pp. 29-38

MAURO, R. *et al.*, 1997, "GOA++: Tecnologia, implementação e extensões aos serviços de gerência de objetos". In: *Anais do XII Simpósio Brasileiro de Banco de Dados (SBBD'97)*, Fortaleza, Brasil, Outubro, pp. 272-286

MEYER, L., MATTOSO, M., 1998, "Parallel query processing in a shared-nothing object database server". In: *Proceedings of the 3rd International Meeting on Vector and Parallel Processing (VECPAR'98)*, Porto, Portugal, pp. 1007-1020

MEIRA, W., SODERO, A., TAVARES, A., CARVALHO, M., 1996, "Parallel Branch-and-Bound: Design and Performance Understanding". In: *Proceedings of the VIII Simpósio Brasileiro de Arquitetura de Computadores - Processamento de Alto Desempenho* (VIII SBAC-PAD), Recife, Brasil, pp. 119-128

MCCORMICK, W., SCHWEITZER, P., WHITE, T., 1972, "Problem Decomposition and Data Reorganization by a Clustering Technique", *Operational Research*, vol. 20(5), pp. 993-1009

MOLINA, H., HSU, M., 1995, "Distributed Databases". In: W. KIM (ed.), *Modern Database Systems*, ACM Press, pp. 484-485

NAVATHE, S., CERI, S., WIEDERHOLD, G., DOU, J., 1984, "Vertical Partitioning Algorithms for Database Design", *ACM Transactions on Database Systems*, vol. 9(4), pp. 680--710

NAVATHE, S., RA, M., 1989, "Vertical Partitioning for Database Design: A Graphical Algorithm". In: *Proceedings of the 1989 ACM SIGMOD*, Portland, Oregon, EUA, pp. 440-450

NAVATHE, S., KARLAPALEM, K., RA, M., 1995, "A Mixed Fragmentation Methodology for Initial Distributed Database Design", *Journal of Computer and Software Engineering*, vol. 3(4)

OPITZ, D., SHAVLIK, J., 1995, "Dynamically Adding Symbolically Meaningful Nodes to Knowledge-Based Neural Networks", *Knowledge-based Systems*, vol. 8, pp. 301-311

ÖZSU, M., VALDURIEZ, P., 1999, *Principles of Distributed Database Systems*, 2nd edition (1st edition 1991), New Jersey, Prentice-Hall

O2 TECHNOLOGY, 1994, "A Technical Overview of the O_2 System", Technical Report 9, O_2 Technology, França

PAZZANI, M., 1996, "Review of Inductive Logic Programming", *Machine Learning*, vol. 23, pp. 103-108

RICHARDS, B., MOONEY, R., 1995, "Refinement of First-Order Horn-Clause Domain Theories", *Machine Learning*, vol. 19(2), pp. 95-131

RUBERG, G., 2001, *Um Modelo de Custos para o Processamento de Consultas em Bases de Objetos Distribuídos*. Tese de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, Maio

RUBERG, G., BAIÃO, F., MATTOSO, M., 2001, "A Cost Model for the Evaluation of Path Expressions in Distributed Object Databases", submetido para publicação, Novembro

SAVONNET, M., TERRASSE, M., YÉTONGNON, K., 1998, "Fragtique: A Methodology for Distributing Object Oriented Databases". In: *Proceedings of the International Conference on Computing and Information* (ICCI'98), Winnipeg, Canadá, pp.149-156, Junho

SHRUFI, A., 1994, "Performance of Clustering Policies in Object Bases", In: *Proceedings of the Third Conference of Information and Knowledge Management (CIKM94)*, pp. 80-87, Gaithersburg, EUA, Dezembro

TAVARES, F., VICTOR, A., MATTOSO, M., 2000, "Parallel Processing Evaluation of Path Expressions". In: *Proceedings of the XV Brazilian Symposium on Databases*, SBC, João Pessoa, Brasil, Outubro

WOGULIS, J., 1994, *An approach to Repairing and Evaluating First-Order Theories Containing Multiple Concepts and Negation*, Ph.D. Thesis, University of California, Irvine, EUA

WROBEL, S., 1996, "First Order Theory Refinement". In: L. De Raedt (ed.), *Advances in Inductive Logic Programming*, IOS Press

SU, S., HUANG, Y., AKABOSHI, N., 1998, "Graph-Based Parallel Query Processing and Optimization Strategies for Object-Oriented Databases", *International Journal of Distributed and Parallel Databases*, Kluwer Academic Publishers, vol. 6, pp. 247-285