# Interoperabilidade de Objetos em Sistemas de Bancos de Dados Heterogêneos

Elvira Maria Antunes Uchôa[1]
elvira@inf.puc-rio.br

*Rubens Nascimento Melo[2]*
*rubens@inf.puc-rio.br*

*Sérgio Lifschitz[3]*
*lifschit@inf.puc-rio.br*

PUC-RioInf.MCC45/96 Dezembro, 1996

## Resumo

Sistemas de bancos de dados heterogêneos têm sido apontados como uma das soluções mais viáveis para a integração de sistemas existentes, autônomos e diferentes, sem a necessidade de alterá-los. O uso de padrões para a orientação a objetos tem se apresentado como uma estratégia para coordenar, com flexibilidade, e integrar os recursos de processamento de informações, interligados por redes de comunicação. Na arquitetura dos sistemas de bancos de dados heterogêneos, estes padrões podem ser utilizados para a integração e interação dos componentes heterogêneos da federação, aumentando sua flexibilidade para adaptar-se a novos ambientes e sua capacidade de expansão. Este trabalho apresenta o HEROS, projeto que está sendo desenvolvido na PUC-Rio, e discute aspectos de interoperabilidade de objetos, mencionado a arquitetura CORBA, padrão para a orientação a objetos. Alguns projetos relacionados - MIND e Jupiter - também são considerados e comparados ao HEROS.

**Palavras-Chave:** Sistemas de bancos de dados heterogêneos, interoperabilidade, CORBA, HEROS.

## Abstract

Heterogeneous database systems have been considered one of the most feasible solution for the integration of existing systems, autonomous and different, without the need of changing them. Object oriented standards have been used as a strategy for a flexible coordination and integration of information resources connected through networks. In the heterogeneous database architecture, these standards may be used for integration and interaction among the heterogeneous components, increasing the federation flexibility and its expansion capacity in new environments. We present in this work the HEROS project, which has been developed in PUC-Rio, and discuss object interoperability issues considering the CORBA architecture, a object oriented standard. Other related projects - MIND and Jupiter - are also cited and compared to HEROS.

**Keywords:** Heterogeneous database systems, interoperability, CORBA, HEROS.

---

[1] *Doutoranda, bolsista CAPES*

[2] *Parcialmente apoiado por projeto integrado de pesquisa do CNPq 300103/93-1*

[3] *Parcialmente apoiado por bolsa de produtividade em pesquisa do CNPq 300048/94-7*

# 1. Introdução

Atualmente, os sistemas de informação informatizados consistem em ilhas de automação. Cada componente de software executa algum conjunto limitado de funções úteis, mas componentes não interoperam efetivamente. A integração que existe é insuficiente e não evolui apropriadamente com as tecnologias dos componentes. A escala deste problema abrange desde o contexto da estação de trabalho de um usuário até níveis *interusuários*, *interdepartamentos*, e *interorganizações*.

Integração de sistemas é uma questão cada vez mais importante. A medida que a tecnologia de software continua a evoluir, há uma tendência crescente para se construir sistemas a partir de sistemas informatizados preexistentes. Este tendência se deve, em grande parte, aos pacotes comerciais de software de alta qualidade, já disponíveis no mercado, que realizam a maioria das principais funções necessárias, nos sistemas existentes de informação.

As expectativas dos usuários quanto a interoperabilidade de sistemas têm aumentado dramaticamente, devido a plataformas com alta interoperabilidade disponíveis comercialmente como Microsoft Windows e Apple Macintosh. Infelizmente, soluções específicas de um fabricante não resolvem os desafios existentes para a integração de sistemas heterogêneos. Sistemas de informação estão cada vez mais heterogêneos; é raro encontrar uma empresa com uma base computacional monolítica (todos os usuários em uma mesma plataforma). Gradativamente, a interoperabilidade entre os sistemas de uma empresa precisa estender-se abrangendo o contexto de seus clientes e fornecedores. Heterogeneidade tende a aumentar a medida que os softwares tornarem-se mais portáteis [MOWB95].

Para criação de sistemas de informação integrados, grandes e distribuídos, a primeira fase consiste em prover interconectividade entre os sistemas, isto é, capacidade de trocar mensagens [NICO93]. Entretanto, interconectividade garante somente comunicação, não cooperação.

A segunda fase consiste em prover interoperabilidade entre os sistemas, isto é, capacidade dos recursos de computação interagirem para executarem tarefas conjuntamente. O grau de interoperabilidade pode variar desde uma simples capacidade de programação, como um mecanismo RPC (*Remote Procedure Call*), estendido com facilidades para tradução de dados, até formas mais avançadas que requeiram compreensão mútua, não somente a nível de sistemas ou programas, mas também a nível de semântica dos dados, como é o caso de Sistemas de Bancos de Dados Heterogêneos (SBDHs).

Para prover interoperabilidade, fatores como escala, heterogeneidade, gerenciamento de configuração e monitoramento de redes são críticos. A tecnologia de objetos facilita o gerenciamento desta complexidade [SOLE95], já que :

- fornece um meio comum onde estruturas de dados complexas, bem como especificações de comportamento, podem ser representadas e,
- acomoda, naturalmente, heterogeneidade e autonomia: heterogeneidade porque mensagens enviadas para os componentes distribuídos dependem apenas das interfaces dos componentes, indiferentes a suas características internas; e autonomia porque componentes podem mudar independentemente e transparentemente, desde que mantenham suas interfaces inalteradas.

O restante deste trabalho está organizado da seguinte forma: na seção 2, apresentamos o CORBA - padrão de interoperabilidade; na seção 3, o HEROS - SBDH em desenvolvimento na PUC-Rio; na seção 4, a aplicação do CORBA ao HEROS, em estudo; na seção 5, alguns trabalhos relacionados; na seção 6, nossas considerações finais, status de nossa pesquisa e alguns passos futuros; e, finalmente, na seção 7, as referências bibliográficas usadas em nosso trabalho.

## 2. Padrão do OMG

Embora a orientação para objetos apareça como uma solução para a interoperabilidade, atualmente, ela vem contribuindo para o problema, devido à diversidade de modelos de objetos [SOLE95].

Esta diversidade é justamente uma das razões que levaram várias empresas especialistas em sistemas orientados a objetos a cooperar, no sentido de definir um mínimo denominador comum, a ser seguido pelos ambientes orientados a objetos, disponíveis comercialmente.

Esta abordagem de mínimo denominador comum é evidente no desenvolvimento do CORBA, que aparece hoje com uma forte tendência de tornar-se um padrão, de fato, para as tecnologias emergentes de objetos distribuídos. A especificação do CORBA é um produto do OMG.

OMG - Object Management Group, contando hoje com cerca de 500 organizações membros, é uma associação comercial internacional, com fins não-lucrativos, constituído em abril de 1989 com o objetivo básico de desenvolver um padrão para interoperação de sistemas heterogêneos distribuídos (principalmente orientados a objetos), enfocando no recebimento e emissão de mensagens [HELA95]. A missão do OMG consiste em [MART94]:

- maximizar a portabilidade, capacidade de reaproveitamento e interoperabilidade de software. OMG é a organização que lidera mundialmente a iniciativa de proporcionar um quadro contextual e as especificações para os ambientes baseados em objetos disponíveis no mercado.
- prover uma arquitetura referencial com termos e definições sobre os quais se baseiem todas as especificações. Implementações dessas especificações tornar-se-ão disponíveis sob condições e termos justos e com eqüidade. OMG pretende criar padrões industriais para os sistemas baseados em objetos disponíveis comercialmente, focalizando no acesso remoto a objetos, via redes, no encapsulamento dos aplicativos existentes e nas interfaces dos bancos de dados baseados em objetos.
- prover um fórum aberto para discussões no que diz respeito à indústria em questão, à educação e à promoção da tecnologia baseada em objetos, endossada pelo OMG. OMG coordena suas atividades com organizações correlatas e atua como um centro de tecnologia e marketing para os softwares baseados em objetos.

O grande diferencial da abordagem dos padrões do OMG é que tais padrões são e serão baseados, tanto quanto possível, em produtos existentes, disponíveis comercialmente. Ao contrário da Open Software Foundation (OSF) da DEC (Distributed Computing Environment), OMG não produz softwares, mas somente especificações [VINO93]. Estas especificações são criadas usando idéias e tecnologias dos membros do OMG que respondem a *Requests For Information* (RFI) e *Requests For Proposals* (RFP) emitidas pelo OMG. A força desta abordagem reside em que a maioria dos participantes do meio comercial de computação distribuída orientada a objetos está entre as centenas de empresas que compõem o OMG.

## 2.1. Object Management Architecture - OMA

OMG tem um Modelo Referencial para uma Arquitetura de Gerenciamento de Objetos (Object Management Architecture - OMA - Figura 1). O objetivo desta arquitetura é o de capacitar diferentes softwares de fornecedores distintos a funcionarem conjuntamente. A intenção é a de influenciar o projeto dos componentes apenas no que tange ao alcance da interoperabilidade e, para tal, OMA procura definir, apenas em um alto nível de abstração, as várias facilidades necessárias para computação distribuída orientada a objetos. OMA abrange:

- como os objetos fazem solicitações e recebem respostas;
- as operações básicas que devem ser providas a cada objeto;
- as interfaces de objetos que provêem recursos comuns úteis a muitos aplicativos.

A Arquitetura de Gerenciamento de Objetos [OMG93] consiste em quatro partes principais, conforme ilustrado na Figura 1.

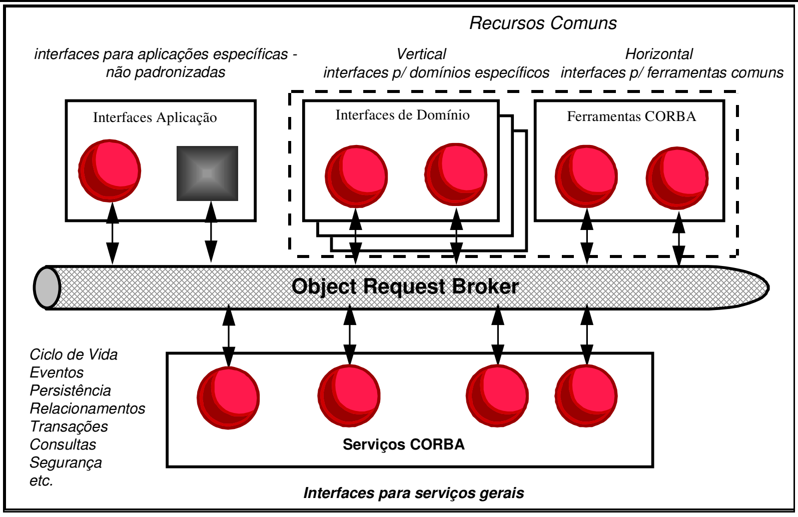



Figura 1: Arquitetura de Gerenciamento de Objetos (OMA) do OMG

⇒ <u>Application Objects</u>

*Aplicativos-Objetos* (*Application Objects*) são aplicativos para uso final que podem ser construídos por diversos fabricantes ou pelos departamentos de informática das empresas.

⇒ <u>Common Facilities</u>

*Recursos Comuns* (*Common Facilities*) [OMG95] são objetos e classes que provêem capacidades de propósito geral úteis a vários aplicativos. É uma biblioteca compartilhada de funções comumente utilizadas, incluindo funções que antes estavam em aplicativos e que agora estão migrando para sistemas operacionais ou para softwares de construção de sistemas. Vários Aplicativos-Objetos podem se tornar Recursos Comuns, quando eles se fizerem populares o bastante para se tornarem padrão de fato. As duas categorias de recursos comuns - *horizontal* e *vertical* - definem regras de engajamento que componentes de negócio precisam para colaborar.

Os *recursos comuns horizontais* [ORFA96] são independentes do domínio da aplicação e podem ser classificados em quatro tipos: 1) interfaces de usuário (*user interface*), reutilizáveis (por exemplo, editores de texto); 2) serviços de gerenciamento de informação (*information management*) - incluem armazenamento de documento e facilidades para troca de informações; 3) serviços de gerenciamento de sistema (*system* *management*) - definem interfaces para gerenciamento, configuração, instalação, operação e restauração de componentes distribuídos de objeto; 4) serviços de gerenciamento de tarefas (*task management*) - incluem workflow, transações longas, agentes, regras e e-mail.

Os *recursos comuns verticais* fornecem interfaces definidas em IDL (*Interface Definition Language*) para segmentos verticais do mercado como saúde, finanças, processamento de dados geoespaciais.

⇒ Common Object Services

*Serviços de Objeto* (*Common Object Services*) [OMG94b] são coleções de serviços que provêem as funções básicas para criar, armazenar, manter, estabelecer subtipos e gerenciar objetos. Eles incluem serviços de gerenciamento do ciclo de vida, gerenciador de transação, controle de concorrência, dentre outros.

Os Serviços de Objeto adotados pelo OMG abrangem a Especificação Comum de Serviço de Objeto (*Common Object Service Specification - COSS*). COSS [OMG94a] é uma série com vários volumes, cada um deles correspondendo a uma *Object Services Request for Proposal* (RFP). Atualmente, a *Object Services Task Force* do OMG já elaborou quatro das cinco RFPs planejadas. São elas [MOWB95]:

- RFP1 inclui serviços de notificação de evento, ciclo de vida, nomeação e persistência;
- RFP2 inclui serviços de concorrência, externalização, relacionamentos e transação;
- RFP3 inclui serviços de segurança e tempo;
- RFP4 inclui serviços de licença, propriedades e consulta.

A RFP5, prevista para final de 1996, deverá incluir serviços de gerenciamento de mudança, coleções, negociação e inicialização (*startup*).

⇒ Object Request Broker (ORB)

*Intermediador de Solicitações de Objetos* (*Object Request Broker - ORB*) é o núcleo, o coração do OMA. Ele fornece transparência de localização, ativação e comunicação de objetos, permitindo que os objetos se comuniquem independentemente de plataformas e técnicas específicas.

Um objeto faz uma solicitação de maneira padrão e o intermediador providencia o processamento da solicitação. O intermediador faz com que o método seja invocado e conduz os resultados ao solicitador.

O intermediador de solicitações parte da premissa de que os objetos de tipos diferentes devem ser capazes de se comunicar, mas não de revelar seu interior.

O intermediador de solicitações é responsável por todos os mecanismos necessários para encontrar a implementação para a solicitação do objeto, preparar a implementação do objeto para receber a solicitação e comunicar os dados que constituem a solicitação. A interface que o cliente (o solicitador) vê é completamente independente do local onde o objeto está localizado, da linguagem de programação em que está implementado, ou de qualquer outro aspecto que não esteja refletido na interface do objeto [OMG91].

É esperado que existam diversos tipos de implementações de intermediadores, de diferentes fornecedores. Um intermediador poderia ser projetado para operar em uma máquina, em um sistema em rede local, em sistemas em rede de um único fabricante ou em sistemas servidos por redes diversas ligando diferentes ambientes. Poderia ser visto pelo sistema operacional como um programa normal, ou poderia ser construído dentro do sistema operacional como um serviço de apoio para reforçar a segurança, a solidez e o desempenho. Poderia estar ligado a um sistema gerenciador de banco de dados (SGBD) baseado em objetos ou a um sistema de biblioteca de classes.

As diferentes implementações do intermediador terão que se comunicar umas com as outras. Como objetos, seus mecanismos internos podem assumir qualquer forma, mas as solicitações e as respostas que eles transmitem têm que ter um formato padrão.

Quando um cliente faz uma solicitação, ele nomeia um objeto, uma operação daquele objeto e nenhum, um ou vários parâmetros para aquela operação. A solicitação é enviada ao intermediador, que providencia para que a solicitação seja processada e que transmita os resultados ao solicitador. Isso pode ser uma operação simples. Contudo, às vezes, o intermediador tem que contatar os Serviços de Objetos (*Common Object Services*), a fim de encontrar a classe e a operação que está sendo solicitada. Os Serviços de Objetos podem incluir um serviço de dicionário de classes ou pode procurar bibliotecas de operações em tempo de execução.

Cada *operação* tem um identificador de operação, normalmente representada por seu nome. A operação pode ser implementada de formas diferentes em diferentes objetos.

Uma operação tem uma *assinatura* que descreve os valores legítimos de parâmetros de solicitação e os resultados devolvidos.

Um conjunto de operações possíveis que um cliente pode solicitar a determinado tipo de objeto é denominado de *interface* de um tipo de objeto. Um cliente conhece apenas a estrutura lógica de um objeto servidor, conforme definida na interface. As interfaces são definidas em IDL.

<u>CORBA</u>

A *Arquitetura para Intermediador de Solicitações de Objetos Comuns* (*Common Object Request Broker Architecture - CORBA* - Figura 3) pressupõe que o Modelo Nuclear (*Core Model*) do intermediador seja implementado de formas fundamentalmente distintas, mas que existem certos componentes comuns acima do modelo nuclear para prover as interfaces comuns. As interfaces comuns escondem as diferenças dos modelos nucleares e assim a arquitetura permite uma liberdade máxima na implementação do intermediador de solicitações [MOWB95].

O ORB pode ser implementado como função de uma biblioteca pré-compilada, como uma camada diretamente sobre RPC ou como um serviço de comunicação de mais alto nível. Esta flexibilidade permite que os fabricantes utilizem seus próprios serviços de rede já disponíveis. Alguns fabricantes (como IONA) estão implementando CORBA com base em RPCs compatíveis com ONC (*Open Network Computing*); outros fabricantes (como HP) estão implementando-o com base no OSF DCE; alguns outros (como SunSoft) estão ignorando a camada RPC e implementando CORBA como camadas de mais baixo nível.

A flexibilidade inerente ao interior do CORBA permite que o arquiteto do software separe as decisões de projeto das de implementação. Porque CORBA torna as propriedades de implementação transparentes, é desnecessário a colocação de tais detalhes no projeto da arquitetura. Isto libera o projetista para usar IDL para especificar todas as fronteiras da arquitetura do software e então escolher, mais tarde, os mecanismos internos de comunicação.

Em 1991, OMG publicou a revisão 1.1 [OMG91] da especificação de uma descrição concreta das interfaces e serviços que têm que ser fornecidos pelos intermediadores (ORBs) compatíveis.

CORBA permite que componentes de objeto, implementados por diferentes fabricantes, possam interoperar através de redes de comunicação e sistemas operacionais. CORBA fornece um intermediador (ORB) que permite que clientes invoquem métodos de objetos remotos estaticamente ou dinamicamente. Se uma interface do componente já está definida, o programa cliente pode ser “amarrado” estaticamente a um módulo (*stub*) para chamar seus métodos; por outro lado, o programa cliente pode descobrir como a interface trabalha em tempo de execução, consultando um Repositório de Interfaces (*Interface Repository*) especificado pelo OMG.

No final de 1994, OMG aprovou um conjunto de especificações - denominado CORBA 2.0 [ORFA96] - que define um *esqueleto* inter-ORB baseado em TCP/IP. CORBA 2.0 também especifica um serviço de comunicação inter-ORB opcional, baseado em OSF da DCE. A nova especificação do Repositório de Interface define extensões que permitem que componentes gerem identificadores globais para suas interfaces de forma a assegurar que elas são únicas a nível global.

CORBA também foi estendido com *serviços de objeto* adicionais e modulares (Figura 1), cada um especificado por especialistas líderes de mercado em áreas específicas de desenvolvimento de software. Cada módulo adicional fornece serviços essenciais de objeto para CORBA.

Atualmente, OMG está definindo padrões para criação, armazenamento, definição e nomeação dos objetos do CORBA. OMG também definiu um serviço de evento que permite que objetos comuniquem-se, em um relacionamento com baixo acoplamento. Recentemente foram finalizados os serviços adicionais de objeto baseados em CORBA - incluindo transações, externalização, controle de concorrência, relacionamentos, consulta, licenciamento, dentre outros. A medida que são concluídos, estes serviços estão sendo adicionados incrementalmente aos intermediadores compatíveis com CORBA.

<u>Alianças</u>

OMG criou importantes alianças para assegurar-se de que seus padrões sejam universais. A *International Standards Organization* (ISO) anunciou que irá sancionar, por referência, todos os padrões do OMG. O serviço persistente definido pelo OMG está fortemente alinhado com as especificações do *ODMG-93*[4] para banco de dados orientado a objetos. Padrões do OMG são recomendados pela *X/Open*[5] como parte da especificação do *Common Application Environment* (CAE), tendo incluído o CORBA no X/Open Portability Guide Release 4. IDL é a linguagem de especificação das APIs (Application Program Interface) das próximas versões de X Windows, denominada de X11R6 FRESCO. Os serviços de transação do objeto do CORBA podem interoperar com as transações procedurais definidas pela X/Open. OMG também está trabalhando com o grupo SysMan da X/Open nas interfaces para gerenciamento de sistemas baseados em ORB - incluindo segurança. Além disso, *Component Integration Laboratories* (CI Labs) - um consórcio de empresas que está desenvolvendo OpenDoc - adotou CORBA como seu modelo de objetos para comunicações intercomponentes. Tanto Taligent como OpenStep estão provendo gateways CORBA para suas comunicações externas ao objeto. CORBA também é

---

[4] *ODMG (Object Database Management Group) é um consórcio de fabricantes de SGBD orientados a objeto (SGBDOO) que está desenvolvendo uma interface padrão para seus produtos, ODMG-93. O padrão inclui uma arquitetura e definição comum para um SGBDOO, um modelo comum de objetos com uma linguagem de definição de objeto (ODL), uma linguagem comum de consulta a objetos (OQL) e amarrações (bindings) padronizadas para linguagem de programação, atualmente para C++ e Smalltalk. ODMG-93 é uma extensão do Persistent Object Service do CORBA, usando o modelo de objetos do OMG como base.*

[5] *X/Open é uma organização independente, mundial para sistemas abertos. Sua estratégia é combinar vários padrões em um ambiente compreensível de sistemas integrados denominado CAE, que atualmente contém um portfolio, em desenvolvimento, de APIs práticas. X/Open apoia suas especificações com um extenso conjunto de testes de conformância pelo qual um produto tem que passar para obter a marca X/Open.*

suportado por especificações de grupos, como *Petroleum Open Systems Consortium* e o *Open Geographic Information System (GIS) Foundation*. Finalmente, mesmo a Microsoft aderiu ao OMG - no final de 1994 - com uma proposta para um OLE "oficial" para o gateway CORBA.

A aceitação do CORBA pelo governo americano também está crescendo rapidamente. O Department of Defense Information Systems Agency (DISA), o National Institute of Standards and Technology (NIST), o National Institute of Health (NIH) e MITRE se juntaram ao OMG, e CORBA foi incluído nos padrões governamentais.

Implementações

Atualmente existem implementações do CORBA disponibilizadas por vários dos principais fabricantes de software: ObjectBroker da DEC, DistributedObjectsEverywhere da SunSoft, System Object Model da IBM, Distributed Smalltalk da HP, Orbix da Iona Technologies. Apple anunciou suporte futuro para CORBA. Microsoft está suportando CORBA através de sua aliança com DEC no desenvolvimento de um produto para desenvolvimento através de plataformas (*cross-platform*) denominado *Common Object Model*.

As implementações disponíveis de CORBA atendem a maioria dos ambientes de sistemas operacionais incluindo Solaris, OS/2, AIX, HP-UX, DG UNIX, Virtual Memory Systems (VMS), OSF/1, IRIX, Macintosh, Windows e Windows-NT.

Fabricantes como Integrated Computer Solutions, WordPerfect e Paragon Imaging anunciaram futuros produtos de aplicações compatíveis com CORBA. Expersoft, FORTE e outras empresas estão desenvolvendo interfaces CORBA para os principais produtos de banco de dados.

## *2.2. Interoperabilidade*

O foco da primeira especificação de CORBA foi na interoperabilidade de linguagem, possibilitada pela capacidade de descrever interfaces de objeto e tipos de dado associados de uma forma independente de linguagem de programação [ROY95]. Para implementar isto, a especificação do CORBA inclui a *Linguagem de Definição de Interface* (IDL - *Interface Definition Language*) do OMG, detalhada em 35 páginas de [OMG92].

IDL [MOWB95] é uma sintaxe independente de tecnologia para descrever encapsulamentos de objeto. Quando usada em arquiteturas de softwares, IDL é a notação universal que define as fronteiras dos softwares. Em IDL, são definidas *interfaces* que são compostas por atributos e assinaturas de operação. IDL suporta herança entre descrições de interface a fim de facilitar o reuso. Estas especificações são compiladas em arquivos de cabeçalho (*header files*) e programas de *stub* para uso direto pelos desenvolvedores. Compiladores IDL são incluídos em produtos ORB e permitem que programadores definam interfaces portáveis checadas pelo compilador.

Além dos arquivos de cabeçalho, o compilador IDL gera automaticamente programas *stub* e *skeleton* para cada interface (Figura 3). O programa cliente é *linkado* diretamente com o *stub* em IDL. Da perspectiva do cliente, o *stub* funciona como uma chamada de função local. Transparentemente, o *stub* fornece uma interface para o ORB que efetua a codificação e decodificação dos parâmetros da operação para os formatos de comunicação suportados pela transmissão. O programa *skeleton* é a implementação correspondente, do lado do servidor, da interface em IDL. Quando o ORB recebe a solicitação, o *skeleton* fornece uma chamada para uma implementação da função fornecida pelo servidor. Quando o servidor completa o processamento da solicitação, o *skeleton* e o *stub* retornam o resultado para o programa cliente, juntamente com qualquer informação de exceção. Exceções podem ser geradas pelo servidor ou pelo ORB em caso de erros.

IDL permite a especificação, para um componente, de: atributos (isto é, variáveis públicas), classes pais de quem ele herda, exceções, eventos tipados, práticas para gerar identificadores únicos globalmente para as interfaces e os métodos que uma interface suporta - incluindo parâmetros de entrada e saída e seus respectivos tipos de dado. IDL é usada para criar especificações de interfaces e não código. Componentes escritos em IDL são portáveis através de linguagens, ferramentas, sistemas operacionais e redes. Com a adoção da especificação do CORBA 2.0, em dezembro de 1994, estes componentes também podem ser interoperados através de intermediadores compatíveis com CORBA de diferentes fabricantes.

IDL é uma linguagem neutra a ser mapeada em diferentes linguagens de programação (Figura 2). OMG já definiu mapeamentos para C, C++ e Smalltalk. Atualmente está trabalhando no mapeamento para Ada, COBOL, dentre outras. Um mapeamento de linguagem define como um programa cliente ou servidor escrito em C++, por exemplo, manipula objetos e tipos de dado definidos em IDL. A vantagem é que o cliente e servidor podem ser escritos em diferentes linguagens de programação, trabalhando sobre as mesmas definições em IDL.

Além de interoperabilidade entre linguagens, IDL oferece portabilidade. Se todos os objetos e tipos são definidos em IDL e mapeados de uma forma bem definida para uma linguagem de programação, então uma aplicação portátil pode ser escrita dependendo apenas das definições. É por esta razão que a especificação CORBA é definida em termos de IDL.

Entretanto, a existência de mapeamento de IDL para várias linguagens de programação tem suas desvantagens. Existem casos onde ocorrem restrições na IDL que não existiriam se as coisas estivessem sendo definidas diretamente em uma linguagem de programação. Por exemplo, programadores de C++ podem sentir falta de sobrecarga. Um outro exemplo é a falta de suporte para ponteiros em IDL. Estas escolhas foram feitas para facilitar o mapeamento para múltiplas linguagens de programação e representar um mínimo denominador comum.

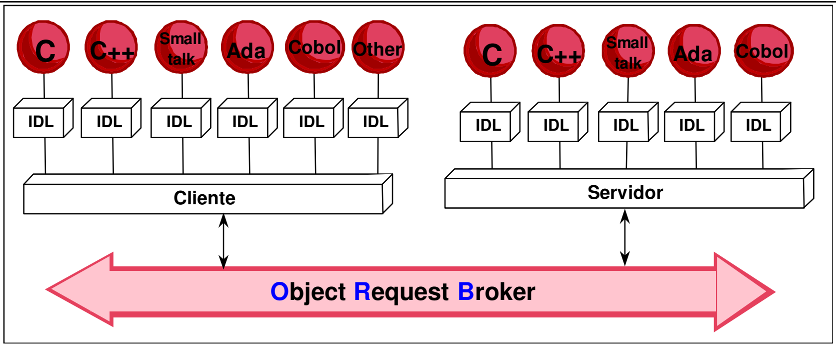



Figura 2: Amarrações da linguagem IDL do CORBA fornecendo interoperabilidade cliente/servidor

## 2.3. Arquitetura CORBA

CORBA (Figura 3) é uma especificação de uma arquitetura e interface que permite que aplicações façam solicitações a objetos, de uma forma independente e transparente, indiferente à linguagem, sistema operacional, ou considerações de localização [CATT94].

Os componentes da arquitetura do CORBA são ([CATT94], [SOLE95] e [HELA95]):

- uma *linguagem de definição de interface* (IDL - Interface Definition Language) que descreve as operações e atributos associados de uma interface de objeto em termos que o resto do sistema possa entender;

- uma completa representação dinâmica de interfaces disponíveis de todos os objetos em um repositório de interface (IR - *Interface Repository*);
- uma completa extensão de chamada dinâmica, permitindo a descoberta, em tempo de execução, de objetos, interfaces disponíveis, construção de mensagens de solicitação, e envio destas mensagens;
- um contexto de execução que permite o controle da execução;
- um "adaptador de objeto" para abstrair detalhes da implementação do objeto do substrato da mensagem.

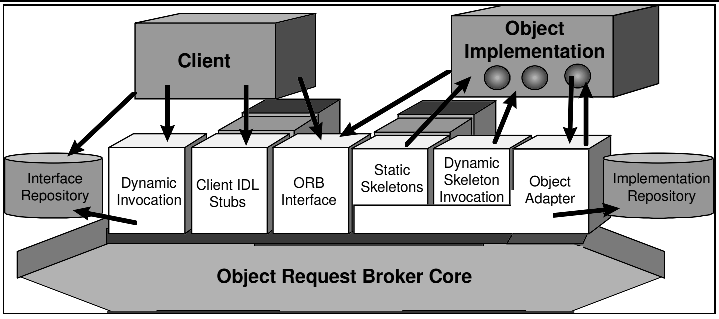



Figura 3: Estrutura de um ORB segundo CORBA 2.0

<u>Client e Object Reference</u>

Na arquitetura do CORBA (Figura 3), *cliente* (*client*) é uma entidade que deseja invocar operações de um objeto destino via ORB. Antes que um cliente possa fazer uma solicitação para um objeto, ele precisa obter uma *referência de objeto* (*object reference)* para aquele objeto. Um ORB usa referências de objeto para identificar e localizar objetos e então poder enviar solicitações para eles. Tão logo o objeto referenciado exista, o ORB permite que sejam efetuadas solicitações de serviços para ele. Referências de objeto podem ser tornadas persistentes a partir de uma solicitação inicial ao ORB para que converta-as em cadeias de caracter (*string*). Clientes podem armazenar estas cadeias de caracter em seus próprios arquivos de dados privados e mais tarde recuperá-las, solicitar ao ORB para que converta-as em referências de objeto e usá-las para fazer solicitações aos respectivos objetos.

<u>Object Implementation e Object Request Broker Core</u>

A *implementação do objeto* (*object implementation*) compreende o código e dados que implementam o comportamento do objeto destino. O ORB localiza uma implementação de objeto para uma solicitação, assegura que a implementação do objeto esteja pronta para receber a solicitação e transmite os dados da solicitação e os resultados entre o cliente e o objeto destino. É o *núcleo do ORB* (*Object Request Broker Core*) quem intermedia solicitações entre clientes e implementações de objeto. A intermediação envolve a localização do objeto destino, o envio da mensagem e amarração do método. Um biblioteca de funções, denominada de *Basic Object Adapter* (BOA), é usada para a localização e inicialização da implementação do servidor e para invocar o método apropriado para atender a solicitação do cliente.

<u>Object Adapter e Implementation Repository</u>

Um *adaptador de objeto* (*Object Adapter*) compreende a interface entre o ORB e a implementação do objeto. Adaptadores de objeto definem como um objeto é ativado, seja através da criação de um novo processo, de uma nova tarefa (*thread*) dentro de um processo existente, ou reusando um processo ou *thread* já existente.

Os adaptadores de objeto fornecem o ambiente de execução para instanciar objetos do servidor, passar solicitações para eles e atribuir-lhes identificações de objeto - o que é denominado em CORBA como *referências de objeto* (*object references*). O adaptador de objeto também registra as classes que ele suporta e suas instâncias de tempo de execução (isto é, objetos) com o Repositório de Implementação (*Implementation Repository*). É na inicialização que o adaptador de objeto do servidor deve instanciar os objetos do servidor que atendem as invocações remotas de método, efetuadas pelo cliente. Estes objetos de tempo de execução são instâncias das classes servidoras da aplicação.

O Repositório de Implementação é usado pelo ORB para localizar objetos ativos e para solicitar a ativação de objetos em um servidor em particular.

Existem muitos tipos potenciais de adaptadores de objeto. Podem existir diferentes adaptadores para usos de propósito geral, para integração de banco de dados, para integração de sistemas legados (*legacy*), e assim por diante. CORBA 1.1 define apenas o *Basic Object Adapter* (BOA), que tem que existir em todo ORB, mas reconhece a necessidade de outros tipos de adaptadores como um *Library Object Adapter* (LOA) para usar com implementações de objeto que precisem ficar co-residentes com as aplicações cliente, e um *Object-Oriented Database Adapter* que fornece uma interface de banco de dados orientado a objetos para o ORB.

O BOA é um adaptador de objeto de propósito geral. Quando uma solicitação de cliente especifica um servidor inativo, BOA automaticamente ativa o processo do servidor. A primeira responsabilidade do servidor é registrar sua implementação com o BOA. O BOA armazena este registro para usar em futuras solicitações de objeto. Depois que um objeto é ativado, pode receber solicitações de clientes através de chamadas de métodos no *skeleton* em IDL. Serviços do BOA incluem registro de implementação de objeto; mecanismos para gerar e interpretar referências de objeto; ativação e desativação de implementações de objeto; invocação de métodos através de *stubs*; passagem de parâmetros; e autenticação de cliente.

A expectativa é de que a maioria das implementações de objeto possam trabalhar com BOA porque seus serviços são flexíveis o bastante para acomodar vários tipos diferentes de implementações de objeto. CORBA define quatro políticas de ativação que especificam regras que uma determinada implementação segue para ativar objetos:

- implementações *persistentes* que são ativadas por alguma outra coisa distinta do BOA;
- implementações *compartilhadas* nas quais vários objetos coexistem em um mesmo programa (isto é, processo);
- implementações *não compartilhadas* nas quais somente uma implementação de objeto existe em cada programa;
- implementações de *método por servidor* onde cada operação suportada por um objeto é um programa separado.

Exceto para implementações persistentes, o BOA ativa automaticamente uma implementação de objeto se for efetuada uma solicitação para ele e ele ainda não estiver executando.

<u>Solicitação</u>

Toda a comunicação entre cliente e servidor restringe-se, em CORBA, a um tipo de mensagem denominado *solicitação*. Cada solicitação é enviada pelo cliente para o servidor e contém uma operação a ser executada e um objeto específico sobre o qual a operação deve ser executada. Os objetos e operações que um cliente pode solicitar e aos quais um servidor pode responder estão definidos pela interface que tanto o cliente quanto o servidor suportam.

IDL

Em CORBA, as interfaces são definidas usando *IDL*. Quando as especificações das interfaces em IDL são compiladas por um compilador CORBA, este gera automaticamente *stubs* para o cliente e *skeletons* para o servidor. Os *skeletons* do servidor permitem, ao ORB e a um adaptador de objeto, traduzir a solicitação do cliente para um método específico do servidor. Um *stub* do cliente mapeia definições de operações em IDL para um tipo de objeto contido em rotinas procedurais que são chamadas para invocar a solicitação.

O CORBA permite que o cliente tanto faça invocações estáticas quanto dinâmicas das interfaces.

Client IDL Stubs e Static Skeletons

Os *stubs* em IDL do Cliente (*Client IDL Stubs*) fornecem a interface estática para serviços do objeto. Estes *stubs* pré-compilados definem como clientes invocam os serviços correspondentes nos servidores. Da perspectiva de um cliente, o *stub* age como uma chamada local.

Um cliente tem que ter um *stub* para cada interface que ele use no servidor. O *stub* inclui código para realizar a comunicação. Isto significa que ele codifica (e decodifica) a operação e seus parâmetros em formatos de mensagem que será enviada para o servidor. Ele também inclui arquivos de cabeçalho que permitem que o cliente invoque o método no servidor a partir de uma linguagem de alto nível (C, C++, Smalltalk) sem se preocupar com protocolos internos ou questões como transporte de dados. O cliente simplesmente invoca um método de dentro de seu programa para obter um serviço remoto.

Os skeletons estáticos (*skeletons static*) fornecem a interface estática para cada serviço exportado pelo servidor. Também são chamados de *up-call interfaces*.

Dynamic Invocation e Interface Repository

Através da interface dinâmica (*DII - Dynamic Invocation Interface*), os clientes definem e constróem solicitações durante a execução. A invocação dinâmica é apoiada pelo repositório de interface. O *repositório de interface* (*Interface Repository*) armazena objetos representando informações em IDL em uma forma usada em tempo de execução. No recebimento de uma solicitação da aplicação, o cliente interroga o repositório de interface para identificar as interfaces capazes de satisfazer a solicitação - algumas das quais podem não estar disponíveis no momento em que o código do cliente foi compilado. O cliente pode usar as primitivas da interface para invocação dinâmica para construir a lista de argumentos de uma solicitação para o objeto destino selecionado.

ORB Interface

A Interface do ORB (ORB Interface) consiste de umas poucas APIs para serviços locais que podem ser de interesse da aplicação. Por exemplo, CORBA fornece APIs para converter uma referência de objeto em uma cadeia de caracter e vice-versa.

## 3. Projeto HEROS

Na área de bancos de dados, SBDHs têm sido projetados para gerenciar vários bancos de dados distribuídos, autônomos e heterogêneos. Conforme o grau de integração oferecido pelo SBDH, entre os sistemas de banco de dados (SBDs) componentes, ele pode ser classificado em [SHET90]: acoplamento forte e acoplamento fraco.

Um SBDH com acoplamento forte é composto por um conjunto de SBDs componentes, heterogêneos, cooperativos mas autônomos, integrados de tal forma na Federação que consultas e atualizações podem ser realizadas, de forma transparente à localização dos dados e aos caminhos de acesso.

O grupo de banco de dados da PUC-Rio, está desenvolvendo, sob o patrocínio do CNPq, um SBDH com acoplamento forte, orientado a objetos: o HEROS - HEteRogeneous Object System.

HEROS suporta operações sobre vários SBDs componentes, além de possuir seu próprio banco de dados. Quanto ao seu banco de dados, HEROS gerencia tanto o esquema quanto os dados associados, enquanto que, no que se refere aos bancos de dados componentes, ele apenas fornece acesso, sem materialização dos dados. Os SBDs componentes podem diferir quanto ao SGBD, modelo de dados, ambiente computacional e localização física (interligados, então, por sistemas de comunicação). HEROS proporciona [UCHO94a]:

⇒ uma completa transparência de localização e replicação para a manipulação de dados, como se o usuário estivesse acessando um único banco de dados centralizado;

⇒ transparência do modelo de dados local, evitando, para o usuário, a complexidade de conviver ao mesmo tempo com vários modelos de dados diferentes;

⇒ interferência mínima no processamento local dos SBDs componentes;

⇒ uma série de abstrações de dados de forma a permitir uma melhor representação das semânticas dos SBDs componentes;

⇒ consistência entre dados replicados em diferentes SBDs componentes, desde que tal dado esteja mapeado no esquema global, em suas várias ocorrências;

⇒ independência física dos dados, de forma que mudanças nos bancos de dados locais integrados pelo HEROS - como alteração de estrutura de armazenamento, localização na rede, migração entre gerenciadores distintos - não impactam as aplicações definidas sobre o esquema global, requerendo apenas alterações nas traduções e mapeamentos correspondentes, pelos administradores de banco de dados locais e global.

Um SBDH, segundo [THOM90], pode oferecer diferentes tipos de funcionalidades como: integração de esquemas, processamento de consultas globais, gerência de transações globais, funções administrativas e tratamento de diferentes tipos de heterogeneidades. Integração de esquemas está associada a visão lógica que o usuário global tem dos dados distribuídos. Gerência de consulta global trata a análise, otimização e execução de consultas que se referem a dados distribuídos. Gerência de transação global cuida da atomicidade, isolamento e durabilidade de transações em um ambiente distribuído. Funções administrativas incluem autenticação, autorização, definição e controle de restrições semânticas sobre os dados, e gerência de dicionário de dados e diretórios. Heterogeneidade inclui diferenças de hardware, sistemas operacionais, redes de comunicação, fabricantes de SGBDs e/ou modelos de dados.

No HEROS, já definimos os mecanismos e sistemática para integração de esquemas, gerência de transação global e tratamento de heterogeneidades. Cada um destes aspectos está detalhado a seguir.

## 3.1. Modelo de Dados

HEROS possui um esquema global integrador dos diversos esquemas de bancos de dados componentes [PACI93]. O modelo de dados utilizado globalmente é um modelo de dados orientado a objetos [UCHO94a]. A Figura 4 apresenta a arquitetura de esquemas do HEROS.

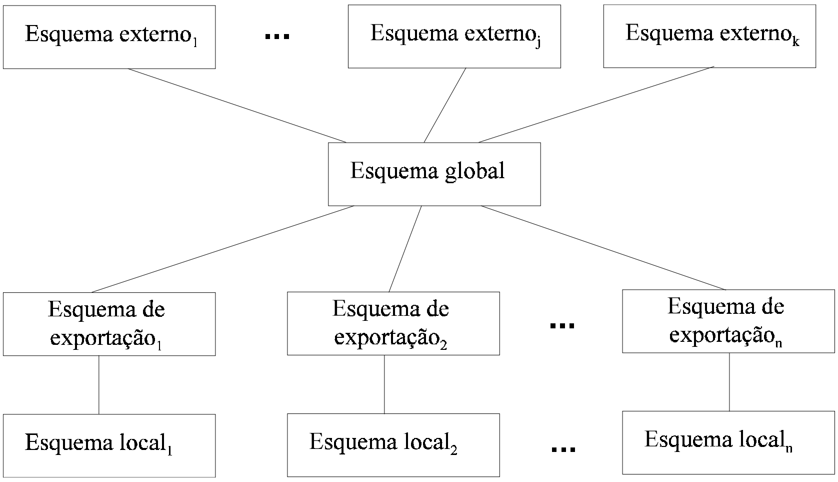



Figura 4 - Arquitetura de Esquemas do HEROS

Cada esquema local possui a descrição dos objetos segundo o modelo de dados utilizado pelo respectivo SGBD. O esquema de exportação de cada local contém a descrição dos objetos locais compartilhados, expressos no modelo de dados global do HEROS. O esquema global contém a descrição integrada dos objetos compartilhados pelos sistemas componentes. Cada esquema externo representa uma visão de aplicação do esquema global.

A integração dos componentes é realizada a partir da definição de esquemas de exportação [UCHO94b]. Esquemas de exportação são compostos a partir da definição do ambiente e dos próprios SGBDs componentes, como classes de objetos do HEROS, e pela definição de regras de tradução (mapeamento) de suas propriedades para os elementos do modelo de dados do HEROS, como serviços das classes. Ao se solicitar um acesso a dado ou operação de componentes, a partir do esquema global, este consulta os mapeamentos armazenados no esquema de exportação, com o objetivo de descobrir o caminho para a informação, e então poder recuperá-la para alteração/consulta ou execução, através dos procedimentos de acesso a dados, definidos para o sistema componente. A hierarquia de classes é usada, no HEROS, para representar a meta informação[6] associada a cada um dos sistemas componentes. A instanciação destas classes corresponde às informações específicas sobre as estruturas e aplicações de cada sistema a ser integrado, sendo feita automaticamente, através de regras, mediante a ocorrência do evento associado à execução de operações.

## 3.2. Gerência de Transações

A gerência de transações em um Sistema Gerenciador de Bancos de Dados

[6] *Informação sobre os SBDs componentes necessária para permitir o acesso e interoperação entre eles. Tal informação, conhecida por "meta informação", inclui nome e localização (endereço na rede), informações sobre o modelo de dados, direitos de acesso e etc.*

Heterogêneos tem por objetivo a obtenção de dados consistentes e a preservação da consistência global na presença de atualizações globais. Nos SGBDs tradicionais, o controle da execução concorrente de transações e a manutenção da consistência dos bancos de dados em caso de falhas é um problema bem definido e com diversas soluções. No entanto, estas soluções têm apresentado restrições no ambiente de SBDHs visto que, em tais ambientes, as transações tendem a ser de longa duração, e a estrutura de controle implícita no modelo tradicional de transação resultaria na manutenção das restrições de acesso a dados por períodos prolongados, reduzindo o nível de concorrência.

### 3.2.1. Modelo de Transação

Modelo de Transação do HEROS utiliza os conceitos de transações compensáveis e compensatórias presentes no modelo de Sagas [CHRY92], e o conceito de transação vital, presente no modelo de transações aninhadas, e na proposta de [MEHR92b], de modelo de transação para sistemas de múltiplos bancos de dados. Uma transação HEROS é uma transação com aninhamento aberto. O modelo permite a composição modular de transações, possibilitando que resultados parciais sejam observáveis por outras transações e evitando que, em transações de longa duração, o acesso aos dados seja bloqueado por longos períodos, favorecendo o uso compartilhado dos dados. Uma outra vantagem do uso de subtransações com aceitação de seus resultados é a maior tolerância a falhas, já que se reduz o número de locais participantes em um determinado instante na execução de uma transação global. A utilização de transações compensatórias possibilita a manutenção da consistência dos bancos de dados integrados, mesmo no caso de falha durante a execução de um protocolo de aceitação distribuída. A utilização do conceito de transação vital possibilita que transações planas tradicionais - não compensáveis - também possam ser executadas.

Uma transação *R* do HEROS é uma hierarquia em dois níveis de (sub)transações, com aninhamento aberto. O nó de mais alto nível é chamado de transação-raiz e os demais nós representam transações-componentes. Transações planas são vistas como casos degenerados das transações aninhadas. As transações-componentes de uma transação HEROS são transações pertencentes a um dos seguintes tipos:

- Compensável: uma transação deste tipo pode ter sua execução aceita e posteriormente desfeita através da execução de uma transação compensatória;
- Compensatória: uma transação deste tipo não pode logicamente falhar. A transação é executada para desfazer os efeitos de uma transação compensável cujos resultados tenham sido aceitos;
- Vital ou Pivô: uma transação deste tipo não é compensável e não é compensatória e, uma vez aceita, não pode ter seus resultados desfeitos através da execução de uma transação compensatória.

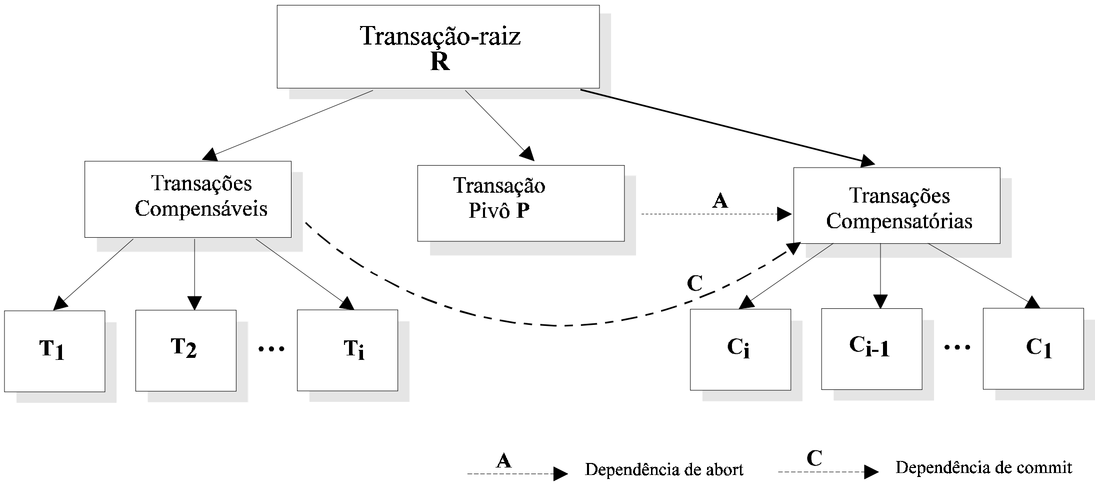



Figura 5 - Representação gráfica da estrutura de uma transação HEROS

Uma transação pode ter um número qualquer de transações compensáveis - e suas correspondentes transações compensatórias - e, sempre, uma única transação vital ou pivô (Figura 5).

Cada transação compensável tem o resultado da sua execução aceito tão logo tenham sido completadas as suas ações. Caso a transação a que pertence venha a ser cancelada, há necessidade de se mover o banco de dados global para um estado consistente, com a retirada dos efeitos das ações executadas e aceitas. A abordagem utilizada é a da execução de transação compensatória que desfaça semanticamente esses efeitos.

A transação pivô deve ser sempre executada e só pode ter o seu resultado aceito após terem sido executadas todas as transações compensáveis. A aceitação da execução da transação pivô implica na aceitação integral da transação.

A especificação da semântica operacional do Modelo de Transação do HEROS foi feita com o uso do formalismo ACTA [CHRY92], e está apresentada em detalhes em [SILV94].

### 3.2.2. Execução das Transações

As transações HEROS são executadas pelo *Sistema de Gerência de Transações* do HEROS (SGT), que efetua a interface entre as aplicações globais - a nível de federação - e os SBDs locais. As operações de uma transação global são submetidas para execução uma a uma. Uma nova operação só é submetida após o SGT retornar o resultado correspondente à execução da operação que a precede.

A correspondência entre conceitos pertencentes a esquemas dos diferentes níveis é expressa através de mapeamentos. Os mapeamentos são utilizados no processo de tradução de operações sobre objetos definidos em um esquema externo em operações sobre objetos definidos em um ou mais esquemas locais, e na composição dos resultados parciais e apresentação do resultado correspondente a uma operação global. No projeto do SGT, foram consideradas apenas as operações sobre objetos definidos no esquema global.

Para cada operação sobre dados recebida para execução, o SGT utiliza os dados sobre mapeamentos armazenados no dicionário de dados global, e gera uma árvore de operações correspondente. Esta árvore contém nos seus nós operações sobre os SBDs locais e as operações de composição e apresentação da resposta à operação global. Como os mapeamentos entre esquema global e esquemas locais são transparentes ao usuário global, é possível haver dependências de dados entre duas subtransações locais derivadas pelo SGT.

Cada operação sobre banco de dados local é submetida ao SGBD correspondente como componente de uma subtransação local. Para cada transação global existe no máximo uma única subtransação por local. As subtransações locais são executadas pelos SGBDs correspondentes da mesma forma que as transações locais comuns, isto é, um SGBD local não distingue entre subtransações globais e transações locais.

Na especificação do SGT, assumiu-se que cada SGBD utiliza o protocolo 2PL estrito [BERN87] para controle de concorrência, garantindo a produção de histórias serializáveis de todas as transações que executa. No caso da ocorrência de falhas, o SGBD possui a capacidade de recuperar um estado consistente.

Na ocorrência do cancelamento de uma subtransação pelo SGBD local - como no caso de solução de bloqueios mútuos - a informação sobre o cancelamento é fornecida, não sendo responsabilidade do SGBD a sua ressubmissão.

A execução de um comando global é precedida pela aquisição de bloqueios para a transação em andamento. A possibilidade de conflitos semânticos entre as operações é verificada através do exame das tabelas de compatibilidade definidas para cada classe de objetos pelo administrador de banco de dados global quando da criação do esquema global. Estas tabelas são criadas com a aplicação do princípio da comutatividade entre operações, isto é, verificando se o efeito final é o mesmo, independentemente da ordem de sua execução.

Uma operação sobre um objeto pode resultar na execução indireta de outras operações, sobre o mesmo ou outros objetos. As descrições de quais classes de objeto são afetadas e quais operações são indiretamente executadas fazem parte da especificação de cada método.

A granularidade das restrições de acessos, a nível global, é a da classe de objeto a que pertence a operação.

Quando os requisitos de acesso da operação global não podem ser integralmente atendidos, a operação entra em uma lista de espera, aguardando condições para sua execução.

### 3.2.3. Arquitetura do Sistema de Gerência de Transações

O SGT é o componente funcional do HEROS que implementa a gerência de transações. Este sistema é composto por um único coordenador - o Subsistema de Gerência Global de transações (SGG) - e por um conjunto de componentes locais - os Subsistemas de Gerência Local de transações (SGLs) - um para cada local participante do HEROS (Figura 6).

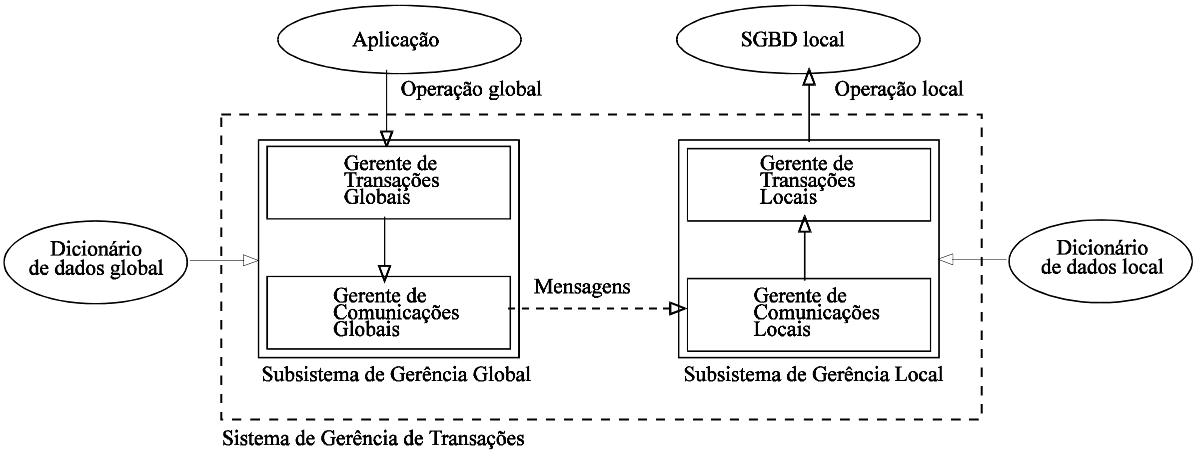



Figura 6 - Arquitetura do Sistema de Gerência de Transações do HEROS

O Subsistema de Gerência global de Transações - SGG é responsável pela interação do HEROS com as aplicações, implementando uma interface que consiste das operações básicas {*Begin_trans_R, Begin_trans_T, Begin_trans_P, Begin_trans_C, Exec_Method, Commit, Abort, Abort_R, EOT*}. Uma transação compensatória é derivada pelo SGT a partir das operações executadas por uma transação compensável e concluída com sucesso. A transação compensatória é executada quando torna-se necessário desfazer semanticamente os efeitos da transação compensável original sobre o banco de dados global.

Os Subsistemas de Gerência Local de Transações são os responsáveis pela interação com os SGBDs locais. Representam, em cada local, o sistema global, controlando a submissão de comandos para execução pelo SGBD respectivo e representando o SGBD no processo de aceitação das subtransações globais, de forma semelhante do método de agentes 2PCA [VEIJ92].

A descrição detalhada dos componentes funcionais do SGT e a especificação formal do modelo de execução das transações estão em [SILV94].

## 4. Aplicação do CORBA ao HEROS

Os componentes do sistema de gerência de transações do HEROS foram especificados como módulos funcionais [SILV94] e implementados como funções na linguagem de programação C em ambiente UNIX. Os componentes que deveriam residir em um mesmo local físico foram *linkeditados* juntos gerando um arquivo executável. A comunicação entre componentes remotos foi implementada como chamada de procedimentos remotos (RPC). A aplicação está residindo no mesmo local que o HEROS.

Nesta proposta, deparamo-nos com uma série de problemas e desafios:

<u>Quanto aos componentes</u>

- dificuldade para compreensão das funcionalidades e interações dos componentes do sistema de gerência de transação;
- falta de um protocolo implementado que indicasse a interface dos gerentes de execução específica, obrigando que se consultasse a documentação a cada novo SGBD incorporado à federação;
- necessidade de compilar e *linkeditar* todos os componentes a cada vez que se acrescenta ou altera um módulo.

Quanto à comunicação

- comunicação entre a aplicação global e o HEROS. Seria desejável que a aplicação pudesse residir em um local físico diferente ao do HEROS;
- o mecanismo do RPC não permite dois procedimentos remotos com mesmo nome, o que obrigou à implementação de mais um nível intermediário de módulos entre o sistema global e o local que permitisse a resolução de endereços sem comprometer a genericidade do componente global.

A solução encontrada, capaz de acrescentar maior flexibilidade e extensibilidade ao HEROS no que se refere aos aspectos analisados acima, foi incorporar, à sua arquitetura, o CORBA. Nesta proposta, todos os componentes do sistema de gerência de transação são especificados como objetos do CORBA, como apresentado na Figura 8, onde o caminho entre os objetos corresponde ao ORB. Também são especificados como objetos do CORBA todas as classes utilizadas para integração de esquemas (Figura 7). As vantagens desta alternativa são:

1. a representação sobre a forma de objetos propicia uma melhor visualização/ compreensão dos componentes e suas funcionalidades;
2. a propriedade do encapsulamento facilita o refinamento/evolução dos componentes do HEROS já que oculta sua implementação que pode ser alterada sem impactar seus usuários, desde que a interface seja mantida;
3. o mecanismo de herança permite que novos gerentes de execução específica sejam criados a partir de uma definição genérica de gerente de execução específica, enquanto que o polimorfismo faz com que diferentes objetos da classe respondam diferentemente a uma mesma invocação de método;
4. qualquer aplicação que siga os padrões do OMG pode usar os serviços do HEROS, isto é, executar procedimentos sobre uma federação, usufruindo da possibilidade de abstrair-se de questões como redundâncias, conflitos semânticos ou de representação, e localização dos elementos que a federação integrar;
5. enquanto objetos do CORBA, os *componentes do HEROS* podem residir em qualquer local físico e serem substituídos, ficando sobre a responsabilidade do CORBA administrar esta comunicação, o que aumenta a extensibilidade do HEROS, isto é, sua capacidade de adaptar-se a novos requisitos/estruturas de comunicação sem a necessidade de alterações internas;
6. a *aplicação e o componente global do HEROS* podem residir em qualquer local, interligados por qualquer forma de comunicação, heterogênea ou não, ficando sobre a responsabilidade do CORBA administrar esta comunicação;
7. a Interface de Invocação Dinâmica do CORBA permite que solicitações a objetos sejam efetuadas sem conhecimento anterior das assinaturas das operações. O servidor executa, em tempo de execução, checagem das solicitações efetuadas, sem necessidade de recompilar o código do cliente na presença de novas funções disponíveis no servidor.

Dentre os serviços comuns (*Common Object Services*) previstos pela OMA, destacam-se:

- serviço de controle de concorrência - fornece gerenciador de bloqueio que pode obter bloqueios em favor de transações ou *threads*.

- serviço de transações - fornece coordenação *two phase commit* (2PC) entre componentes recuperáveis usando transações planas ou aninhadas.

Avaliamos a possibilidade de utilizarmos tais serviços na arquitetura do HEROS ao invés de especificarmos classes próprias para tais fins, no entanto, chegamos a conclusão de que sua utilização direta não é possível devido a:

- serviço de transação

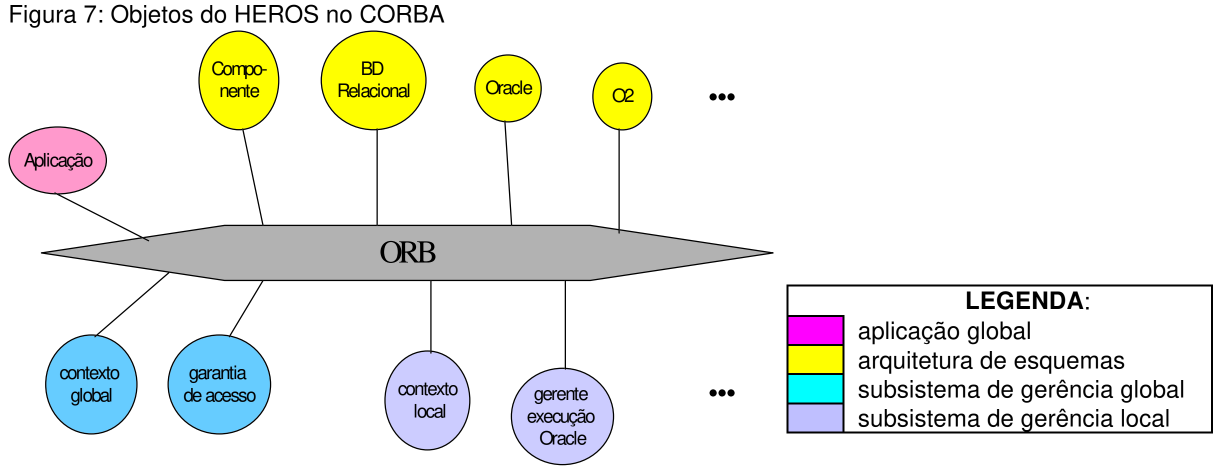
  ⇒ OMA trabalha apenas com os modelos de transação plana e aninhada, enquanto que o modelo de transações do HEROS, embora use transações aninhadas, abrange ainda o conceito de transação compensatória, do modelo Sagas, acrescido de um outro tipo de transação - a pivô [SILV94];

  ⇒ a idéia do OMA de implementar o serviço de transação como um conjunto de classes afins [ORFA96], no entanto, é bastante interessante e será aproveitada na especificação do serviço de transação do HEROS (Figura 8).

- serviço de controle de concorrência

  ⇒ o serviço de controle de concorrência do HEROS, denominado como *Gerência de Garantia de Acesso* [SILV94], prevê algumas funcionalidades inexistentes no serviço de controle de concorrência do OMA [ORFA96] como, por exemplo, o serviço de *Manutenção de Fila de Espera pelas Garantias de Acesso*. No entanto, estamos estudando a possibilidade de criar a *Gerência de Garantia de Acesso* do HEROS como uma especialização das classes relativas ao serviço de controle de concorrência da OMA.

Figura 7: Objetos do HEROS no CORBA



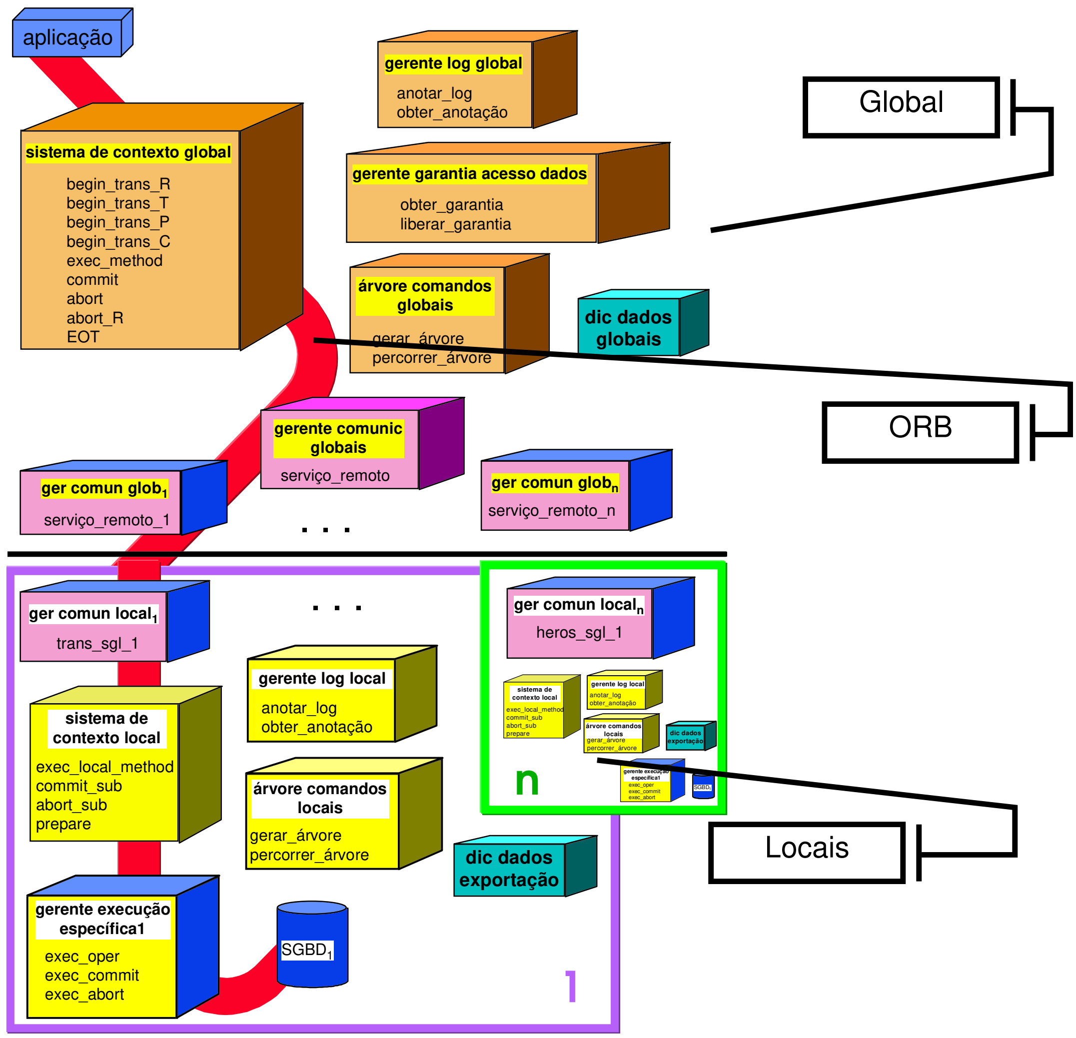



Figura 8: Sistema de Gerência de Transação do HEROS usando CORBA

## 5. Trabalhos Relacionados

CORBA vem sendo usado em inúmeros projetos para integrar sistemas de banco de dados. No entanto, somente poucos projetos estão relatados na literatura.

No EDS-Navigator-Project [QUAC94] está sendo desenvolvido um núcleo de controle para um sistema de projeto de equipamentos. A abordagem adotada neste projeto para acessar banco de dados, usando CORBA, é similar à relatada em [KOSC96], para um sistema de informações geográficas (GIS - Geographic Information System), em desenvolvimento na Alemanha.

Para um sistema de informação para clientes da World Wide [HAST94] está sendo usado uma combinação da tecnologia WWW e CORBA. Neste projeto está sendo desenvolvida uma camada de objetos que passa, a partir de formulários HTML, informações em pares compostos por nome e valor, para chamadas de métodos de objetos, como uma abordagem genérica para envolver fontes de informação.

Júpiter [MURP94a ] e MIND [DOGA95d ] são protótipos de implementações de *multidatabases* que usam CORBA como uma camada de interoperabilidade. Tais sistemas tratam consultas arbitrárias em SQL, logo, um método genérico de acesso para sistema de banco de dados teve que ser adotado. Este método usa interfaces genéricas para consulta aos sistemas de banco de dados componentes, tal como interface a nível de chamadas (*call-level*), SQWL dinâmico ou um intérprete de SQL.

A seguir, alguns destes projetos são apresentados com mais detalhes.

## 5.1. Projeto MIND

METU INteroperable Database System (MIND), parcialmente suportado por Motorola Inc. - USA e Sevgi Foundation - Turkey, é um sistema de múltiplos bancos de dados que trata interoperabilidade entre SBDs federados e heterogêneos. O principal objetivo do projeto é desenvolver uma arquitetura e um conjunto de ferramentas que propiciem interoperabilidade entre SGBDs existentes, relacionais e orientados a objeto. O projeto pretende facilitar o trabalho cooperativo e corporativo de sistemas existentes, preservando sua autonomia, enquanto oferece, ao usuário, uma única visão integrada. Maiores detalhes são apresentados em [DOGA95d] e [DOGA95c ].

| Projeto | MIND |
|---|---|
| Arquitetura do sistema<br>(Figura 9) | Baseada no modelo de gerenciamento de objetos distribuídos do OMG, o que permite que clientes (usuários e programas de aplicação) possam acessar bancos de dados que residam em qualquer local no ambiente transparentemente e sem qualquer conhecimento de localização ou formato de consultas ou operações envolvidas, visualizando apenas objetos homogêneos através de uma interface comum. O modelo de dados comum da interface é ODMG-93 e a linguagem de consulta da interface é a linguagem de consulta do OMG denominada OQL.<br>Atualmente apenas um subconjunto de OQL está operacional. |
| Implementação [DOGA96c] | Implementado em SunOS, no topo de um ORB compatível com CORBA, o DEC's ObjectBroker que roda em várias plataformas como Windows, SunOS, AIX, Open VMS. |
| Tipo de SBDH | Acoplamento forte. |
| Modelo de dados | ODMG-93 [CATT94], com extensões para suportar modelos de dados relacional e orientado a objetos e com construções que enriquecem os esquemas dos componentes possibilitando a integração semântica. |
| Arquitetura de esquemas | Usa arquitetura de esquemas em 4 níveis: local, exportação, federado e externo.<br>Os objetos das classes dos esquemas de exportação e federado, ao invés de serem armazenados pela própria classe, são derivados a partir de objetos de outras classes. |
| Linguagem | A linguagem de consulta da interface é a linguagem de consulta do OMG denominada OQL<br>Usa uma linguagem de definição de objetos (ODL - Object Definition Language) |
| SBDs Componentes [DOGA96b] e [DOGA96a ] | SBDs relacionais e orientados a objeto;<br>atualmente Oracle7, Sybase, Adabas D e MOOD (METU Object-Oriented Database system [DOGA94a], [DOGA94b], [DOGA95a] e [DOGA95b]) |

| Projeto | MIND |
|---|---|
| Componentes da arquitetura do MIND | classe Query Processor (QP)<br>classe Global Database Agent (GDA)<br>classe Schema Information Manager (SIM)<br>classe Local Database Agent (LDA)<br>classe Database Object<br>interface gráfica |
| Integração de esquemas | A integração de esquemas de exportação é executada manualmente com o uso de uma linguagem de definição de objetos (ODL - Object Definition Language) que é baseada na linguagem de definição de interface do OMG. O DBA (administrador de banco de dados) constrói o esquema integrado como um visão sobre esquemas de exportação. As funcionalidade da ODL permitem seleção e restruturação de elementos de esquema a partir dos esquemas locais.<br>Há a intenção de se fazer com que o esquema de exportação seja traduzido automaticamente para o modelo canônico. A integração de esquemas de exportação no esquema federado deverá ser efetuada através de mecanismo de definição de visão, que deverá especificar regras e rotinas de conversão para resolver heterogeneidades de dado e comportamento. |
| Gerência de transação | MIND adotou a gerência de transação para sistemas autônomos federados desenvolvido em [HWAN94].<br>A existência de SGBDs com protocolos de *commit* diferentes ou não existentes tornou necessário comandos de compensação, a nível da linguagem, que precisam ser especificados pelo programador da aplicação. As facilidades para gerência de transação abrangem comandos de *begin, prepare to commit, commit, rollback* e *end* para as subtransações, em uma versão simplificada do protocolo TP.<br>O sistema MIND usa um modelo de transação aninhada em dois níveis: transações locais e globais.<br>O gerente de transações é executado a nível do topo do CORBA, tratando gerência de transação, controle de concorrência e ordenação de eventos. |

| Projeto | MIND |
|---|---|
| Funcionamento | Foi definida uma interface de um objeto genérico *Database Object*, acessível através do CORBA, e foram desenvolvidas várias implementações desta interface, uma para cada SGBD componente, que são Oracle7, Sybase, Adabas D e MOOD.<br><br>Quando uma aplicação cliente executa uma consulta global em SQL que acessa vários bancos de dados, esta consulta global é decomposta em subconsultas locais e estas subconsultas são enviadas via ORB que transfere-as para o respectivo servidor de banco de dados na rede. No local do servidor, a subconsulta local é executada usando as rotinas correspondentes da interface de chamada e o resultado é retornado para o cliente novamente através do ORB (Figura 10). Os resultados retornados para o cliente a partir dos respectivos servidores podem ser processados pelo cliente, se necessário. |

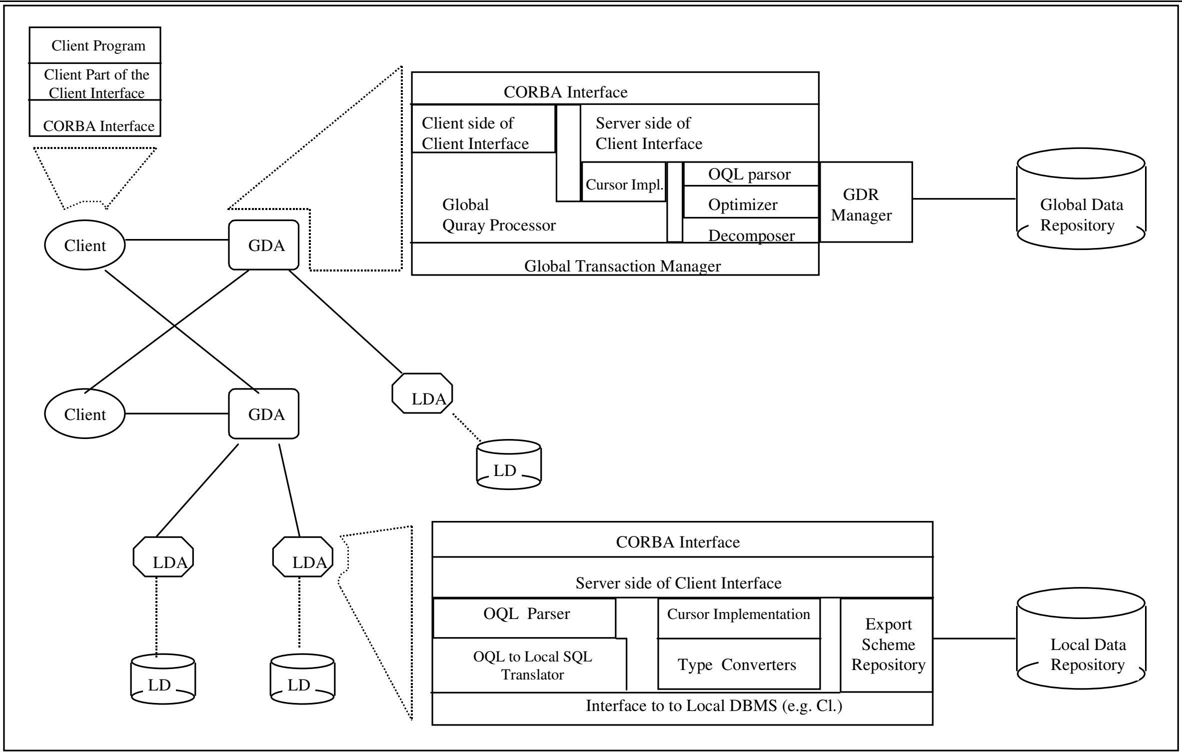



Figura 9: Arquitetura do MIND

Clientes acessam o sistema MIND através de uma coleção de *Global Database Agents* (GDA) [DOGA96c]. Os objetos da classe GDA são responsáveis por analisar e decompor as consultas de acordo com a informação obtida pelo *Schema Information Manager*, que contém informação sobre o esquema global. Eles são responsáveis pelo gerenciamento de transação global, que assegura a serializabilidade de transações globais planas e aninhadas sem violar a autonomia dos SGBDs locais [HALI95], e por programar e controlar operações entre locais diferentes. Objetos da classe GDA tratam *commit* global e *abort* global usando protocolo 2PC sobre objetos da classe LDA.

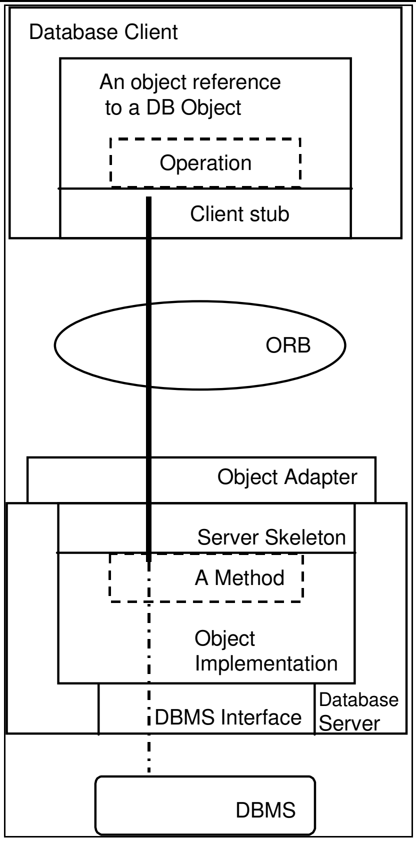



Figura 10: Invocação de operação de uma instância da Object Database através do ORB

Para um cliente o GDA é um servidor. Do ponto de vista do GDA, os *Local Database Agents* (LDA) são servidores. Os objetos da classe LDA são responsáveis por manter esquemas de exportação fornecidos pelos SGBDs locais, traduzir as consultas recebidas na linguagem de consulta global para a linguagem de consulta local e fornecer uma interface para os SGBDs locais.

Cada banco de dados corresponde a um objeto da classe *Database Object* definida no CORBA. A classe *Database Object* transfere as solicitações do cliente para os SGBDs componentes, usando as *Call Level Interfaces* destes SGBDs. As interfaces de chamadas (*call interfaces*) destes SGBDs suportam definição de dado em SQL, manipulação de dado, consulta e facilidades para controle de transação, implementadas em C++.

Atualmente, MIND usa um otimizador dinâmico de consultas. Depois que uma consulta é decomposta, as subconsultas globais são enviadas para os respectivos objetos LDA. O processo de otimização começa depois que se torna disponível o primeiro resultado parcial a partir de um objeto LDA. Para otimizar as operações inter-locais entre estes resultados parciais, está sendo usado um mecanismo estatístico de inferência [OZCA96 ]. Operações inter-locais estão sendo realizadas por objetos da classe *Query Processor*. Vários objetos QP podem estar rodando em paralelo, se necessário.

## *5.2. Projeto Jupiter*

O projeto Jupiter[7] tem por principal objetivo especificar formalmente e construir um protótipo de um “interoperador” que permita que sistemas autônomos e, possivelmente, heterogêneos cooperem e compartilhem informação de uma forma controlada. Jupiter enfoca especialmente em interoperabilidade entre banco de dados e, para tal, descreve uma arquitetura para interoperabilidade de banco de dados que consiste em um conjunto de serviços e uma linguagem para múltiplos bancos de dados, a *Jupiter Interoperator Language* (JIL), que permitam que provedores de informação construam sistemas de informação fracamente acoplados, autônomos e interoperáveis Maiores detalhes são apresentados em [MURP94b] e [MURP95].

| Projeto | Jupiter |
|---|---|
| Arquitetura do sistema | O objetivo básico de Jupiter é propiciar interoperabilidade entre SBDHs. O núcleo do sistema Jupiter consiste em um sistema avançado de dicionário [MURP94c ] e uma linguagem para múltiplos bancos de dados que fornece um conjunto de serviços |

7 *Parcialmente fundamentada no ESPRIT Programme da Commission of the European Union como parte do EDITH Project 7507, European Distributed Information Technology for Healthcare.*

| Projeto | Jupiter |
|---|---|
| | sobre o CORBA.<br>Júpiter foi implementado como uma camada de sistema de objetos distribuídos.<br>Cada esquema federado é visto como um objeto que propicia, a um conjunto de usuários, acesso a algum conjunto federado de bancos de dados participantes. Os objetos distribuídos podem ser acessados pelo componente Jupiter/MDD (Multidatabase Dictionary) [MURP93b].<br>As primitivas básicas de JIL foram implementadas e encapsuladas por objetos Orbix que podem estar distribuídos através do sistema e serem acessados transparentemente a partir de outros nós.<br>A camada do Interoperador tem um duplo papel que é fornecer persistência para o metadado da federação e também fornecer distribuição para este metadado, juntamente com a distribuição dos serviços de Jupiter, pelos nós remotos da federação. |
| Implementação | Implementado no topo de um ORB compatível com CORBA, o Orbix da Iona Technologies. |
| Tipo de SBDH | Acoplamento fraco. |
| Modelo de dados | Modelo formal de objetos, descrito na notação Z [MURP92] |
| Arquitetura de esquemas | Usa uma arquitetura de esquemas em 4 níveis: esquema local, esquema de participação, esquema de exportação, esquema federado. |
| Linguagem | Jupiter Interoperator Language (JIL) [MURP93a ] é uma linguagem para ser usada pelos administradores de bancos de dados locais para gerenciar os recursos da federação<br>Pretendem definir uma interface orientada a ícones ou formulários para uso por usuários finais. |
| SBDs Componentes | SBDs relacionais |
| Componentes da arquitetura do Jupiter (Figura 11) | System Services (*pronto*)<br>Application Services (*incompleto*)<br>Dictionary (Metadata) Services (*pronto*)<br>Jupiter Query Services (*incompleto*)<br>Data Movement Services (*pronto*)<br>Object Services (*futuro*)<br>Negotiation Services (*incompleto*) |
| Integração de esquemas | A resolução de heterogeneidade sintática e semântica é efetuada com a ajuda de comandos de amarração (*bind*) da linguagem JIL. |
| Gerente de transação | Não possui gerente de transação por ser um SBDH com acoplamento fraco. |

| Projeto | Jupiter |
|---|---|
| Funcionamento [MURP94d] | Jupiter suporta uma federação de bancos de dados com acoplamento fraco, propiciando seu gerenciamento através de sua linguagem multidatabase (JIL).<br><br>Os usuários podem especificar o dado que querem compartilhar com outros usuários através de esquemas de exportação. Outros usuários podem consultar os esquemas de exportação e construir uma variedade de esquemas da federação para uso próprio. Esquemas da federação contém informações que possibilitam que os bancos de dados heterogêneos possam ser manipulados através da linguagem JIL.<br><br>Consultas e protocolos de troca de dados são projetados como métodos de Jupiter, definidos sobre o modelo de dados comum. Todos os SGBDs participantes podem acessar estes métodos através da linguagem JIL. O usuário pode solicitar acesso direto a uma fonte de dados ou optar por uma *quasi-copy*. Jupiter e JIL suportam o protocolo de negociação descrito em [ALON88].<br><br>Um usuário pode solicitar informação que está *aproximadamente atualizada*. O usuário pode especificar tempo, versão e restrições aritméticas na informação recebida via *quasi-copy*.<br><br>O usuário pode submeter consultas a um processo do servidor Jupiter a partir de seu nó local usando JIL, a linguagem de consulta do SGBD local ou a linguagem de consulta de qualquer outro SGBD participante.<br><br>Uma consulta deve ser submetida em um nó da federação e o componente *Jupiter Query Services* usa informação da federação armazenada em seu dicionário distribuído para decompor a consulta em subconsultas que podem ser enviadas para bancos de dados participantes. O resultado destas subconsultas é combinado pelo componente *Jupiter Data Management Services*. |

Um servidor CORBA (Orbix) é necessário em cada local onde o Interoperador Jupiter estiver sendo executado. Os SGBDs participantes da federação são encapsulados como clientes e servidores Orbix. Solicitações remotas e transferências de *quasi-copy* são tratadas por chamadas ao Jupiter, que estabelece um objeto representante local para a solicitação remota. As solicitações a locais remotos são tratadas como se a fonte remota do dado fosse, de fato, uma fonte local; o objeto representante encaminha a solicitação para o local remoto do Jupiter e, quando a solicitação é processada, o dado é devolvido pelo representante no *site* local. O objetivo é que o usuário final tenha uma única imagem do sistema, independentemente da quantidade de SBDs heterogêneos participantes.

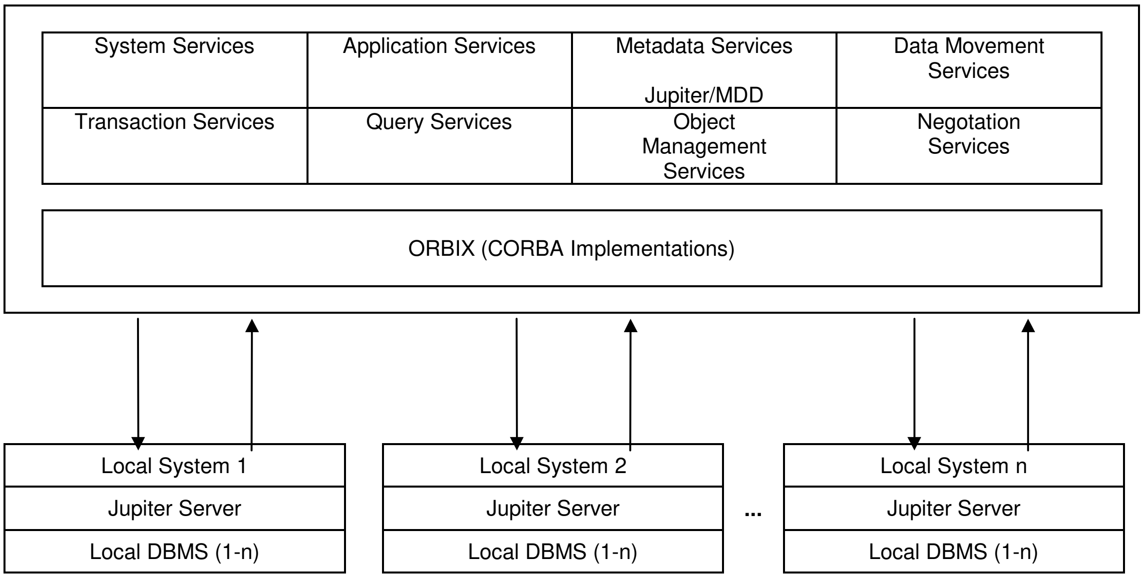


Figura 11: Arquitetura da implementação do Jupiter

## 6. Conclusão e Trabalhos Futuros

CORBA, em desenvolvimento pelo OMG, é uma especificação e uma arquitetura que possibilita que objetos em sistemas heterogêneos possam ser acessados de forma independente de sua implementação.

No CORBA, clientes solicitam a execução de serviços e servidores executam tais serviços, tudo em termo de tarefas denominadas operações que são executadas por entidades denominadas de objetos.

Aplicações interagem umas com as outras sem saber onde as outras aplicações estão na rede ou como elas executam suas tarefas. Usando CORBA, é possível encapsular aplicações e suas respectivas operações como conjuntos de objetos distribuídos de tal forma que se possa *plugar* e *desplugar* capacidades de clientes e servidores a medida que estes necessitem ser adicionados ou substituídos em um sistema distribuído.

Logo, CORBA trata heterogeneidade a nível de plataforma, inclusive heterogeneidade a nível de banco de dados, oferecendo uma infra-estrutura para implementar SBDH, deixando, no entanto, a nível da programação do SBDH o tratamento da interoperabilidade semântica.

É justamente nesta linha de pesquisa que estamos trabalhando no projeto HEROS, SBDH com acoplamento forte, em desenvolvimento na PUC-Rio, que usa um modelo integrador orientado a objetos.

No momento, estamos identificando os objetos que compõem a arquitetura do HEROS tanto a nível de integração de esquemas quanto da execução de transação global, a serem especificados no CORBA. Este trabalho está nos exigindo a restruturação de alguns módulos tanto na gerência de transação quanto na integração de esquemas. O próximo passo será a escrita de tais especificações em IDL, a definição da política de implementação em algum ORB, possivelmente o ORBeline [POST94] que existe no Laboratório de Informática da PUC-Rio e, finalmente, a execução da respectiva implementação.

A abordagem do projeto HEROS é bem semelhante a do projeto MIND descrito na *seção 5.1*. As principais diferenças residem em:

<u>Quanto ao SBDH</u>

- modelo de dados

  ⇒ HEROS utiliza um modelo de objetos próprio;

  ⇒ MIND utiliza o modelo de objetos e linguagens baseados no padrão do ODMG-93.

- integração de esquemas

  ⇒ HEROS utiliza um conjunto de classes que representam o metadado, estando estas classes organizadas em uma hierarquia de especialização em vários níveis, onde destacam-se: *componente*, *tipo de ambiente*, *produto comercial*; com o intuito de facilitar a incorporação de novos componentes;

  ⇒ MIND revela a criação apenas de uma classe *Database Object* para a integração de novos componentes. As particularidades de cada SGBD componente são deixadas para o desenvolvimento das implementações, uma para cada SGBD a ser integrado à federação.

- mapeamento automático de esquemas

  ⇒ HEROS pretende utilizar mecanismo de regras;

  ⇒ MIND pretende utilizar mecanismos de definição de visão;

- gerência de transação

  ⇒ Além de MIND e HEROS utilizarem modelos de transação diferentes, é importante ressaltar que MIND já utiliza o agente global da gerência de transação distribuído, isto é, existente em vários nós, enquanto que HEROS tem esta abordagem como plano futuro, trabalhando atualmente com a existência de um único gerente global, existente em apenas um nó.

<u>Quanto ao uso do CORBA</u>

Existem basicamente duas diferenças entre as duas abordagens:

- a primeira delas é de que enquanto MIND implementa a gerência de transações como duas classes especificadas no CORBA (GDA e LDA), a abordagem do HEROS segue o exemplo do próprio OMA, e decompõe a gerência de transações em uma quantidade maior de classes, diminuindo a granularidade da representação, de forma a propiciar uma melhor legibilidade e extensibilidade, tanto para criar novas subclasses como para alterar as implementações sem impactar nas aplicações, considerando-se estas como as demais classes do própria gerência de transação;

- a segunda diferença é de que o controle da concorrência do HEROS pretende ser uma extensão do serviço de controle de concorrência da OMA, aproveitando-se de sua implementação, conhecimento e portabilidade já que faz parte do padrão, enquanto que MIND parece ter desenvolvido um módulo próprio para tal fim.

## 7. Referências Bibliográficas

---

MOWB95 Mowbray, T.J. & Zahavi, R.. **The Essential CORBA: Systems Integration Using Distributed Objects**. John Wiley &Sons, Inc, 1995, 316p.

NICO93 Nicol, J.R.; Wilkes, C.T. & Manola, F.A.. Object Orientation in Heterogeneous Distributed Computing Systems. **IEEE Computer**. Vol. 26, No. 6, Jun. 1993.

SOLE95 Soley, R.M. & Kent, W.. The OMG Object Model. In: **Modern Database Systems : The Object Model, Interoperability, and Beyond**. W. Kim, ed. Addison-Wesley Publishing Company, pp. 18-41, 1995.

HELA95 Helal, A. & Elmasri, R.. Standard Developments for Database and System Interoperability. In: **Simpósio Brasileiro de Banco de Dados**, X, 1995, Recife, Pernambuco. Recife, 1995.

MART94 Martin, J.. **Princípios de Análise e Projeto Baseados em Objetos**. Editora Campus. 1994. 486 p.

VINO93 Vinoski, S.. Distributed Object Computing with CORBA. **C++ Report Magazine**, jul.-aug. 1993.

OMG93 Object Management Group. **Object Management Architecture Guide**. John Wiley & Sons, Inc. 1993.

OMG95 Object Management Group. **Common Facilities Architecture**. OMG Document 95.1.2, Jan. 1995.

OMG94b Object Management Group. **Object Services Architecture**. OMG Document 94.11.12, Nov. 1994.

OMG94a Object Management Group. **Common Object Services Specifications**. John Wiley & Sons, Inc. 1994.

OMG91 Object Management Group. **The Common Object Request Broker: Architecture and Specification**, OMG TC Document Number 91.12.1, Revision 1.1, Dec. 6, 1991.

ORFA96 Orfali, R; Harkey, D. & Edwards, J.. **The Essential Distributed Objects - Survival Guide**. John Wiley & Sons, Inc. 1996. 603 p.

ROY95 Roy, M. & Ewald, A.. Distributed Object Interoperability. **Object Magazine**. SIGS Publication. Mar.-Apr. 1995.

OMG92 Object Management Group. **The Common Object Request Broker: Architecture and Specification**. John Wiley & Sons, Inc. 1992.

CATT94 Cattell, R.G.G.. The **Object Databases Standard: ODMG-93**. Morgan Kaufmann Publishers. 1994. 176 p.

SHET90 Sheth, A.P. & Larson, J.A.. Federated Database Systems for Managing Distributed. **Heterogeneous and Autonomous Databases**. ACM Computing Surveys. Vol. 22, No. 3, Sept. 1990.

UCHO94a Uchôa, E.M.A.. **HEROS - Um Sistema de Bancos de Dados Heterogêneos: Integrando Esquemas.** Rio de Janeiro : Pontifícia Universidade Católica do Rio de Janeiro, 1994. 203 p. Dissertação de Mestrado em Informática: Ciência da Computação. Departamento de Informática da PUC-Rio.

THOM90 Thomas, G.; Thompson, G.R.; Chung, C.; Barkmeyer, E.; Carter, F.; Templeton, M.; Fox, S. & Hartman, B.. Heterogeneous Distributed Database Systems for Production Use. In: **ACM Computing Surveys**, Vol. 22, No. 3, pp. 237-266, Sept. 1990.

PACI93 Pacitti, E.; Silva, S.D.; Duarte, C.H.C. & Melo, R.N.. HEROS: Um Sistema de Bancos de Dados Heterogêneos Orientado a Objetos. In: **Simpósio Brasileiro de Banco de Dados**, VIII, 1993. Campina Grande, Paraíba. Anais... Campina Grande, 1993.

UCHO94b Uchôa, E.M.A.; Silva, S.D. & Melo, R.N.. Modelo de Dados do HEROS - Sistema de Bancos de Dados Heterogêneos Orientado a Objetos. In: **Simpósio Brasileiro de Banco de Dados**, IX, 1994, São Carlos, São Paulo. Anais... São Carlos, 1994.

CHRY92 Chrysanthis, P.K. & Ramamritham, K.. ACTA: The Saga Continues. In: **Database Transaction Models for Advanced Applications**, editado por A. K. Elmagarmid, Morgan Kaufmann Pub., USA, 1992.

MEHR92b Mehrotra, S., Rastogi, R., Korth, H.F. & Silberschatz, A.. **A Transaction Model for Multidatabase Systems**, Technical Report TR-92-14, Department of Computer Sciences, University of Texas at Austin, Austin, Texas, USA, Mar. 1992.

SILV94 Silva, S.D.. **Sistemas de Bancos de Dados Heterogêneos: Modelo de Execução da Gerência de Transações**, Rio de Janeiro, Pontifícia Universidade Católica do Rio de Janeiro, 1994. 150 p. Tese de Doutorado em Informática, Ciência da Computação, Departamento de Informática da PUC-Rio.

BERN87 Bernstein, P., Hadizlacos, V. & Goodman, N.. **Concurrency Control and Recovery in Database Systems**, Addison-Wesley, USA, 1987.

VEIJ92 Veijalainen, J., & Wolski, A.. Prepare and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Database Systems. In: **Proceedings of the 8th International Conference on Data Engineering**, 1992.

QUAC94 Quack, K.. Corba-Implementierung macht Einzelanwendungen zum System. **Computerwoche**, Nov. 1994.

KOSC96 Koschel, A.; Kramer, R.; Nikolai, R.; Hagg, W.; Wiesel, J. & Jacobs, H.. **A Federation Architecture for an Environmental Information System incorporating GIS, the World Wide Web, and CORBA**, 1996.

HAST94 Hastings, E.E. & Kumar, D.H.. Providing Customers Information Using the WEB and CORBA. In: **2nd WWW-Conference**, 1994. http:/www.ncsa.uiuc.edu/SDG/IT94/ Proceedings/DDay/hastings/hastings.html

MURP94a Murphy, J. & Grimson, J.. **The Jupiter System: An Environment for Multidatabase Interoperability**. Technical Report, Dublin City University and Trinity College Dublin, Jan., 1994.

DOGA95d Dogac, A.; Kilic, E.; Ozhan, G.; Dengi, C.; Kesim, N. & Koksal, P.. Experiences in Using CORBA for a Multidatabase Implementation. In: **Proceedings of the 6th International Workshop and Conference on Database and Expert Systems Applications DEXA'95**, London, UK, Sept. 1995.

DOGA95c Dogac, A.; Dengi, C.; Kilic, E.; Ozhan, G.; Ozcan, F.; Nural, S.; Evrendilek, C.; Halici, U.; Arpinar, B.; Koksal, P.; Kesim, N. & Mancuhan, S.. **METU Interoperable Database System**. Technical Report 6-1, Software Research and Development Center, Scientific and Technical Research Council of Turkey, Middle East Technical University, Ankara Turkiye, Jun. 1995

DOGA96c Dogac, A.; Halici, U.; Kilic, E.; Ozhan, G.; Ozcan, F.; Nural, S.; Dengi, C.; Mancuhan, S.; Arpinar, B.; Koksal, P. & Evrendilek, C.. METU Interoperable Database System. Demo Description, In: **Proceedings of ACM SIGMOD International Conference on Management of Data**, Montreal, Jun. 1996.

DOGA96b Dogac, A.; Ozhan, G.; Kilic, E.; Ozcan, F.; Nural, S.; Dengi, C.; Halici, U.; Arpinar, B.; Koksal, P.; Mancuhan, S. & Evrendilek, C.. Making Oracle7, Sybase and Adabas D Interoperable through CORBA: MIND Project. In: **Proceedings of European Oracle User Group Conference**, Amsterdam, Apr. 1996.

DOGA96a Dogac, A.; Dengi, C.; Kilic, E.; Ozhan, G.; Ozcan, F.; Nural, S.; Evrendilek, C.; Halici, U.; Arpinar, B.; Koksal, P.; Kesim, N. & Mancuhan, S.. A Multidatabase System Implementation on CORBA. **6th International Workshop on Research Issues in Data Engineering (RIDE-NDS'96)**, New Orleans, pp. 2-11, Feb. 1996.

DOGA94a Dogac, A.; Ozkan, C.; Arpinar, B.; Okay, T. & Evrendilek, C.. METU Object-Oriented DBMS. In **Object-Oriented Database Systems**, edited by Dogac, A., Ozsu, T., Biliris, A., Sellis, T., Springer-Verlag, 1994.

DOGA94b Dogac, A. et al. METU Object-Oriented Database System. Demo Description, In: **Proceedings ACM SIGMOD International Conference on Management of Data**, Minneapolis, May, 1994.

DOGA95a Dogac, A.; Altinel, M.; Ozkan, C. & Durusoy, I.. Implementation Aspects of an Object-Oriented DBMS, In: **ACM SIGMOD Record.** Vol. 24, No. 1, pp. 9-14, Mar. 1995.

DOGA95b Dogac, A.; Altinel, A.; Ozkan, C.; Durusoy, I. & Altintas, I.. METU Object-Oriented DBMS Kernel. In: **Proceedings of International Conference on Database and Expert Systems Applications**, London, Sept. 1995 (Lecture Notes in Computer Science, Springer-Verlag, 1995).

HWAN94 Hwang, S.-Y.; Srivastava, J. & Li, J.. Transaction Recovery in Federated Autonomous Databases. In: **Distributed and Parallel Databases**, Vol. 2, No. 2, pp. 151-182, Apr. 1994.

HALI95 Halici, U., Arpinar, B. & Dogac, A.. **Serializability of Nested Transactions in Multidatabases**. Technical Report 95.10.1, Software R&D Center, METU, Oct. 1995.

OZCA96 Ozcan, F.. **Dynamic Query Optimization on a Distributed Object Management Platform**. MS Thesis, Computer Engineering Department. METU, Jan. 1996.

MURP94b Murphy, J. & Grimson, J.. The Jupiter System: a Prototype for Multidatabase Interoperability. In: **Proceedings of 12th British National Conference on Databases**, Guildford, England, Jul. 1994.

MURP95 Murphy, J. & Grimson, J.. Multidatabase Interoperability in the Jupiter System. **Information and Software Technology**. Vol. 37, No. 9. 1995.

MURP94c Murphy, J. & Grimson, J.. Jupiter/MDD: a Formal Model of a CORBA-based Multidatabase Dictionary. **Conference on Object-Oriented Information Systems**. 1994.

MURP93b Murphy, J. & Grimson, J.. **JIL/MDD: The Jupiter Interoperator Dictionary for Heterogeneous Database Systems**. Technical Report 97. Dublin City University. Aug. 1993.

MURP92 Murphy, J. & Grimson, J.. An Object Model in Z. **Database Technology**. Vol. 4, No. 4. pp. 297-304. 1992.

MURP93a Murphy, J. & Grimson, J.. **The Denotational Semantics of a Database Language**. Technical Report CA-1393. Dublin City University. Jan. 1993.

MURP94d Murphy, J. & Grimson, J.. The Jupiter System: An Environment for Multidatabase Interoperability. **Journal of Information Science**. Elsevier Science Publishers. Vol. 20, No. 2. 1994.

ALON88 Alonso, R. & Barbara, D.. Negotiating Data Access in Federated Database Systems. **Technical Report CS-TR-160-88**. Princeton University, Department of Computer Science, Jun. 1988.

POST94 PostModern Computing. **ORBeline User's Guide.** PostModern Computing Technologies, Inc., USA, 1994.