Pós-Graduação em Ciência da Computação

"Uma Abordagem para Gerenciamento de Consistência em um Ambiente de Banco de Dados Heterogêneos"

Por

Renata Costa Guedes Pereira

Dissertação de Mestrado

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE, Fevereiro/1999

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RENATA COSTA GUEDES PEREIRA

"Uma Abordagem para Gerenciamento de Consistência em um Ambiente de Banco de Dados Heterogêneos"

ESTE TRABALHO FOI APRESENTADO À PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DO CENTRO DE INFORMÁTICA DA UNIVERSIDADE FEDERAL DE PERNAMBUCO COMO REQUISITO PARCIAL PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO.

ORIENTADORA: Profa. Dra. Ana Carolina Salgado

RECIFE, 26 de Fevereiro de 1999

Dissertação de Mestrado apresentada por **Renata Costa Guedes Pereira** a Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e da Natureza da Universidade Federal de Pernambuco, sob o título **"Uma Abordagem para Gerenciamento de Consistência em um Ambiente de Banco de Dados Heterogêneos"**, orientada pela Profa. Ana Carolina Salgado e aprovada pela Banca Examinadora formada pelos professores:

Prof. Fernando da Fonseca de Souza
Departamento de Informática / UFPE

Profa. Marta L. Queirós Mattoso
COPPE / UFRJ

Prof. Fernando Ferreira Rezende
CPT – FT3/EK - Alemanha

Visto e permitida a impressão.
Recife, 26 de fevereiro de 1999.

Prof. TERESA BERNARDA LUDERMIR
Coordenadora da Pós-Graduação em Ciência da Computação do Centro de Ciências Exatas e da Natureza da Universidade Federal de Pernambuco.

Pereira, Renata Costa Guedes
Uma abordagem para gerenciamento de consistência em um ambiente de banco de dados heterogêneos / Renata Costa Guedes Pereira. – Recife : O Autor, 1999.
xviii, 160 folhas : il., fig., tab.

Dissertação (mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da computação, 1999.

Inclui bibliografia.

1. Banco de dados – Banco de dados heterogêneos. 2. Consistência – Técnicas de integração. I. Título.

005.74 CDD (22.ed.) MEI2008-003

Á memória da minha avó Diva e aos meus pais ,
Marcos e Socorro, com todo o meu amor, admiração e respeito.

Agradecimentos

A todos que contribuíram direta e indiretamente para que esse trabalho fosse realizado o meu muito obrigada. Ao CNPq por custear o primeiro ano do meu trabalho, os meus sinceros agradecimentos. Agradeço sobretudo à Peter Schneider e Dr. Fernando Rezende que deram todo apoio financeiro para a realização desse trabalho durante todo o segundo ano.

A todos que fazem parte do departamento FT3-EK, o meu muito obrigada. Em especial, agradeço a Ulrich Schaefer, Alex, Christine, Haike e Grekor pela amizade e disponibilidade sempre que foi preciso.

Agradeco ao grupo MENTAS do qual participei quando da elaboração desta dissertação.

Agradeço mais que em especial, ao amigo e chefe Dr. Fernando, por sua dedicação, amizade, e apoio irrestrito durante todo o meu trabalho. Com certeza esse trabalho só chegou até aqui devido a toda ajuda e apoio prestado por ele.

Uli Hermsen, a você o meu muito obrigada, por ter se tornando um amigo tão especial. Obrigada por todo o apoio e amizade prestados durante o ano. Obrigada pelas aulas de alemão e por todo o apoio.

O meu muito obrigada as meninas do Apê: minhas duas cunhadas, Ana Cláudia e Daniela, Márcia e Micheline, que me proporcionaram um ano maravilhoso do qual nunca esquecerei.

Obrigada, Márcia, por ter-se mostrado uma amiga tão sincera. Obrigada por seus emails diários, que aliviavam um pouco a distância de todos do Brasil. Por ler toda a minha tese e ter dado a sua contribuição.

Obrigada aos meus amigos Inês, Harley, Karina, Michele, Helena, Débora, Stênio, Valério e Viviane por terem me dado o apoio emocional tão necessário quando eu estava a milhas de quilometros de vocês.

A Tia Tuta e Mila todo o meu carinho. Obrigada por sempre querer que eu estivesse ai pertinho.

Aos meus pais e irmãos, todo o meu amor. Obrigada, Frederico e Guilherme, dos quais sempre procurei ser uma imagem. Obrigada, Papai e Mamãe, por todo o amor passado por vocês. Obrigada por serem pessoas tão corretas e maravilhosas.

0 Índice

Lista de Tabelas

Lista de Figuras

Resumo

Atualmente, é muito comum nas empresas a distribuição dos dados ao longo dos departamentos e linhas funcionais. Dessa forma, recursos e dados são fragmentados contribuindo para o surgimento das chamadas “ilhas de informações”. Os dados são organizados e gerenciados por diferentes Sistemas Gerenciadores de Bancos de Dados (SGBDs) de diferentes fornecedores e diferentes sistemas operacionais os quais utilizam diferentes protocolos de rede. Em essência, os dados de uma empresa constituem-se de servidores de bancos de dados de vários fornecedores, *legacy systems* e fontes de dados relacionais e não relacionais. Infelizmente, estas fontes de dados não têm habilidade para comparar e relacionar dados entre si.

Nesta dissertação apresentamos uma metodologia para a resolução de heterogeneidade semântica de fontes de dados heterogêneas. A resolução de heterogeneidade semântica visa determinar precisamente os possíveis relacionamentos entre objetos que modelam informações similares em diferentes bancos de dados componentes. Além disto, uma outra meta é detectar os conflitos nas estruturas representacionais dos objetos que causam problemas durante a integração desses componentes. Por fim, deve ser providenciado o tratamento correto no relacionamento entre esses objetos.

Ao contrário de outras propostas para integrar bancos de dados heterogêneos, onde existe a presença de esquemas globais totais ou esquemas globais parciais (sistemas federados), utilizamos o conceito de navegação entre os bancos de dados integrados. Este método trata conflitos de dados apenas quando uma consulta envolve bancos de dados distintos. Ou seja, não existe um tratamento prévio das diferenças para a montagem de esquemas globais. Assim, os conflitos são gerenciados somente quando necessário.

Através do conceito de navegação e utilizando a tecnologia de *middleware* de banco de dados, apresentamos neste trabalho toda a problemática do nosso ambiente e as nossas soluções. Dentre estas podemos destacar: a utilização de tabelas intermediárias para resolver problemas de heterogeneidade de esquemas de “um-para-muitos atributos”, ou seja, um atributo em um banco de dados corresponde a vários em outro banco de dados. Em segundo lugar, empregamos um método avançado de busca por proximidades para recuperar de um banco de dados informações semelhantes às desejadas pelo usuário, quando o valor exato de um dado informado não pode ser encontrado. Por último, através de um mecanismo especial de checagem de consistência, cada usuário pode definir individualmente os seus desejos de testes de consistência a serem realizados durante a navegação entre bancos de dados. Através desta tecnologia, provemos ao usuário uma maior segurança quando o mapeamento entre valores de dados correspondentes não pode ser feito automaticamente pelo sistema.

Palavras Chave: Banco de Dados, Banco de Dados Heterogêneos, Gerenciamento de Consistência.

Abstract

A common problem within most corporations nowadays is the distribution of data along departmental and functional lines, contributing to the emergence of the so-called “islands of information”. The data are organized and managed by a mix of different DBMSs from different software vendors and different operating systems that use different network protocols. Usually, the total corporate data resource is composed of multi-vendor database servers, legacy systems, and relational and non-relational data sources. Unfortunately, these data sources have no ability to compare and relate data amongst themselves.

In this thesis, we present an approach to solve the semantic heterogeneity among heterogeneous data sources. The task of resolving semantic heterogeneity aims at determining precisely the relationships between objects that are used to model similar information in different components. Moreover, another major goal is the detection of possible conflicts in structural representation of objects, which pose problems during the integration of such components. At last, the correct treatment and handling of these objects should be considered as well.

In contrast to other proposals in the field of heterogeneous databases’ integration, where a single global schema or partial global schemas (federated systems) are usually employed, we exploit the concept of navigation through the integrated databases. This method treats data conflicts only when a statement relates data from distinct databases at once. This means, we do not treat the global schema’s construction beforehand, but on the contrary, we detect and cope with the conflicts during database navigation on the fly.

We present in this work the problems we were faced with during the design and implementation of our integrated databases’ environment, and furthermore, we discuss our solutions to these problems. Among these, we can emphasize: the use of intermediary tables to solve problems of schemas’ integration of the type “one-to-many attributes”, i.e., an attribute in a database corresponds to many others in another database. In addition, we use an advanced methodology for proximity search in order to retrieve for the user similar information from the databases, in case the desired data value cannot be found for any reason. At last, by means of a special mechanism for consistency checking, we allow each user to individually define the level of consistency that must be enforced during navigation through the integrated databases. On the basis of this technology, we provide the users with a kind of expert system behavior when the mapping between correspondent data values cannot be made automatically and without inconsistencies.

Keywords: Databases, Heterogeneous Databases, Consistency Management

1 Introdução

1.1 Motivação

Nos anos 90 as organizações estão passando por períodos de fortes transformações. Pressões de várias naturezas forçam-nas a se adaptarem, a reagirem. O processo de globalização dos mercados traz à tona um grau de interdependência entre as nações e um fluxo de produtos, serviços e idéias nunca antes vivenciado. Esse fluxo implica na intensificação do impacto da evolução tecnológica que se alastra pelo globo, tornando velho o que era novo e obsoleto o que há pouco era moderno. Fatores como custo, produtividade, qualidade e inovação tornaram-se ainda mais críticos para a conquista e principalmente para a manutenção da vantagem competitiva.

Esse processo alavancado pela globalização dos mercados e pela intensificação das mudanças tecnológicas, o aumento das organizações e o aumento de competitividade existente levam as empresas a uma reestruturação. A gerência inicialmente centralizada dos dados evoluiu para um modelo mais flexível. As organizações fragmentaram-se em unidades menores, descentralizando a gerência do negócio. Com isto, os dados e processamento também foram descentralizados. Cada unidade passou a possuir e controlar suas próprias fontes de dados [Pire97]. Forças tecnológicas capacitantes, como a computação em grupo e a interconexão em rede, permitem que empresas tenham alta performance e tornem-se integradas e ampliadas.

Nos dias de hoje, é muito comum nas empresas a distribuição dos dados ao longo dos departamentos e linhas funcionais. Dessa forma, recursos e dados são fragmentados contribuindo para o surgimento das chamadas “ilhas de informações”. Os dados são organizados e gerenciados por diferentes **Sistemas Gerenciadores de Bancos de Dados** (SGBDs) de diferentes fornecedores e diferentes sistemas operacionais os quais utilizam diferentes protocolos de rede. Em essência, os dados de uma empresa constituem-se de servidores de banco de dados de vários fornecedores, *legacy systems* e fontes de dados[1] relacionais e não relacionais. Infelizmente, estas fontes de dados não têm habilidade para relacionarem-se entre si [RH98, IBM95].

1. Nesta dissertação o termo fonte de dados sugere o mesmo significado que banco de dados (BDs).

O problema de fontes de dados heterogêneas é tão comum no ambiente de corporacões que, durante o *Workshop on Future Database Systems Research* patrocinado pela *National Science Foundation* (NSF), foi identificada a necessidade da criação de um ambiente que controle o compartilhamento e a troca de informações entre bancos de dados autônomos e heterogêneos. Esta área foi caracterizada como de grande importância nas pesquisas futuras de bancos de dados [Ham94, FHM92, FHMS90].

É nesse contexto que está inserido o problema que o **MENTAS** - *MotorEntwicklungsAssistent* - se propõe a resolver. O MENTAS é um projeto de inovação da *DaimlerChrysler AG Research and Technology*, que está sendo desenvolvido para engenheiros mecânicos do desenvolvimento de motores da *Mercedes-Benz*. Hoje, o ambiente de desenvolvimento de motores na *Mercedes-Benz* é caracterizado pelo isolamento, pelo uso de *softwares* e bancos de dados de vários fornecedores. Essencialmente, cada departamento envolvido no desenvolvimento de um motor constitui uma ilha de informação com suas próprias ferramentas e fontes de dados.

O principal problema encontrado pelos engenheiros neste ambiente completamente heterogêneo diz respeito à habilidade de recuperar as informações das várias fontes de dados para realizar a comparação e correlacioná-las. Os bancos de dados são encapsulados pelas aplicações, de forma que os engenheiros não têm acesso direto aos dados armazenados. Portanto, eles devem utilizar diferentes *interfaces* providenciadas pelas aplicações para recuperar informações de cada banco de dados. Outra limitação é que tais *interfaces* não possibilitam a criação de consultas *ad hoc*, sendo limitadas a um número de consultas pré-definidas para acessar os bancos de dados locais. Como o projeto e desenvolvimento de um novo motor é um processo extremamente criativo, os engenheiros sempre vêem-se limitados na sua criatividade porque as *interfaces* não providenciam um método adequado para recuperar informações nem de uma única fonte de dados, e muito menos quando estão envolvidos duas ou mais fontes de dados heterogêneas. Ou seja, não há o suporte à realização de comparações e correlações entre os dados.

A proposta do MENTAS é realizar a interconexão das fontes de dados e das ferramentas, montando um ambiente de desenvolvimento orientado à engenharia para a rápida concepção e análise comparativa dos motores. Para atingir este objetivo, deve ser providenciado um acesso automático e integrado aos bancos de dados heterogêneos e às várias ferramentas de simulação.

A solução para a heterogeneidade nas fontes de dados utilizadas pelo MENTAS é baseada na tecnologia *middleware* de banco de dados. Acima do sistema *middleware*, foi projetada e implementada uma interface de acesso aos bancos de dados, através da qual o usuário tem a possibilidade de formular consultas SQL de modo homogêneo como se tratasse de um banco de dados homogêneo e não de uma federação de bancos de dados heterogêneos. As consultas feitas pelo usuário para manipular os bancos de dados seguem o padrão ISO SQL2 (*Structured Query*

Language [Mel90, DD97]). Através de uma GUI (*Graphic User Interface*), os usuários são guiados no processo de criação de suas consultas as quais podem acessar desde um simples banco de dados, como também integrar duas ou todas as demais fontes de dados presentes no MENTAS, tornando um trabalho complexo numa tarefa trivial para o usuário. A integração dos dados no MENTAS é feita através do conceito de **navegação** entre os bancos de dados. Apesar dos dados estarem espalhados ao longo de bancos de dados remotos, autônomos e heterogêneos, o que o usuário percebe é um esquema global. Dessa forma os engenheiros podem navegar por entre os bancos de dados comparando dados e relacionando informações através de uma única e simples *interface* gráfica. Isso significa que a informação correta torna-se disponível aos usuários de maneira mais confortável e, ainda mais importante, muito mais rápido que no ambiente de desenvolvimento atual.

Para que a interface do usuário seja simples e homogênea, faz-se necessário que um conjunto de módulos a auxiliem nos diversos problemas encontrados tanto a nível de um único banco de dados, e principalmente quando trata-se dos bancos de dados integrados. Nesta dissertação, estamos particularmente interessados no trabalho executado por um dos alicerces dessa arquitetura, o **Monitor de Consistência** do MENTAS. Além de tratar a entrada de dados do sistema no momento da montagem das consultas, esse módulo é responsável por possibilitar a navegação entre os bancos de dados integrados.

1.2 Objetivos da Dissertação

Consistência é uma das propriedades mais esperadas em um sistema computacional, e também uma das mais difíceis de ser garantida. A própria noção de consistência é definida de diferentes formas, de acordo com a comunidade científica interessada [Cho96]. Nesta dissertação, a palavra consistência está diretamente relacionada à resolução de heterogeneidade semântica das fontes de dados integradas. A resolução de heterogeneidade semântica visa *determinar precisamente os possíveis relacionamentos entre objetos que modelam informações similares em diferentes componentes (bancos de dados)* e *detectar os possíveis conflitos nas estruturas representacionais dos objetos que causam problemas durante a integração desses componentes*. Além disso, *deve ser providenciado o tratamento correto entre esses objetos.*

O objetivo principal desta dissertação é apresentar a metodologia empregada para resolução da heterogeneidade das fontes de dados dentro do ambiente apresentado pelo MENTAS. Ao contrário de outras propostas para integrar bancos de dados heterogêneos, onde existe a presença de esquemas globais totais ou esquemas globais parciais (sistemas federados), o MENTAS utiliza o conceito de navegação. Este método trata conflitos de dados apenas quando uma consulta envolve dois bancos de dados distintos. Ou seja, não existe um tratamento prévio das diferenças

para a montagem de esquemas globais (parciais ou totais). Através do conceito de navegação e utilizando a tecnologia *middleware* de banco de dados, apresentaremos toda a problemática do nosso ambiente e a nossa solução para o problema.

Além disto, esta dissertação apresenta um panorama geral da pesquisa na área de integração de bancos de dados heterogêneos, sobre o qual está fundamentado este trabalho. Após uma breve apresentação do universo de banco de dados distribuídos, são examinadas várias técnicas utilizadas para a resolução do problema das fontes de dados heterogêneas que compartilham informações. Este estudo serviu de base para a concepção do Monitor de Consistência atualmente empregado no MENTAS.

1.3 Estrutura da Dissertação

Os demais capítulos que constituem esta dissertação estão divididos da seguinte forma:

- Capítulo 2: **Sistemas de Bancos de Dados Distribuídos**

Este capítulo descreve alguns conceitos de bancos de dados distribuídos, identificando o posicionamento dos sistemas de bancos de dados heterogêneos como uma ramificação dos bancos de dados distribuídos.

- Capítulo 3: **Sistemas de Bancos de Dados Heterogêneos**

Este capítulo dedica-se a descrever toda a problemática existente na união de fontes de dados heterogêneas. Também, apresentamos as principais abordagens da literatura para a resolução deste problema.

- Capítulo 4: **MENTAS**

Neste capítulo introduzimos o projeto MENTAS, os problemas encontrados e os objetivos do projeto.

- Capítulo 5: **Integração dos Bancos de Dados no MENTAS**

Este capítulo apresenta a arquitetura utilizada pelo MENTAS para realizar o acesso aos bancos de dados. Aqui também é apresentada a funcionalidade da *interface* [Oli99] provida para o usuário final, focalizando os aspectos da navegação.

- Capítulo 6: **Monitor de Consistência**

Este capítulo apresenta o Monitor de Consistência utilizado pelo MENTAS, descrevendo suas características, seus objetivos básicos, as estratégias utilizadas e os resultados alcançados.

- Capítulo 7: **Conclusão**

Neste último capítulo, são exibidas conclusões sobre o trabalho desenvolvido, destacando as contribuições. Por fim, algumas sugestões de trabalhos futuros são indicadas.

2 Sistemas de Bancos de Dados Distribuídos

2.1 Introdução

No início da década de 60, os sistemas de bancos de dados foram propostos como uma solução para o problema do acesso compartilhado aos arquivos de dados heterogêneos criados por múltiplas aplicações em um ambiente centralizado [HDRK97]. Estes arquivos de dados eram difíceis de serem gerenciados. Eles freqüentemente continham duplicações, inconsistências, redundâncias, e vários tipos de heterogeneidades tanto a nível estrutural como a nível dos dados. Para resolver estes problemas, os arquivos autônomos foram substituídos por um **banco de dados** o qual opera embaixo do controle centralizado de um **Sistema de Gerenciamento de Banco de Dados** (SGBD).

Existem vários critérios utilizados para classificar os **Sistemas de Gerenciamento de Bancos de Dados** (SGBDs). Segundo [EN94], os principais critérios para a classificação são: *modelo de dados*, *o número de usuários* e o *número de nós* (Figura 2.1).

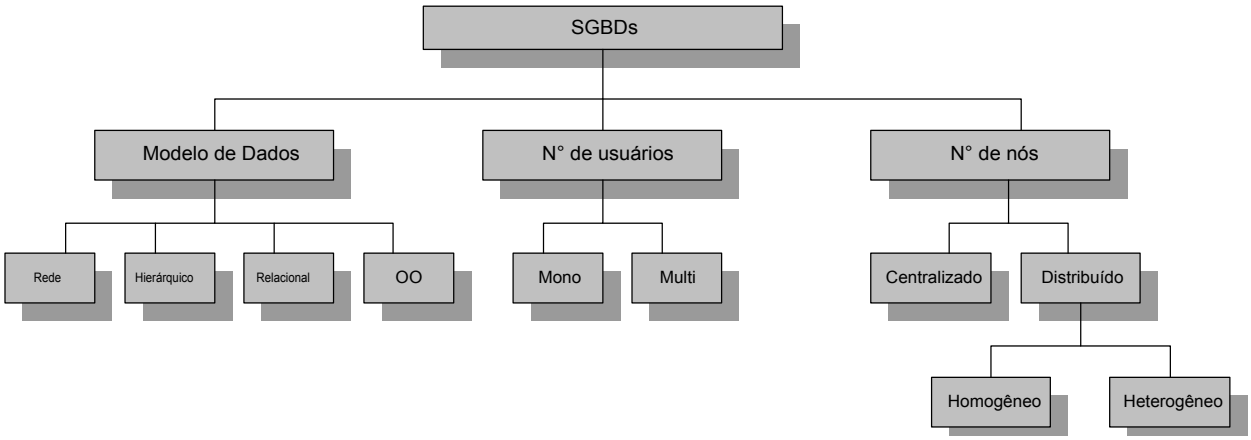
Descendo um nível da hierarquia, partimos para a classificação do *modelo de dados* no qual o sistema é baseado. Os modelos de dados utilizados com mais freqüência entre os SGBDs são o *modelo de rede*, *modelo hierárquico*, o *modelo relacional* e mais recentemente, o *modelo orientado a objetos*.

O segundo critério utilizado para classificar os SGBDs, *número de usuários* suportados pelo sistema, é aprofundado na hierarquia em *monousuário* e *multiusuário*. Sendo o próprio nome auto explicativo, um sistema monousuário possui apenas um único usuário por vez e é mais utilizado em computadores pessoais. Já um sistema multiusuário, onde estão incluídos a maioria dos SGBDs, suporta múltiplos usuários trabalhando concorrentemente.

O terceiro e último parâmetro utilizado na classificação dos SGBDs é o *número de nós* que o sistema pode estar distribuído. Muitos sistemas podem ser *centralizados*, significando que os dados estão armazenados em um único computador, ou nó. Um SBD centralizado pode suportar múltiplos usuários, mas o SGBD e os bancos de dados residem completamente em um único nó. Um SBD *distribuído* pode ter tanto o banco de dados como o *software* que o gerencia (SGBD) distribuídos

através de múltiplos nós, sendo conectados através de uma rede de computadores. Um nível abaixo dos sistemas distribuídos, encontram-se os sistemas *homogêneos* e os sistemas *heterogêneos.* Se o *software* utilizado para fazer o gerenciamento for o mesmo em todos os nós da rede e todos os clientes utilizarem o mesmo *software*, o sistema é chamado de homogêneo. Em qualquer outra situação, ele é dito heterogêneo.

Figura 2.1 Classificação dos SGBDs.



Os sistemas distribuídos foram amplamente difundidos na década atual [1]. O processo alavancado pela globalização dos mercados, pela intensificação das mudanças tecnológicas além do aumento de competitividade existente, enfatiza a necessidade do trabalho em grupo, porém de natureza distribuída e paralela. Como conseqüência, é requisitado das organizações a capacidade de compartilhar informações que foram implantadas em vários tipos de plataformas. No tocante aos Sistemas de Gerenciamento de Banco de Dados, a principal motivação é integrar os dados de uma organização, de forma a oferecer um acesso centralizado e controlado a esses dados.

2.2 Sistemas de Gerenciamento de Bancos de Dados Distribuídos

O termo Sistema de Gerenciamento de Bancos de Dados Distribuídos (SGBDD) designa sistemas com aspectos diversos. A característica principal de tais sistemas é o fato dos dados e do *software* estarem distribuídos através de vários nós conectados de alguma forma por uma rede de comunicação. De acordo com [OV91], um

1. Muito embora os sistemas distribuídos venham sendo amplamente estudados desde a década de 70, as pesquisas iniciais concentram-se em sistemas operacionais distribuídos [Bar97].

SGBDD é definido como o *software que permite o gerenciamento de um banco de dados distribuído e que, além disso, torna a distribuição transparente aos usuários*. Onde, **banco de dados distribuídos** é definido por [EN94] como sendo *a coleção de dados logicamente relacionados mas fisicamente distribuídos sobre os nós de uma rede de computadores.*

Alguns fatores contribuem para o desenvolvimento de SGBDDs. Entre os principais podemos destacar *a natureza distribuída de algumas aplicações de banco de dados*, o *aumento da confiabilidade e disponibilidade dos sistemas, o compartilhamento dos dados assegurando um certo controle local* e *aumento de performance*.

Natureza distribuída

Muitas aplicações de bancos de dados são naturalmente distribuídas através de diferentes localizações. É natural que os bancos de dados utilizados por tais aplicações sejam também distribuídos ao longo dos nós da rede. Essas aplicações podem ter usuários locais que enxergam apenas os dados locais, mas podem ter também usuários globais que possuem uma visão global dos dados espalhados entre os vários nós.

Aumento da confiabilidade e disponibilidade

Estas são duas das vantagens mais comuns citadas. Confiabilidade é em geral definida como a probabilidade do sistema estar funcionando em um momento particular, e disponibilidade a probabilidade do sistema ser continuamente disponível durante um intervalo de tempo. Com os dados de um SGBD distribuídos em diversas localidades, no caso de ocorrer uma falha em uma localidade, as demais continuam operando. Só os dados e o *software* que existem na localidade que falhou é que não podem ser acessados, aumentando assim tanto a confiabilidade quanto a disponibilidade. Em um sistema centralizado, se ocorrer uma falha, geralmente todo o sistema torna-se inacessível para todos os usuários.

Compartilhamento dos dados *versus* controle local

Como os dados estão distribuídos ao longo dos nós da rede, é possível que os usuários locais tenham os dados armazenados localmente no nó. Entretanto, usuários de outras localidades podem acessar esses dados remotamente através do *software* (SGBDD). Isto permite um compartilhamento das informações, assegurando um controle local.

Aumento de performance

A distribuição de um banco de dados em vários nós da rede faz com que haja um aumento de performance no que diz respeito à execução das transações em um banco de dados. Isso ocorre devido a possibilidade dos dados mais

freqüentemente acessados estarem armazenados próximos aos seus usuários, e ainda a possibilidade de ocorrer a execução paralela da transação. Se as transações acessarem informações de um único nó, o tempo de resposta será mais rápido devido a porção do banco de dados estar reduzida. No caso de acessar vários nós, a transação pode ser dividida e executar paralelamente.

A idéia da arquitetura cliente-servidor está sendo incorporada aos SGBDs de forma a reduzir a complexidade de tais sistemas. A princípio, o SGBD é dividido em dois módulos - cliente e servidor -, onde a máquina servidora possui o *software* que é acessado por múltiplos clientes. A divisão da funcionalidade do SGBD entre o cliente e o servidor até agora não foi estabelecida. O *software* de um sistema gerenciador de bancos de dados distribuídos pode ser dividido em três níveis [EN94]:

- **servidor**: *software* responsável pelo gerenciamento local dos dados no nó, semelhante a um SGBD centralizado.
- **cliente**: *software* responsável pela maioria das funções de distribuição. Ele acessa as informações distribuídas do catálogo do SGBD e processa todos os pedidos que requerem acesso a mais de um nó componente.
- ***software* de comunicação**: prover as primitivas de comunicação que são utilizadas pelo cliente para transmitir comandos e dados entre os vários nós quando necessário. Ele não constitui necessariamente uma parte integrante do SGBD, mas oferece serviços e primitivas de comunicação essenciais ao seu funcionamento.

2.3 Tipos de Bancos de Dados Distribuídos

Existem alguns critérios e fatores que diferenciam os sistemas de bancos de dados distribuídos. Segundo [EN94, SL90], podemos caracterizar tais sistemas de acordo com o *grau de heterogeneidade*, o *grau de distribuição* e o *grau de autonomia*.

2.3.1 Grau de Heterogeneidade

Se todos os servidores usam *softwares* idênticos e todos os clientes também usam *softwares* idênticos, o SGBDD é dito homogêneo; de qualquer outra maneira, é chamado heterogêneo.

A heterogeneidade pode ocorrer de várias formas em sistemas distribuídos, desde diferenças de *hardware* e diferenças em protocolos de rede, até variações em gerenciadores de dados. No caso da heterogeneidade entre os SGBDs (relacional, de rede, orientado a objetos, ou mesmo de fabricantes diversos), é necessário a tradução da consulta originada por um cliente para as linguagens que cada servidor entende. Os fatores mais importantes, a partir da perspectiva de banco de dados,

referem-se aos modelos de dados, linguagens de consulta, *interfaces*, protocolos de gerenciamento de transações e a semântica dos dados.

Como podemos notar, o problema da heterogeneidade em sistemas de bancos de dados interoperáveis é uma conseqüência da autonomia de projeto, devido as diversas escolhas disponíveis aos projetistas nos diferentes níveis do mesmo [HDRK97]. A heterogeneidade pode ser classificada em quatro categorias distintas: *heterogeneidade de sistemas, heterogeneidade sintática, heterogeneidade de esquema e heterogeneidade semântica.*

Muitas vezes, a heterogeneidade semântica e a heterogeneidade de esquemas são fundidas em um único conceito [KS91, KGCS95, SL90]. Isto acontece porque a diferença do que constitui semântica e o que constitui estrutura não é bem clara.

Heterogeneidade de Sistemas

Consiste das diferenças de mais baixo nível das arquiteturas das plataformas que os sistemas empregam, como configuração de *hardware*, sistemas operacionais e tipo de comunicação. Por exemplo, um sistema de banco de dados pode ser implementado em uma plataforma Unix, enquanto um segundo sistema pode ser implementado no sistema operacional VMS. Aqui também estão incluídas as heterogeneidades que são conseqüências de diferentes técnicas de sistemas utilizadas por diferentes SGBDs, como estratégias de processamento de consultas, mecanismos de controle de concorrência e gerenciamento de transação.

Pesquisas nesta área têm resolvido este problema, através de adaptadores para cada sistema componente[1], de forma que consigam "conversar" com o sistema de *software* que manipula todos os bancos de dados componentes (sistemas *multidatabase*) [HDRK97, SL90].

Heterogeneidade Sintática

Ocorre devido às diferenças sintáticas entre os modelos de dados (em outras palavras, diferenças nas facilidades disponíveis para representar e manipular os dados) que são providos pelos SGBDs componentes. Como exemplo de heterogeneidades desta categoria, temos:

- Diferenças nas primitivas das estruturas de dados providas pelos modelos de dados (relações *versus* classes, por exemplo).
- Diferenças nos mecanismos utilizados para expressar as restrições de integridade dos bancos de dados.

1. Nesta dissertação o termo sistema componente e sistema local são utilizados com o mesmo significado.

- Diferenças nas linguagens de acesso providas pelos SGBDs.

Este problema tem sido resolvido através da utilização de um modelo de dados comum a nível global. Cada componente do sistema é passado por um processo de tradução, o qual passa o modelo de dados a nível local para o modelo de dados comum a nível global [SL90].

Heterogeneidade de Esquemas

Este tipo de heterogeneidade ocorre quando dados equivalentes ou relacionados são apresentados em diferentes bancos de dados possuindo representações estruturais diferentes.

A resolução deste problema em si não representa um problema muito grande. Segundo [BLN86], dois bancos de dados que possuem esquemas diferentes podem ser integrados pela reestruturação e fusão dos esquemas componentes. O principal problema aqui é detectar quais são os conflitos estruturais nos conceitos dos dados que representam o mesmo conceito do mundo real. Em outras palavras, quais são as diferenças representacionais que existem entre os objetos relacionados nos bancos de dados. Portanto, é preciso nesta situação a habilidade de capturar a semântica dos dados independentemente das suas propriedades estruturais. Semântica dos dados neste contexto refere-se ao significado, interpretação e uso dos conceitos dos dados, não das propriedades de esquemas.

Heterogeneidade Semântica

Um problema central de interoperabilidade que deve ser levado em consideração no momento do suporte ao compartilhamento das informações entre uma coleção de banco de dados heterogêneos é a **heterogeneidade semântica**. Este termo é definido como as variações na maneira na qual o dado é especificado e estruturado em diferentes componentes. Heterogeneidade semântica é uma conseqüência natural da criação independente e a evolução de bancos de dados autônomos que são feitos sob medida para os requisitos dos sistemas dos quais eles fazem parte.

Segundo [BLN86, KS91, HM93] existem três maiores causas para a heterogeneidade semântica:

- *Diferentes perspectivas*: Este é um problema de modelagem, o qual está diretamente ligado à fase do projeto do esquema do banco de dados. Diferentes grupos de usuários ou projetistas adotam seus próprios pontos de vista para modelar a mesma informação.
- *Construtores equivalentes*: O rico conjunto de construtores nos modelos de dados permite um grande número de possibilidades na modelagem, os quais resultam em variações na estrutura conceitual do banco de dados. Tipica-

mente, em modelos conceituais, algumas combinações de construtores podem modelar o mesmo domínio do mundo real de forma equivalente.

- *Incompatibilidade na especificação do projeto*: Diferentes especificações de projeto resultam em diferentes esquemas.

2.3.2 Grau de Distribuição

Este fator trata do local do armazenamento dos dados. Os dados podem estar distribuídos sobre múltiplos bancos de dados. Ainda, os bancos de dados podem estar armazenados em um único sistema computacional ou em múltiplos sistemas, geograficamente distribuídos ou não, mas interconectados por um sistema de comunicação.

Um conceito importante referente à distribuição é a transparência de distribuição. Se o usuário vê um único esquema integrado sem qualquer informação sobre fragmentação, replicação [Bar97], ou distribuição, o SGBDD é dito ter um alto grau de transparência de distribuição (ou integração de esquema). Por outro lado, se o usuário percebe como os dados encontram-se distribuídos ao longo da rede, o SGBD distribuído não possui transparência de distribuição (nenhuma integração de esquema). Neste caso, o usuário deve especificar, explicitamente, a que fragmento dos dados, localizado em que nó da rede, ele se refere quando formular uma consulta ao banco de dados.

Este é um dos sérios problemas encontrados em sistemas distribuídos. No caso de SGBDs distribuídos que não oferecem transparência na distribuição, é dada ao usuário a responsabilidade de identificar de forma não ambígua os dados que deseja acessar. Esta tarefa é mais severa no caso dos sistemas federados, onde cada servidor (SGBD local) foi desenvolvido independentemente, e como resultado, devem existir nomes conflitantes entre os diversos servidores. Entretanto, no caso de um SGBD que possui um esquema integrado, este problema (conflitos de nomes) torna-se interno ao sistema, já que é apresentado ao usuário um único esquema sem ambigüidades.

2.3.3 Grau de Autonomia

O grau de autonomia refere-se à distribuição de controle e indica o grau em que cada SGBD pode operar independentemente. Ele envolve uma série de fatores, tais como a maneira com que os componentes do sistema trocam informações e a possibilidade de um determinado componente executar transações independentemente e modificá-las. A classificação apresentada por [SL90] subdivide a autonomia de três formas: com relação ao *projeto*, com relação à *comunicação* e por fim com relação à *associação*.

A autonomia de projeto refere-se a habilidade de um componente escolher suas próprias diretivas de projeto. Dentre as diretivas podemos citar: gerência dos dados, representação de tais dados (modelo de dados e linguagem de consulta), interpretação semântica desses dados[1], definição dos requisitos de integridade, definição das operações suportadas pelo sistema, como será o compartilhamento com os outros sistemas e finalmente como será a implementação.

A autonomia de comunicação diz respeito à capacidade de decisão de quando e como um banco de dados componente deve responder a um pedido de outro banco de dados componente. Além disso, também está relacionada à capacidade de resolução de quando comunicar com outro componente.

A autonomia de execução refere-se à habilidade de um componente executar operações locais (comandos ou transações submetidas diretamente pelo usuário local a um banco de dados componente) sem a interferência de operações externas. O componente também deve ser capaz de decidir a ordem que serão executadas as operações externas.

2.4 Taxonomia dos Bancos de Dados que Compartilham Informações

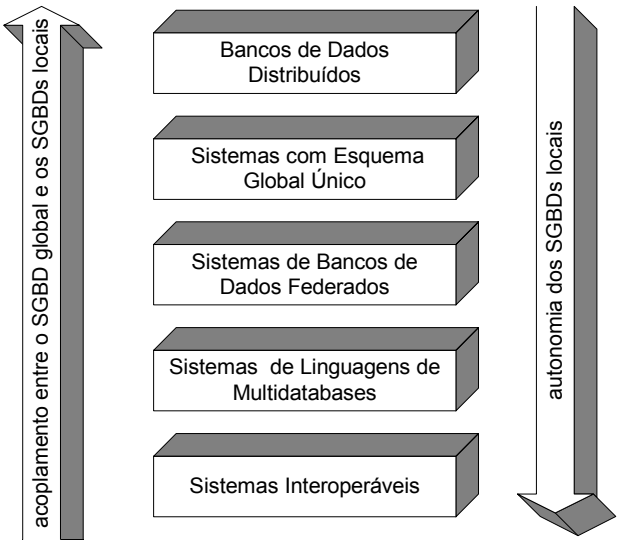
Existe um grande número de soluções para o problema do compartilhamento de dados globais em um ambiente distribuído e pode-se encontrar na literatura uma variedade de expressões para descrevê-las. Todos os termos empregados descrevem um sistema distribuído, que possui um componente global para fornecer acesso às informações que devem ser compartilhadas globalmente e múltiplos componentes locais que manipulam somente os dados locais. As diferenças residem na estrutura do componente global e como ele interage com os componentes locais [BHP92, Pire97]. De acordo com [BHP92], podemos caracterizar um sistema de várias maneiras, baseado na análise do grau de autonomia e do tipo de integração existente entre os componentes do sistema.

Ainda de acordo com [BHP92] dizer que um sistema é fortemente acoplado significa que as funções globais acessam a baixo nível as funções locais. Isto significa a possibilidade de uma boa sincronização entre os nós e um eficiente processamento global. Entretanto, isto também implica que as funções globais podem ter prioridades sobre as funções locais e portanto, que os SGBDs locais não têm o controle completo sobre os recursos locais. Em um sistema fracamente acoplado, as funções globais acessam as funções locais através da interface externa dos SGBDs. A sincronização global e a eficiência do processamento global não são tão altos como nos sistemas fortemente acoplados, mas em compensação o SGBD

1. Este fator contribui para o problema da heterogeneidade semântica.

local possui o controle completo sobre os dados e o processamento local, providenciando um alto grau de autonomia do nó. Na maioria dos sistemas fracamente acoplados existem poucas funções globais e a interface local para as informações globais utiliza as aplicações residentes acima da interface dos SGBDs locais. As definições a seguir começam dos sistemas mais fortemente acoplados até o mais fracamente acoplado (Figura 2.2). Na Tabela 2.1 podemos perceber as principais diferenças entre os sistemas de bancos de dados que compartilham informações

Figura 2.2 Taxonomia de Sistemas de Bancos de Dados que Compartilham Informações.



- ***Bancos de Dados Distribuídos***: Possui o acoplamento mais forte dos sistemas que compartilham dados. Funções globais e locais compartilham *interfaces* internas de baixo nível e são completamente integradas de forma que existem pouquíssimas diferenças entre elas. Esses sistemas são projetados de maneira *top-down*, com funções locais e globais implementadas simultaneamente. Os bancos de dados locais são tipicamente homogêneos (com relação ao modelo de dados). O sistema global tem o controle sobre todos os dados e processamentos locais. Este sistema, tipicamente, mantém um esquema global criado pela integração de esquemas de todos os SGBDs locais. Um esquema é a descrição estruturada de informações disponíveis nos bancos de dados [MIR93]. Usuários globais acessam o sistema submetendo consultas ao esquema global. Devido à forte integração, eles podem sincronizar o processamento global. Além disso, como as funções globais têm o controle sobre as funções locais, o processamento pode ser otimizado para os requisitos globais do sistema. Como resultado, os bancos de dados distribuídos têm uma perfor-

mance global excelente, mas custando aos bancos de dados locais a falta de autonomia.

- ***Sistemas com Esquema Global Único***. São sistemas com acoplamento mais fraco que os bancos de dados distribuídos porque as funções globais deste sistema acessam informações locais através de uma interface externa do banco de dados local. Entretanto esses sistemas ainda mantém um esquema global, o que leva os sistemas locais a terem que cooperar com o sistema global para manter o tal esquema. Ao contrário dos sistemas distribuídos, são tipicamente projetados de maneira *bottom-up* o que fornece a possibilidade de integrar bancos de dados pré-existentes sem alterá-los. Normalmente tais sistemas integram SGBDs locais heterogêneos. Ou seja, os sistemas com esquema global são compostos por um conjunto de SGBDs componentes, heterogêneos, cooperativos mas autônomos, integrados de modo a tornar as consultas e atualizações às informações globais totalmente transparentes quanto à localização e caminhos de acesso. A transparência é obtida pela tradução dos diferentes esquemas dos bancos de dados locais para um modelo de dados comum e integrado, compondo um esquema global único.
- ***Sistemas de Banco de Dados Federados***. São sistemas com acoplamento mais fraco que os de esquema global. Não existe um esquema global único, sendo que cada componente local mantém um esquema de importação e outro de exportação. O esquema de exportação é uma descrição das informações que o componente local compartilha com o sistema global. O esquema de importação é uma descrição da origem e representação dos dados dos nós remotos que podem ser acessados localmente. Cada esquema de importação é essencialmente um esquema global parcial. Portanto cada nó componente só precisa cooperar diretamente com os nós que ele acessa.
- ***Sistemas de Linguagens de Multidatabase***s. São sistemas de acoplamento mais fraco que os sistemas de banco de dados federados porque não existe um esquema global parcial. O sistema global suporta todas as transações globais através de ferramentas de linguagem de consulta que proporcionam a integração das informações dos SGBDs locais. Consultas podem ser feitas sobre qualquer dado dos esquemas locais de qualquer nó componente do sistema. Ferramentas de linguagem incluem um espaço global de nomes e funções especiais para mapear informações, de diferentes modelos e representações, para um modelo e representação significativos para o usuário. Assim como os dois últimos sistemas apresentados (com esquema global e os federados) este sistema pode integrar pré-existentes e heterogêneos SGBDs sem modificá-los, preservando a autonomia dos SGBDs locais.
- ***Sistemas interoperáveis***. São sistemas com o acoplamento mais fraco dos sistemas de compartilhamento de informações. As funções globais são limitadas a simples troca de dados e não há suporte para todas as facilidades de banco de dados. Protocolos padrões são definidos para a comunicação entre os nós. Como o sistema global não é orientado a banco de dados, os sistemas

locais podem ser constituídos por outros tipos de repositórios como sistemas especialistas ou sistemas baseados em conhecimento. Sistemas interoperáveis estão, ainda, em estágio de pesquisa.

Tabela 2.1. Diferenças entre Sistemas de Bancos de Dados que Compartilham Informações.

Classe	Nível da Interface Global para os SGBDs Locais	Tipos de Nós Locais	Método de Integração Global
Bancos de Dados Distribuídos	Funções internas ao SGBD	Homogêneos	Esquema global
Multidatabase com Esquema Global	Interface do usuário	Heterogêneos	Esquema global
Bancos de Dados Federados	Interface do usuário	Heterogêneos	Esquema global parcial
Sistemas de Linguagens de Multidatabase	Interface do usuário	Heterogêneos	Linguagem de acesso
Sistemas Interoperáveis	Aplicações no topo do SGBD	Qualquer fonte de dados	Nenhuma integração global

2.5 Considerações

Atualmente, os bancos de dados são criados dentro das organizações de acordo com as necessidades de cada departamento/divisão. Isso implica que uma mesma organização pode conter bancos de dados de vários fabricantes, que operam em diferentes plataformas e que foram modelados de acordo com as necessidades dos usuários locais a estas fontes. Algumas vezes, os sistemas de banco de dados dentro dos departamentos são distribuídos porém homogêneos. Outras vezes, até mesmo dentro do mesmo departamento encontra-se a presença de sistemas heterogêneos. Mas nos dias atuais torna-se imprescindível que as informações presentes nesses bancos de dados sejam compartilhadas por todo departamento/corporação, mantendo porém, a autonomia local das fontes de dados após a integração das mesmas.

A necessidade da integração dos sistemas heterogêneos traz à tona todos os problemas de heterogeneidade que devem ser resolvidos pelos Sistemas Gerenciadores de Bancos de Dados Distribuídos e Heterogêneos, ou como são mais conhecidos, Sistemas de Bancos de Dados Heterogêneos. Esse fato leva a corrente de pesquisa na área de *banco de dados distribuídos e heterogêneos* a concentrar-se no problema de como integrar os sistemas existentes - os chamados *legacy systems* - de forma a prover a organização como um todo, informações corretas e, ainda mais importante, de forma mais rápida.

3 Sistemas de Bancos de Dados Heterogêneos

3.1 Introdução

Hoje, o ambiente de desenvolvimento de motores na Mercedes-Benz é caracterizado pelo isolamento, pelo uso de *softwares* e bancos de dados de vários fornecedores. Estes *softwares* são, na maioria, ferramentas de cálculos e simulações, e sistemas CAD, os quais não conseguem entender ou se comunicar uns com os ou tros. Além disso, os sistemas de banco de dados infelizmente não têm nenhuma habilidade para relacionar dados de fontes heterogêneas. Essencialmente, cada departamento envolvido no desenvolvimento de um motor constitui uma ilha de informação com suas próprias ferramentas e fontes de dados.

Existem algumas soluções propostas para o problema do desenvolvimento de aplicações que necessitam acessar dados residentes em diferentes SGBDs. Uma possibilidade seria a criação de padrões que deveriam ser adotados pelos diversos SGBDs. Esta idéia não é tão simples já que depende da obtenção de um consenso entre os diversos fabricantes para se chegar a um padrão. Sabendo que cada um investiu em soluções distintas, seria muito difícil alcançar um consenso.

Outra solução proposta para o problema seria a adoção de um único SGBD no contexto de uma organização ou corporação. Esta alternativa, no entanto, não é factível devido:

- aos requisitos atendidos pelos diferentes SGBDs existentes serem distintos, não havendo um único SGBD que atenda a sua totalidade;
- aos SGBDs serem projetados para plataformas específicas de *hardware* e *software,* e um único SGBD não atende a todos os diferentes ambientes que podem ser encontrados em uma organização de grande porte;
- à migração de um SGBD para outro ser extremamente dispendiosa e difícil devido à necessidade de conversão de dados, de aplicações e treinamento de usuários.

Uma outra possibilidade é a conversão e migração dos dados de um SGBD para outro. Tentativas de migração de dados de uma fonte para outra são dificultadas pela complexidade das tarefas. No caso de migração de todos os dados de um SGBD para outro, as aplicações pré-existentes devem continuar funcionando. Su-

porte a aplicações pré-existentes após uma migração completa de dados é um problema bastante complexo de ser resolvido.

Existem outros projetos propostos para resolver o problema de bancos de dados heterogêneos que precisam compartilhar informações. Neste capítulo, serão abordadas as soluções mais intensivamente abordadas pela literatura, o que corresponde ao estado da arte sobre a integração de sistemas de bancos de dados heterogêneos.

3.2 *Multidatabases*

Pesquisas na construção de arquiteturas completas que suportem o compartilhamento de informação entre sistemas de banco de dados heterogêneos começaram apenas na década de 80. O termo **Sistema de Bancos de Dados Heterogêneos (SBDH)** foi originalmente utilizado para distinguir os trabalhos que tratam as he terogeneidades nos modelos de banco de dados e esquemas conceituais dos trabalhos de *banco de dados distribuídos*[1] que observam principalmente características relacionadas à distribuição. Recentemente, existe um ressurgimento das pesquisas na área de projeto de sistemas de bancos de dados heterogêneos. Este trabalho pode ser caracterizado pelos diferentes níveis de integração dos sistemas de bancos de dados componentes e pelos diferentes níveis de serviços globais providos pelos sistemas de bancos de dados heterogêneos.

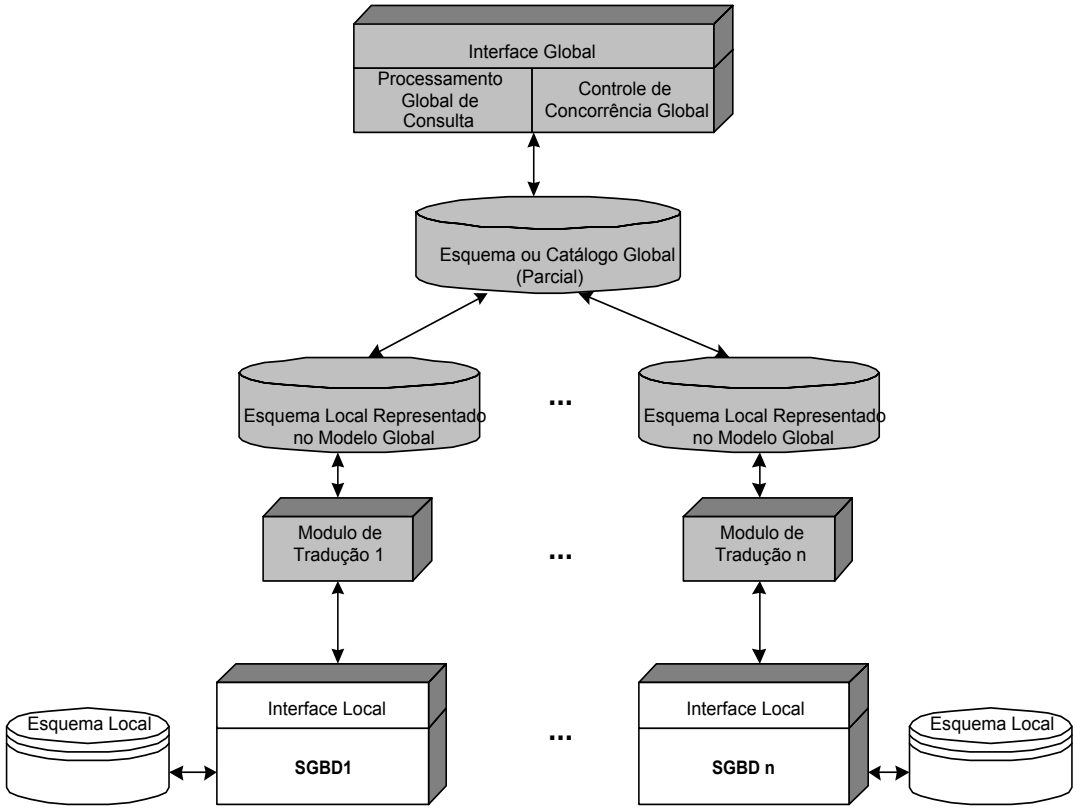
Um **Sistema de Bancos de Dados Heterogêneos** *é um sistema distribuído que atua como interface para múltiplos, heterogêneos e pré-existentes SGBDs locais ou é estruturado como uma camada global do sistema, no topo dos SGBDs locais* [Pire97]. O sistema global (*software* que faz o gerenciamento do SBDH chamado de Sistema de Gerenciamento de Bancos de Dados Heterogêneos) fornece todas as facilidades de bancos de dados e interage com os SGBDs locais através das suas *interfaces* externas. Apesar dos sistemas locais manterem algumas funções globais, os SGBDs locais participam do SGBDH sem modificações e mantêm controle total sobre os dados e processamento locais. Ou seja, os SGBDs locais são **autônomos**. O sistema global fornece algum meio de resolução das diferenças funcionais e de representação de dados entre os SGBDs componentes. O usuário global pode acessar informações de múltiplas fontes com uma simples e direta re quisição. Nesta dissertação, usaremos o termo *multidatabase* com o mesmo si gnificado que sistemas de bancos de dados heterogêneos.

A Figura 3.1 apresenta a arquitetura de um Sistema de Gerenciamento de Bancos de Dados Heterogêneos [Pire97]. Cada SGBD componente compartilha parte ou

1. O termo banco de dados distribuído é utilizado aqui denotando um sistema homogêneo fortemente acoplado e logicamente centralizado, mas distribuído fisicamente entre os bancos de dados componentes.

todos os seus dados através de visões que são disponibilizadas pela sua interface externa. Para os SGBDs locais o SBDH nada mais é do que mais um usuário comum. Um módulo do SBDH residente no mesmo nó da rede que o SGBD local é responsável pela tradução da visão local (no modelo de dados do SGBD local) para a visão local equivalente no modelo de dados global que é acessível pela linguagem de manipulação global. As visões locais expressas no modelo de dados global são então combinadas em um esquema global parcial ou algum tipo de catálogo de dados globais. Este esquema ou catálogo global representa a soma de todas as informações disponíveis no SBDH.

Figura 3.1 Arquitetura Genérica de um SGBDH.



Como visto na seção 2.4 existe uma grande variedade de soluções que são empregadas para resolver o problema de ambientes distribuídos que compartilham informações. Uma diferença marcante nas metodologias é o fato da manutenção ou da autonomia dos SGBDs que compõe o SBDH. Dentre os sistemas distribuídos apresentados, apenas o *banco de dados distribuídos* não oferece essa característica. Ou seja, apenas os *bancos de dados distribuídos* são excluídos dos SBDHs de acordo com a definição que foi dada anteriormente.

A variedade de termos existentes na literatura para denominar os SBDH torna muitas vezes difícil o entendimento. Por exemplo, [BHP92] utiliza o termo *multi-*

databases para SBDH. Já [SL90] utiliza o termo *multidatabase* simbolizando os SBDH em conjunto com os sistemas distribuídos. Ainda segundo [SL90], o termo *sistemas de bancos de dados federados* é correspondente à nossa definição de SBDH.

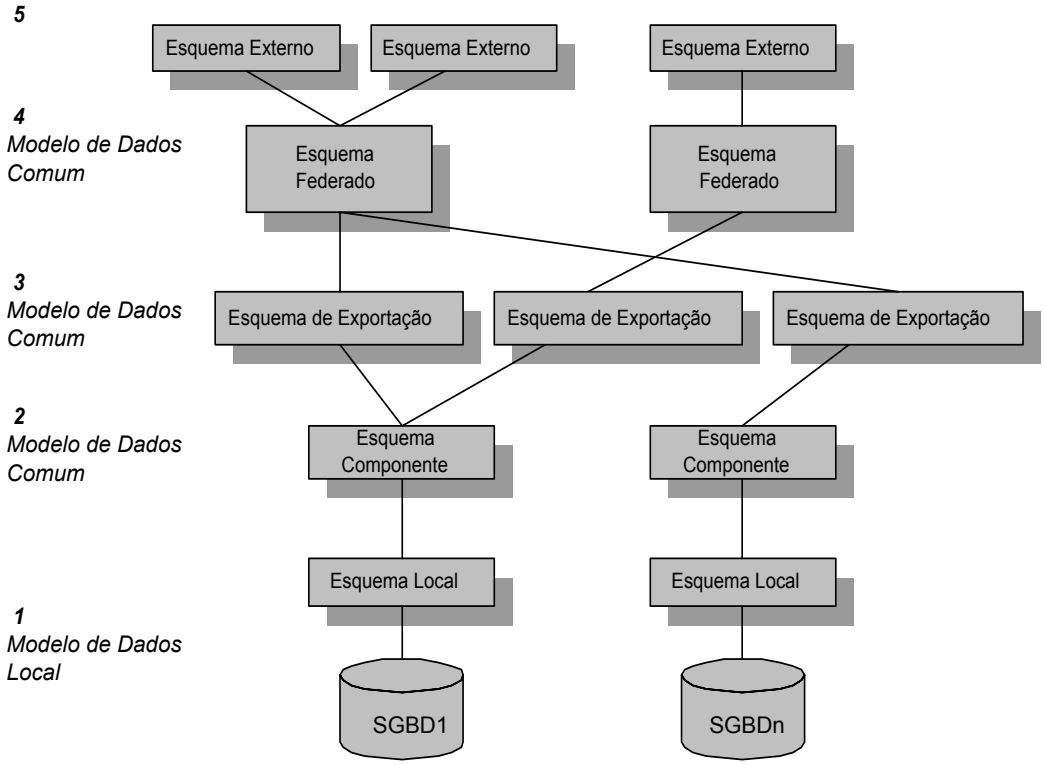
3.3 Taxonomia dos SBDHs

Em [SL90] é apresentada uma taxonomia para os sistemas de bancos de dados heterogêneos baseada no grau de autonomia e no tipo de integração existente entre os componentes do sistema. Os sistemas, dependendo do tipo de integração entre os seus componentes podem ser classificados como *sistemas de acoplamento forte* e *sistemas de acoplamento fraco*.

Nos sistemas com acoplamento forte existe uma autoridade central que é responsável pela administração da federação. As consultas e atualizações podem ser realizadas sem que o usuário saiba os caminhos de acesso ou a localização dos dados. O sistema global tem controle sobre o(s) esquema(s) global(is) e o processamento de transações e consultas globais. Os sistemas com acoplamento forte podem ser construídos com um único esquema global ou com múltiplos esquemas parciais globais. No artigo de [SL90] é proposta uma arquitetura genérica de cinco níveis de esquema para descrever um SBDH (Figura 3.2). Nesta arquitetura, cada banco de dados componente possui um:

- Esquema local (primeiro nível): define todos os seus dados. Este esquema é expresso no modelo de dados nativo do BD, e portanto, diferentes esquemas locais podem ser expressos em diferentes modelos de dados.
- Esquema componente (segundo nível): é a tradução do esquema local para o modelo de dados comum.
- Esquema de exportação (terceiro nível): é a parte do esquema componente que será disponibilizada para a federação. Podem existir diversos esquemas de exportação para cada banco de dados local.
- Esquema Federado (quarto nível): é a combinação de esquemas de exportação; estes esquemas representam as porções do esquema global lógico
- Esquema Externo (quinto nível): define um esquema para um usuário e/ou aplicação ou uma classe de usuários/aplicações. Os usuários globais acessam a federação através destes esquemas.

Figura 3.2 Arquitetura de um SBDH com Acoplamento Forte.



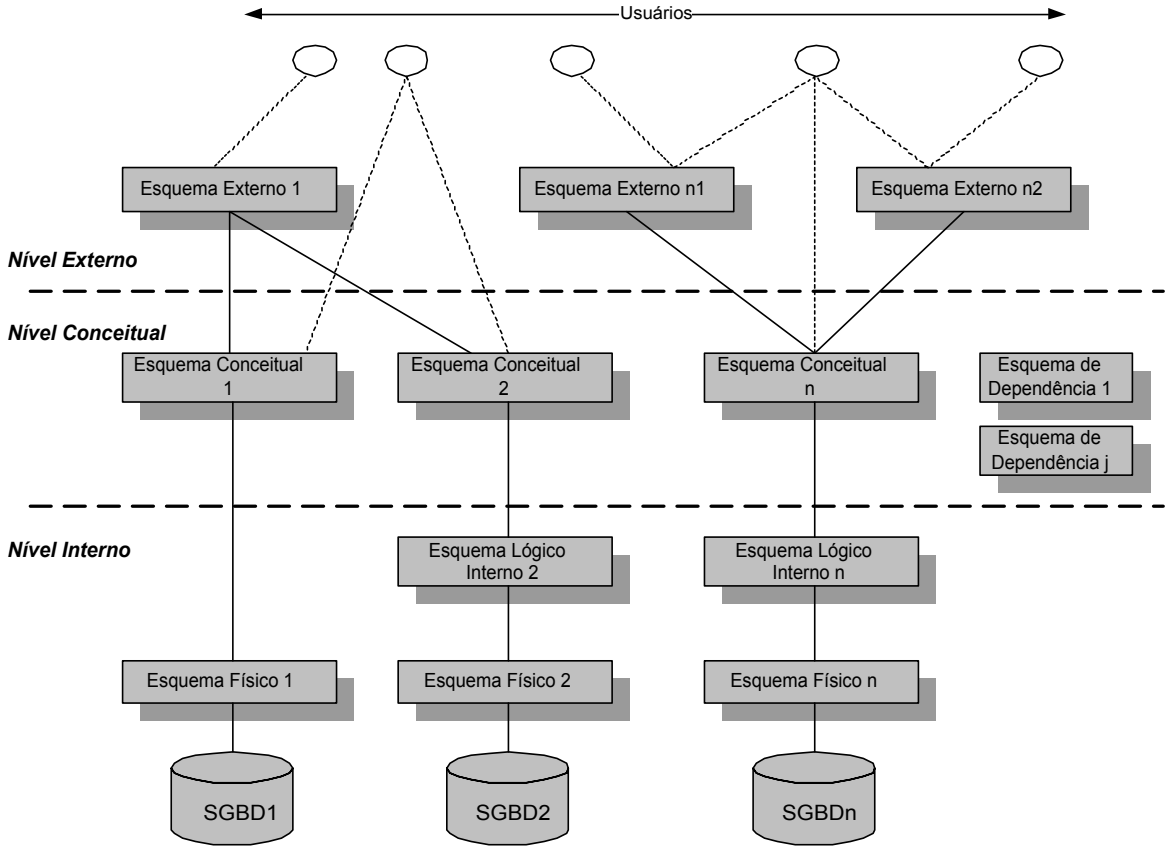
Em sistemas de acoplamento fraco existe uma *interface* para acesso direto entre os SGBDs locais componentes, não existindo um esquema global único. Os usuários são responsáveis pela administração da federação. Não existe uma autoridade central para controlar o acesso à criação de dados. Os usuários globais acessam os dados locais através de uma *interface* que disponibiliza um determinado conjunto de esquemas de exportação. O sistema global suporta todas as transações globais através de ferramentas de linguagem de consulta que proporcionam a integração das informações dos SGBDs locais. Consultas podem ser feitas sobre qualquer dado dos esquemas locais de qualquer nó componente do sistema. Ferramentas de linguagem incluem um espaço global de nomes e funções especiais para mapear informações, de diferentes modelos e representações, para um modelo representativo para o usuário. O usuário é responsável por compreender a semântica dos dados que compõem os esquemas de exportação. De uma maneira geral podemos dizer que uma arquitetura genérica para sistemas de acoplamento fraco pode ser representada pela Figura 3.2 retirando o quarto nível da arquitetura. Na Figura 3.3[1]

1. A arquitetura apresentada na Figura 3.3 é específica de um sistema de linguagens de *multidatabase* sendo que esta arquitetura captura o caso geral de uma arquitetura para sistemas fracamente acoplados [HDRK97].

apresentamos a arquitetura de um sistema fracamente acoplado [LMR90] a qual apresenta três níveis:

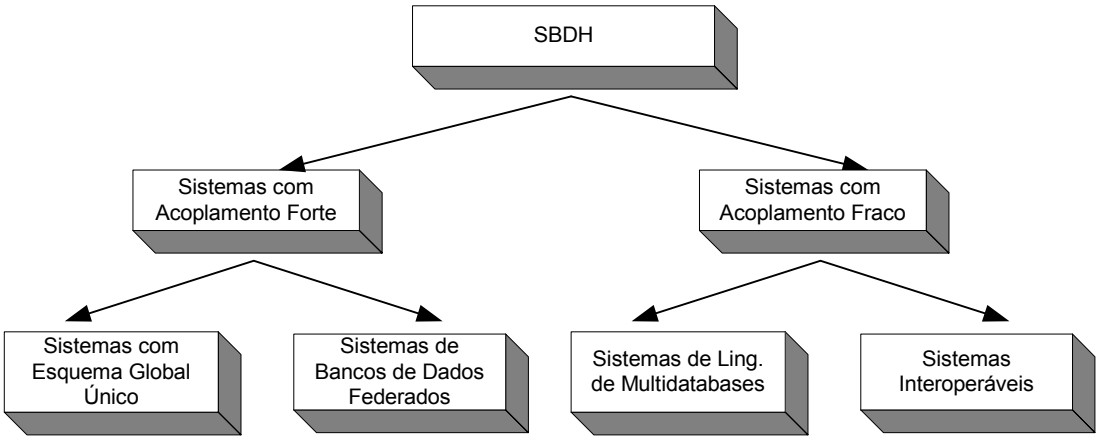
- *Nível Interno*: O nível interno é composto dos SGBDs cada um tendo seu esquema físico.
- *Nível Conceitual*: O nível interno apresenta um esquema conceitual para o nível conceitual. Como pode ser visto na Figura 3.3 o SGBD1 apresenta o esquema conceitual diretamente. Neste caso o esquema conceitual do nível conceitual é o esquema conceitual real do banco de dados componente, isto é, o componente o compartilha completamente com os outros usuários. Para o caso que um banco de dados componente deseja que apenas um subconjunto do seu esquema seja compartilhado, este subconjunto torna-se o conteúdo do esquema conceitual ao nível conceitual e o esquema conceitual real é representado pelo esquema lógico interno no nível interno (veja o SGBD2 e SGBDn na Figura 3.3). Se um modelo canônico é requerido no nível conceitual, como quase sempre é o caso, é responsabilidade do SGBD componente assegurar que o esquema conceitual apresentado no nível conceitual respeite esse requerimento. Neste caso, o componente deve também fazer o mapeamento do esquema conceitual para o esquema local. O nível conceitual também inclue uma facilidade de definir dependências entre coleções de banco de dados. Estes são expressos na arquitetura através dos esquemas de dependência. O propósito destes esquemas é permitir aos administradores da federação especificar as restrições para os bancos de dados inter-relacionados as quais oferecem a habilidade de forçar alguns elementos de consistência na falta dos esquemas integrados. Um exemplo típico de dependências que podem ser especificadas nos esquemas de dependência são equivalências de atributos e/ou domínios.
- *Nível Externo*: Um usuário pode construir esquemas externos no nível externo como sendo visões de coleções de esquemas conceituais. Coleções de bancos de dados podem ser apresentadas com um único banco de dados integrado no esquema externo, mas isto não deve ser confundido com os esquemas integrados dos sistemas fortemente acoplados. Nos sistemas fracamente acoplados os esquemas externos são criados e mantidos com a total responsabilidade do usuário. Não pode ser garantida nenhuma consistência se os conflitos não são resolvidos ou as restrições de integridade dos bancos de dados inter-relacionados não são forçadas.

Figura 3.3 Arquitetura de SBDH com Acoplamento Fraco.



De acordo com a classificação apresentada, sistemas com acoplamento forte e sistemas com acoplamento fraco, podemos ver na Figura 3.4 a classificação dos sistemas apresentados na seção 2.4.

Figura 3.4 Taxonomia dos SBDHs.



3.4 Comparação das Arquiteturas de SBDHs

A proposta do *esquema global único* para o projeto de um SBDH é derivada diretamente dos bancos de dados distribuídos. O esquema global é somente outra camada sobre os esquemas externos locais que fornece uma independência de dados adicional. A principal diferença entre sistemas distribuídos e os sistemas com esquema global é que, no segundo, os esquemas locais são desenvolvidos independentemente do esquema global. Outra grande diferença é que sistemas de esquema global podem integrar esquemas locais de múltiplos modelos de dados. Esta proposta faz com que o acesso aos dados globais sejam bastante amigáveis. Usuários globais vêem um único banco de dados integrado. A interface global é independente de toda a heterogeneidade dos SGBDs locais. Para usuários e aplicações podem ser definidas visões específicas no topo do esquema global. Este é normalmente replicado em cada nó componente para tornar o acesso eficiente.

O esquema global único é construído através da integração dos esquemas dos bancos de dados locais. Esta integração normalmente requer uma homogeneidade dos conflitos existentes entre os esquemas componentes. Devido às diferenças de representação e as interdependências entre dados em diferentes nós, este processo torna-se bastante complexo. Apesar de existir muitas técnicas/metodologias para o processo de integrar múltiplos e distintos esquemas, este ainda é um processo muito dependente do trabalho humano. Os projetistas do SGBDH devem possuir um grande conhecimento de todos os esquemas locais e dos requisitos globais para poder decidir como integrá-los. O maior problema desta proposta é justamente o conhecimento geral necessário para projetar tal esquema.

Outra dificuldade apresentada por esta proposta está na parte de manutenção do esquema global. Mudanças nos esquemas locais devem ser refletidas no esquema global. As técnicas de integração usadas no projeto do esquema global e os tipos de mudanças nas representações dos dados locais podem complicar o mapeamento destas a nível global. Mudanças locais podem forçar a reconsideração de muitas decisões de projeto feitas durante o processo inicial de integração. Novamente, a pessoa responsável pela administração do BD (DBA) deve possuir um grande conhecimento global de todos os esquemas locais, do esquema global e das decisões iniciais do projeto.

Para amenizar os problemas encontrados no projeto do esquema global único pode ser utilizado esquemas globais parciais. Estes são os chamados *sistemas de bancos de dados federados*. Para os usuários globais deste sistema, um determinado esquema parcial aparece como um esquema global único porque eles não acessam partes do sistema que não estejam incluídas neste esquema parcial [Pire97].

A proposta de *linguagem de multidatabase* [MRJ99, LMR90] busca resolver alguns dos problemas associados com os esquemas globais, tais como nível de conhecimento exigido dos administradores (DBAs), tempo de desenvolvimento necessário para a criação do esquema global (parciais ou único), as dificuldades de manutenção e os requisitos de processamento/armazenamento dos nós locais componentes. Um exemplo de sistema que utiliza essa solução é o MRDSM, com a linguagem MSQL [LMR90].

Ao contrário do esquema global, esta proposta coloca grande parte da responsabilidade de integração, sobre o usuário. Mas, de certa forma, fornece funções de suporte e maior controle da informação. São fornecidos operadores adequados e construtores específicos para os usuários resolverem os conflitos semânticos a vários níveis de abstração. A maioria da linguagens de múltiplos bancos de dados são similares ao SQL com um acréscimo significante na sua funcionalidade.

Os sistemas que utilizam-se destas linguagens, devem fornecer ao usuário quais são as informações disponíveis e os locais onde podem ser encontradas. É necessário que o usuário saiba exatamente qual informação é necessária e o local provável de sua localização. Ou seja, a responsabilidade de integração antes dada ao administrador global (na alternativa do esquema global), agora é passada para os usuários ou administrador local, que são responsáveis por entender os esquemas e ainda de detectar e resolver os conflitos semânticos. Aqui, os usuários devem ter conhecimento global das diferenças de representação e a origem dos dados, sendo que apenas das informações a serem utilizadas. A linguagem deve oferecer operadores adequados para que os usuários resolvam os conflitos semânticos em vários níveis de abstração. Algumas dessas características interessantes da linguagem MSQL apresentadas em [BBE98] são: Nomes globais, Dependências entre bancos de dados e Consultas entre banco de dados.

Uma grande facilidade acrescida nestas linguagens envolve a manipulação de representação de dados. Como existem diferenças de representação na execução de uma consulta, a linguagem deve ser capaz de transformar a informação fonte na representação mais significativa para o usuário.

[SL90] aponta a metodologia de acoplamento fraco como a melhor solução para sistemas com um grande número de bancos de dados autônomos utilizados apenas para leitura, como por exemplo, sistemas de informações públicas. Já a metodologia de acoplamento forte é apontada como melhor solução para sistemas que necessitam dos controles extras providos por esta metodologia. Deve-se levar em consideração os objetivos do SBDH no momento da escolha da arquitetura adequada. Se o objetivo é montar um sistema que seja utilizado por usuários leigos, com certeza os sistemas interoperáveis e os sistemas de linguagens de *multidatabases* não devem ser escolhidos.

3.5 Funcionalidades providas pelos SGBDH

Segundo [TTCB+90, BHP92], um SGBDH pode oferecer diferentes tipos de funcionalidades. Algumas delas são a *integração de esquemas*, o *gerenciamento de consultas distribuídas*, o *gerenciamento de transações distribuídas*, *funções administrativas* e *tratamento de diferentes tipos de heterogeneidades*. Integração de esquemas é a forma pela qual os usuários podem ter uma visão lógica dos dados distribuídos. O gerenciamento distribuído de consultas cuida da análise, otimização e execução de consultas que referenciam dados distribuídos. Gerência de transações distribuídas trata das propriedades de atomicidade, isolamento e durabilidade das transações em um ambiente distribuído. As funções administrativas incluem autorização, autenticação, definições de restrições de integridade e gerência do dicionário de dados. Resolução das heterogeneidades inclue o tratamento das diferenças de *hardware*, sistemas operacionais, protocolos, fabricantes e/ou modelos de dados de SGBDs.

3.5.1 Integração de Esquemas em SGBDH

Se não houvessem conflitos nos valores e estruturas dos dados, a integração de banco de dados seria uma tarefa trivial. Utilizando uma tecnologia como o *middleware*, a integração de banco de dados heterogêneos estaria praticamente concluída. Entretanto, em geral existem duas fontes potenciais de dificuldades na integração de BDs: diferenças nos esquemas e diferenças nos dados.

Cada banco de dados local possui seu esquema local, descrevendo a estrutura dos dados no BD. Cada usuário tem uma visão, descrevendo a porção dos dados distribuídos que são do seu interesse. É através da integração de esquemas que se obtem o mapeamento entre a visão dos usuários e os esquemas locais. A **integração**

de esquemas é *o processo de desenvolvimento de um esquema conceitual, livre de duplicações ou heterogeneidades, que integre uma coleção de esquemas locais* [Ham94].

[DH84] utiliza o termo integração de esquema apenas como sendo a resolução de conflitos das estruturas dos esquemas. Ainda segundo [DH84], o processo de construção do esquema global pode ser dividido em duas classes de problemas: diferenças de esquemas e diferenças de dados. Integração de esquemas inclue a resolução de *conflito de nomes*, *conflito na representação dos dados*, *diferenças estruturais* e *diferenças na abstração* [DH84]. Os problemas na integração de dados podem ser descritos através de duas situações distintas: *Os bancos de dados são mutuamente inconsistentes, mas corretos* e *os bancos de dados são mutuamente inconsistentes e incorretos.*

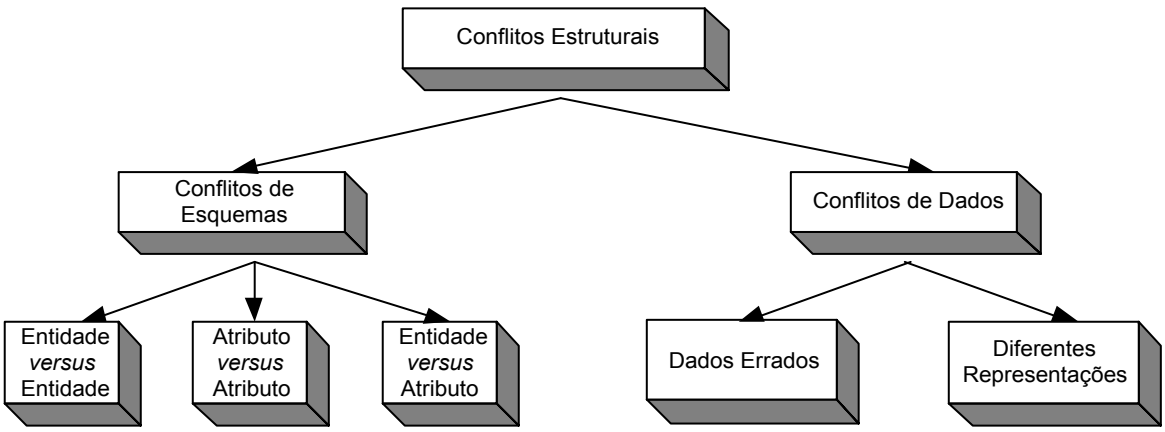
- ***Conflito de nomes***: Ocorre quando dados semanticamente iguais possuem nomes diferentes (os atributos neste caso são chamados *sinônimos* [BBE98, DH84]). Podem existir ainda entidades de BDs distintos com o mesmo nome para um atributo que representa diferentes objetos ou relacionamentos no mundo real, ou seja, dados semanticamente distintos mas com nomes idênticos (neste caso os atributos são chamados de *homônimos* [BBE98, DH84]).
- ***Conflito na representação dos dados***: Atributos correspondentes em dois ou mais bancos de dados podem ter tipos ou representações diferentes.
- ***Diferenças estruturais***: Podemos citar como diferença estrutural a falta de entidades ou atributos nos BDs. Diferenças na modelagem também estão incluídas nesta categoria, onde uma entidade de um BD pode corresponder a apenas um atributo de uma entidade em outro BD. Por exemplo, o conceito de endereço do mundo real pode ser modelado como um atributo (cadeia de caracteres representando todo o endereço) de uma relação em um banco de dados. Da mesma forma, o endereço pode ser modelado como uma entidade, com os atributos rua, número, cidade, CEP, etc.
- ***Diferenças na abstração***: Entidades e/ou atributos são representados em diferentes níveis de generalização em dois bancos de dados distintos.
- ***Os bancos de dados são mutuamente inconsistentes, mas corretos:*** Considere dois BDs contendo a entidade EMP com os seguintes atributos: *EmpNo* (inteiro) e Salário (inteiro). O atributo *EmpNo* é a chave primária da entidade em cada banco de dados. Suponha que exista uma tupla (t1) na tabela que representa a entidade EMP tal que EmpNo(t1) = 1 e o Salário(t1) = 25 em um banco de dados e uma tupla (t2) tal que EmpNo(t2) = 1 e o Salário(t2) = 30 no outro banco de dados. Uma razão para esta discrepância entre os valores do atributo Salário pode acontecer porque as entidades que parecem ser a mesma na realidade são diferentes. Então, embora as tuplas possuam o mesmo valor para o atributo *EmpNo* (chave primária), elas representam diferentes empregados no mundo real. Isso implica que *EmpNo* não é um identificador para as

entidades EMP. Outra possibilidade para esta diferença é que, embora *EmpNo* seja realmente um identificador para a entidade EMP, os atributos são diferentes (homônimos). Por exemplo, eles representam o salário para dois cargos diferentes.

- ***Os bancos de dados são mutuamente inconsistentes e incorretos:*** Neste caso, a solução mais viável é usar o dado mais confiável que pode ser conseguido com a ajuda do especialista.

Segundo [KS91], duas causas básicas de conflitos estruturais são identificadas no momento da integração de bancos de dados heterogêneos. Uma classe de conflitos acontece quando diferentes esquemas de bancos de dados utilizam diferentes estruturas para representar a mesma informação (tabelas e atributos). Outra classe de conflitos surge quando diferentes esquemas de banco de dados utilizam estruturas similares mas especificações diferentes (como diferenças de nomes e domínios) para representar a mesma informação. [KS91] também propõe um *framework* de forma a identificar os possíveis conflitos originados no momento da integração de banco de dados heterogêneos. Este esquema de possíveis conflitos estruturais é válido para a integração de banco de dados relacionais. Em [KGCS95] é proposto um novo esquema tomando como base o proposto em [KS91], mas este leva em consideração a integração dos bancos de dados orientados a objetos. Dessa forma, os problemas apresentados anteriormente por [DH84] podem ser melhor estruturados utilizando o esquema da Figura 3.5.

Figura 3.5 Classificação dos Conflitos Estruturais.



Conflitos de Esquema

A. Conflitos Entidade *versus* Entidade

1. Conflitos uma para uma entidade
   a. Conflitos entre nomes de entidades
      1) Diferentes nomes para entidades equivalentes
      2) Mesmo nome para entidades diferentes
   b. Conflitos entre as estruturas das entidades
      1) Falta de atributos
      2) Falta de atributos, porém com presença implícita
   c. Conflitos entre as restrições de integridade das entidades
2. Conflitos muitas para muitas entidades

B. Conflitos Atributo *versus* Atributo

1. Conflitos um para um atributo
   a. Conflitos entre nomes dos atributos
      1) Diferentes nomes para atributos equivalentes
      2) Mesmo nome para atributos diferentes
   b. Conflitos de valores *default*
   c. Conflitos entre as restrições de integridade dos atributos
2. Conflitos muitos para muitos atributos

C. Conflitos Entidade *versus* Atributo

Conflitos de Dados

A. Dados errados

1. Conflitos uma para uma entidade
2. Dados obsoletos

B. Diferentes representações para o mesmo dado
(Mesma representação para dados diferentes)

1. Expressões diferentes
2. Unidades diferentes
3. Precisões diferentes

O desenvolvimento de protótipos e projetos no esforço de integrar esquemas de bancos de dados heterogêneos começou no final dos anos 70 e início dos anos 80, com ênfase em metodologias para projeto de bancos de dados relacionais. Nesta primeira fase, havia o objetivo de produzir uma descrição de um esquema global de um banco de dados baseada nas diferentes perspectivas dos usuários. Após essa fase, os esforços das pesquisas focalizaram a integração de banco de dados orientados a objetos, para obter uma visão global de informações relacionadas em diferentes domínios. O termo **integração de esquemas** [RR99], utilizado com muita

freqüência na literatura, inclue *a resolução da heterogeneidade semântica*, *a junção* e *a reestruturação de esquemas* [Ham94].

Existem duas metodologias mais conhecidas para a integração de banco de dados heterogêneos, que são: esquema global e banco de dados federados [BHP92]. A integração de esquema é o coração destas metodologias. De acordo com [RR99], integração de esquema é o processo de gerar um ou mais esquemas através de outros pré-existentes. Estes esquemas representam a semântica dos bancos de dados que estão sendo integrados e são usados como entrada para o processo de integração. A saída do processo é um ou mais esquemas que representam a semântica do banco de dados integrado. Tais esquemas são representados usando um modelo de dados comum, e eles escondem qualquer heterogeneidade resultante das diferenças semânticas ou diferenças de modelos dos bancos de dados integrados.

Ainda de acordo com [RR99], o termo integração de esquemas é usado na literatura para expressar a integração de visão, devido a estas duas metodologias utilizarem técnicas comuns. Entretanto, esses dois processos podem ser diferenciados de várias maneiras:

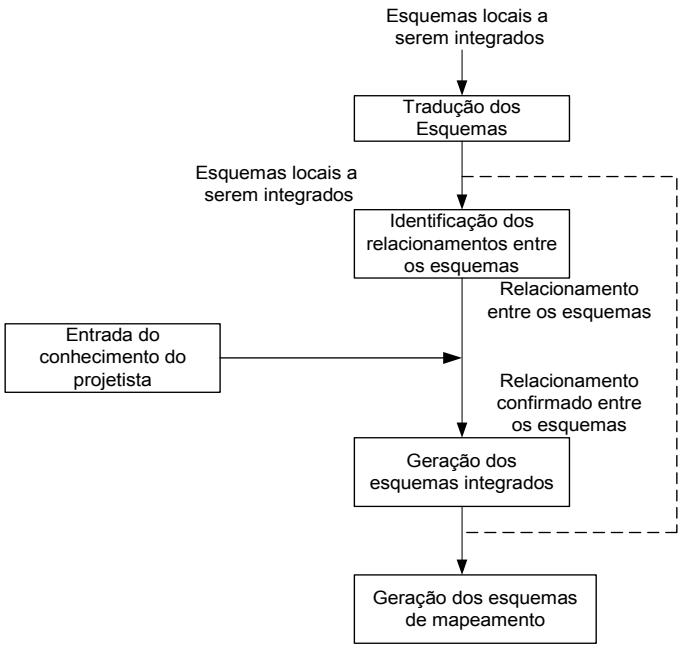
- Integração de visão é o processo de gerar um único esquema integrado para múltiplas visões de usuários e é usado com mais freqüência no projeto de um novo esquema de banco de dados. Portanto, integração de visão é utilizado no projeto *top-down* de banco de dados. A partir de múltiplas visões de usuários, é gerado um esquema integrado correspondente a essas visões e então é projetado o banco de dados correspondente a este esquema. Ao contrário, integração de esquema, é um projeto *bottom-up* porque trata com integração de esquemas já existentes;
- Na integração de visão, o usuário define visões utilizando um único modelo de dados. Na integração de esquemas, como os bancos de dados podem ser heterogêneos, os esquemas a serem integrados podem ser representados por múltiplos modelos de dados.
- Quando a integração de visão é realizada, as visões dos usuários não refletem os dados existentes nos bancos de dados. Entretanto, na integração de esquemas, os esquemas representam dados existentes nos banco de dados. Esta é uma grande diferença, porque o esquema gerado pelo processo de integração não pode violar a semântica dos bancos de dados existentes. Entretanto, na integração de visões, como estas representam objetos abstratos, existe maior flexibilidade na interpretação semântica.

Integração de esquema é um problema complexo e que consome tempo. Em primeiro lugar porque muitas representações de esquemas não podem capturar a semântica do banco de dados completamente. Portanto, o processo de integração requer uma grande interação com o projetista e o administrador dos bancos de dados para entender a semântica dos bancos de dados e assegurar que após a integra-

ção esta permaneça compatível. Isto significa que o processo de integração não pode ser completamente automatizado[HMS93]. [RR99] o divide em quatro fases (Figura 3.6). Note que esses passos devem ser realizados de forma interativa para resolver a heterogeneidade e chegar a uma representação integrada dos esquemas componentes.

- *Tradução de Esquemas*: Os esquemas que serão integrados são traduzidos em um esquema de modelo comum, tal como, o modelo relacional [Che76].
- *Identificação dos relacionamentos entre os esquemas*: Esta fase tem como objetivo a identificação de objetos (entidades, atributos e relacionamentos) entre os esquemas que serão integrados, através da análise da semântica destes objetos, bem como o agrupamento dos relacionamentos desses objetos em suas categorias. Nesta fase, a ajuda do especialista contribui para que os resultados destas identificações sejam confiáveis.
- *Geração dos esquemas integrados*: O conjunto de resultados produzidos na fase anterior é utilizado para gerar o esquema integrado. Para isso, entretanto, devem ser resolvidos os diversos tipos de heterogeneidades que podem existir entre os objetos [SK93, KS91, KGCS95, KS95, DH84, HM93].
- *Geração de esquemas de mapeamento*: Este passo é o responsável por armazenar informações sobre o mapeamento entre objetos na transformação dos esquemas integrados e objetos nos esquemas locais.

Figura 3.6 Fases da Integração de Esquemas.



Em [KGCS95] são apresentados os conflitos e possíveis soluções para o problema de heterogeneidade durante a integração de fontes de dados. Em [SK93] são citados os conflitos de forma a situar as possíveis similaridades semânticas entre dois objetos. Da mesma forma [KS95] endereça o problema dos conflitos com relação aos aspectos semânticos e não com relação à heterogeneidade de esquemas/representação/estrutura como [KS91, KGCS95].

3.5.2 Gerenciamento de Consultas Distribuídas

O gerenciamento de consultas globais provê a habilidade de combinar dados de diferentes bancos de dados locais em uma única operação de recuperação. Sistemas distribuídos podem ou não oferecer esta facilidade. Em alguns sistemas, para uma aplicação conseguir dados de diferentes bancos de dados é necessário enviar explicitamente consultas para os bancos de dados individualmente. Se o gerenciamento de consultas é oferecido, a aplicação envia apenas uma única consulta, que é decomposta em um conjunto de subconsultas - uma para cada SGBD local que está envolvido na execução.

Então, o otimizador de consultas cria uma estratégia de acesso que especifica que SGBDs locais estão envolvidos, a tarefa de cada um, como será combinado o resultado intermediário, e onde o processamento global vai ocorrer. As restrições globais devem ser verificadas e garantidas durante a execução da consulta. O processamento inicial da consulta normalmente ocorre no nó que a consulta foi submetida sendo a execução da consulta distribuída pelo sistema.

Durante a execução da consulta global, pode ocorrer a tradução de consultas de acordo com a passagem pelas várias camadas do sistema. Estas traduções permitem a existência de diferentes linguagens e representações entre as camadas e, também, resolvem as diferenças de representação.

Se o gerenciamento de consulta distribuída é oferecido pelo sistema, existem muitas possibilidades para otimizar as consultas e gerenciar sua execução. Dependendo das características do sistema, pode ser desejável explorar o paralelismo na execução das consultas. A replicação e/ou fragmentação dos dados também adicionam complexidade [Bar97]. Um método simples de gerenciamento de consultas é mover todos os dados desejados para o nó de onde a consulta é oriunda e então combiná-los. Métodos mais sofisticados podem considerar uma grande quantidade de algoritmos e fatores nas avaliações de tais consultas.

3.5.3 Gerenciamento de Transações Distribuídas

O conceito tradicional de transações com tempo curto de duração e atomicidade não é compatível para ambientes de múltiplos bancos de dados. Transações em

sistemas de bancos de dados que compartilham informações envolvem múltiplos e separados SGBDs bem como várias camadas de traduções em dados e/ ou consultas. Mais importante é que os SBGDs locais possuem autonomia, e dessa forma, não estão sujeitos ao controle global.

Controle de concorrência organiza o acesso aos dados de transações concorrentes de forma que sejam feitas com aparência serial. Entretanto, isso requer o conhecimento de todos as transações ativas no momento bem como a habilidade de controlar o acesso aos dados. *Interfaces* de usuários em SGBDs normalmente não oferecem informações sobre transações de outros usuários ou dados que estão com restrição de acesso no momento, timestamps, etc. Ainda, diferentes SGBDs podem utilizar diferentes controles de concorrência. Devido a isso, muitos SGBDHs restrigem o acesso às informações globais apenas para recuperação, sendo as atualizações realizadas através da *interface* do SGBD local.

O **gerenciamento de transações distribuídas** *fornece a habilidade de ler e/ou atualizar dados de múltiplos nós através de uma única transação, preservando as propriedades de transação*. Esta capacidade pode ou não ser oferecida pelos sistemas de bancos de dados distribuídos. Se é oferecido, existem dois aspectos considerados: protocolos de controle de concorrência e protocolos de *commit*.

Protocolos de controle de concorrência asseguram que a execução de transações locais, quando originada pelo nó local ou um nó remoto, encontra o padrão de isolamento do sistema local. Em muitos casos isso significa que a execução é serializada; isto é, a execução atual é equivalente à execução serial das transações em alguma ordem. Protocolos de controle de concorrência são projetados para assegurar que transações distribuídas são globalmente serializadas; isto é, que a serialização dos componentes locais de todos os bancos de dados são compatíveis com alguma serialização global de todas as transações distribuídas.

Protocolos de *commit* locais garantem a atomicidade e durabilidade de transações locais; ou todas as ações de uma transação local são completadas (*commit)* ou então nenhuma é. Já o protocolo de *commit* distribuído garante a atomicidade e durabilidade global; ou todas as subtransações locais de uma transação distribuída são completadas (*commit* global) ou então nenhuma é completada.

Estes protocolos distribuídos trazem um custo operacional, bem como custo de implementação, já que a falha de um nó pode deixar dados bloqueados em outros nós por longos períodos de tempo. Então, pode ser vantajoso não tê-los se não forem necessários. Em algumas situações, a semântica dos bancos de dados e as aplicações podem fazer ambos os protocolos desnecessários. Em [BGS92, CR94, SRK91, TTCB+90, GRS97] podem ser encontrados maiores detalhes.

3.5.4 Administração

Nos sistemas de bancos de dados heterogêneos, as funções administrativas podem ser disponibilizadas em diferentes graus de centralização e transparência. Como função de administração temos: *autorização de usuários* (segurança de uma forma geral), *definir e assegurar restrições de integridade*, e *a manutenção de informações dos esquemas*.

Gerenciamento de Autorização

Segurança em sistemas distribuídos não é uma tarefa fácil. Alguns dos problemas incluem canais de comunicação não confiáveis, vários níveis de segurança em diferentes nós, e o grande número de usuários. O acesso a múltiplos sistemas, significa, na maioria dos casos, a necessidade de múltiplos códigos de autorização e identificação [Rüt 99].

De acordo com o grau de centralização desta funcionalidade, podemos dividí-la em três casos distintos. Completamente descentralizada, onde cada usuário possui uma identificação e senha garantida pelo administrador local (DBA) e assegurada pelo SGBD local. A segunda opção é que a segurança do sistema seja completamente centralizada, onde cada usuário possui uma única identificação e senha garantida pelo administrador global e assegurada pelo gerenciador de consultas globais. Como terceira e última opção, temos uma situação intermediária as apresentadas, com o usuário possuindo uma identificação e senha global, mas asseguradas pelos bancos de dados locais.

As vantagens do sistema centralizado é que o usuário tem acesso aos bancos de dados integrados, mas não tem acesso ao banco de dados local. Nos sistemas descentralizados, não é possível prevenir o acesso aos dados parcialmente, o que a torna menos interessante.

Gerenciamento de Restrições de Integridade

O sistema global precisa especificar alguns métodos e assegurar as restrições de integridades nos relacionamentos e dependências entre os bancos de dados, devido ao fato de sistemas locais diferentes poderem representar dados semanticamente equivalentes ou dados relacionados semanticamente. Mais uma vez, temos duas possibilidades de garantir as restrições de integridade. A primeira delas é de forma centralizada, através do esquema global conceitual. A segunda é localmente, através dos esquemas locais. A vantagem da forma centralizada mais uma vez é que regras de integridade distribuída (ou seja, regras que dependem de dados armazenados em diferentes bancos de dados) podem ser garantidas.

Restrições de integridade globais requerem um sistema de policiamento para definir como gerenciá-las. Como exemplo temos uma restrição de atualização nos bancos de dados integrados, onde a atualização de um banco de dados deve atualizar o objeto equivalente no outro banco de dados. Se a atualização é propagada, a autonomia do nó pode ser comprometida. Caso contrário, o primeiro nó também deve rejeitar a atualização (função de integridade) ou então aceitá-la (violando a restrição de integridade o que causa inconsistência nos dados).

Transparência e Manutenção de Esquemas

Transparência é a habilidade de trabalhar com dados distribuídos sem ter o conhecimento de onde estão ou mesmo se existe uma distribuição. Existem dois níveis de transparência relevantes: o nível do usuário e o nível do administrador do banco de dados.

Os usuários podem ter que especificar o nome da máquina ou endereço, o identificador do SGBD, o nome do banco de dados e o nome do dado que deseja acessar (nenhuma transparência). Por outro lado, os usuários podem apenas especificar os nomes lógicos dos ítens de dados.

Existe também o nível de transparência provido pelo administrador para programadores e usuários. O SBD pode manter individualmente múltiplos dicionários de dados em múltiplos nós descrevendo quais dados podem ser encontrados e onde. Estes dicionários de dados podem representar o esquema global e descrever todos os dados do sistema, ou eles podem descrever apenas alguns dados de interesse particular de determinados usuários. Por outro lado, o sistema pode tomar todas as decisões sobre o local e manutenção dos dados e manter os dicionários automaticamente.

3.5.5 Resolução de Heterogeneidade

A comunidade científica, na maioria das vezes, considera apenas a heterogeneidade dos modelos dos SGBDHs. Mas, a heterogeneidade pode surgir por vários pontos, já que sistemas de banco de dados heterogêneos agrupam bancos de dados distintos, implementados fisicamente em *hardware* e *software* distintos, que utilizam protocolos distintos, além de serem gerenciados por sistemas operacionais também diferentes. Além disso, os SGBDs envolvidos podem ser originados de fabricantes diferentes e/ou possuírem modelos de dados diferentes.

Diferentes *hardwares* e sistemas operacionais podem utilizar diferentes representações para os dados (diferentes representações para números de ponto flutuante, por exemplo). Ainda, diferentes fabricantes de SGBDs podem utilizar diferentes formas de definir e manipular os dados. Diferentes modelos de dados podem apresentar dificuldades para a tradução dos esquemas e o problema da tradução de con-

sultas, característica importante do ponto de vista de performance. Portanto, o SGBDH deve providenciar a resolução de todos os tipos de heterogeneidades existentes em SBDH.

3.6 Sistemas Heterogêneos Existentes

Abaixo listamos alguns projetos de sistemas distribuídos que agem como interface para múltiplos bancos de dados locais. Os sistemas escolhidos apresentam-se com mais freqüência na literatura. Estes projetos são os resultados das pesquisas de uma grande variedade de países e instituições. Alguns são aplicados para a resolução de uma área específica. Outros são sistemas completamente comerciais. A grande quantidade de projetos e organizações indicam a importância desta área. A classificação dos sistemas segue a taxonomia proposta por [SL90].

3.6.1 Não Orientados a Objetos

Amoco Distributed Database System (ADDS) [BR95] - Amoco Production Company, Research

- O projeto começou no final de 1983 com o objetivo de integrar os bancos de dados distribuídos através da corporacão e foi um dos primeiros projetos nesta área. Na terminologia de [SL90], o ADDS é um sistema federado fortemente acoplado que suporta múltiplos esquemas federados. É baseado no modelo de dados relacional e utiliza uma extensão da álgebra relacional como linguagem de consulta. Os bancos de dados locais são mapeados em múltiplos esquemas de bancos de dados federados, chamados de *Composite Database Definitions* (CDB). Os mapeamentos são armazenados no dicionário de dados do ADDS, que é completamente replicado em todos os nós do ADDS. Um CDB é usualmente definido para cada aplicação, mas existe também o caso em que um CDB é compartilhado entre as aplicações. Os CDBs suportam a integração de modelos de dados hierárquicos, relacionais e de rede. Dentre os SGBDs integrados pelo ADDS encontram-se IMS, SQL/DS, DB2, RIM, INGRES e FOCUS[1]. A interface do usuário consiste de uma API (*Application Program Interface*) e uma interface interativa. A API é composta de um conjunto de procedimentos que oferecem o acesso aos programas de aplicação do ADDS. A interface interativa permite que usuários executem consultas, mostra os resultados das consultas e efetua o salvamento de tais resultados. A interface interativa nada mais é que um programa de aplicação que utiliza a API para prover uma interface de alto nível para o ADDS. O usuário pode tanto montar suas próprias consultas (usuários mais experientes) como ser guiado por

1. IMS, SQL/DS e DB2 são marcas registradas da *International Business Machines Corporation*. RIM é marca registrada da *Boeing Computer Services*. INGRES é marca registrada da *Ingres Corporation*. E finalmente, FOCUS é marca registrada da *Information Builders*, *Inc*.

menus para compor as consultas (usuários menos experientes com o ADDS e com a linguagem SQL). Ainda em relação à interface, este sistema oferece a facilidade de salvamento de consultas num catálogo de consultas, sendo que estas podem ser escolhidas e modificadas pelo usuário antes de serem enviadas para execução. As consultas submetidas para execução são compiladas e otimizadas para um custo mínimo de transmissão de dados. A nível de sistemas operacionais é percebido a presença de VM, MVS e UNIX[1] entre os sistemas que encontram-se geograficamente distribuídos. O ADDS mantém a autonomia dos sistemas locais e não requer qualquer mudança ao SGBD local. A única comunicação entre o ADDS e o SGBD local é a submissão da consulta e a recuperação dos dados.

DATAPLEX [TTCB+90] - General Motors Corporation

- É um sistema federado fortemente acoplado que suporta múltiplos esquemas federados. Permite que consultas e transações recuperem e atualizem os dados distribuídos pelos diversos sistemas já que oferece transparência na localização dos dados. Neste ambiente, diferentes sistemas gerenciadores de dados podem operar em diferentes sistemas operacionais que por sua vez podem utilizar diferentes protocolos de comunicação. O modelo relacional é utilizado como modelo de dados global. Como podem ser utilizados diferentes modelos de dados, a definição dos dados para cada banco de dados compartilhado pelo sistema de banco de dados heterogêneo é transformado para uma definição de dados relacional ou esquema conceitual equivalente. O modelo conceitual é implementado como um conjunto de esquemas relacionais, um para cada localização. As relações de cada localização representam os dados que precisam ser acessados pelos usuários deste local. Conseqüentemente, os esquemas conceituais nem são centralizados e nem replicados. A interface do usuário é apresentada por uma linha de comando. O protótipo do DATAPLEX integra os SGBDs IMS e INGRES operando sobre o comando dos sistemas operacionais MVS E VMS[2] respectivamente.

Integrated Manufacturing Data Administrations System (IMDAS) [TTCB+90] - National Institute of Standards and Technology, U. Florida

- É um sistema federado fortemente acoplado com um único esquema global. O objetivo deste sistema é oferecer o acesso as muitas fontes de dados entre os vários sistemas empregados no processo de fabricação. O modelo de dados integrado é o *Semantic Association Model,* um modelo de dados semântico capaz de representar estruturas complexas, relacionamentos e muitos requisitos de integridade encontrados numa fábrica. Um esquema auxiliar mapeia o modelo global para os bancos de dados que fazem parte da federação, supor-

1. VM e MVS são marcas registradas da *International Business Machines Corporation*. UNIX é marca registrada da AT&T.
2. VMS é uma marca registrada da *Digital Equipment Corporation*.

tando tanto o particionamento horizontal como o vertical de uma dada classe de objetos. Em geral, o IMDAS suporta tanto recuperação como atualização distribuída, mas o esquema auxiliar não suporta replicação, sendo esta uma limitação significante do sistema.

Ingres/STAR [TTCB+90] - Ingres Corporation

- *Ingres Corporation* surgiu de um projeto de pesquisa sobre a tecnologia de bancos de dados relacionais da universidade de *Berkeley*. O *Ingres*/STAR oferece acesso transparente aos dados distribuídos. O *Ingres*/STAR é uma federação fortemente acoplada que suporta múltiplos esquemas federados [SL90]. O SGBD *Ingres* oferece o acesso ao banco de dados *Ingres*. A interface do *Ingres* submete consultas SQL ao SGBD *Ingres* que recuperam os dados armazenados no banco de dados. O *Ingres Gateway* oferece um método onde dados armazenados em outros sistemas gerenciadores de dados comportam-se como se estivessem armazenados no banco de dados *Ingres*, tornando-os disponíveis para a *interface*. O sistema *Ingres*/STAR permite que usuários acessem um banco de dados distribuído, que é definido como uma coleção de tabelas de um ou mais bancos de dados *Ingres*. Qualquer conjunto de tabelas de qualquer conjunto de bancos de dados *Ingres* pode ser combinada para formar um novo banco de dados distribuído *Ingres*/STAR. Isto inclue não só bancos de dados de um SGBD *Ingres* mas também bancos de dados acessíveis através do *Ingres Gateway* e outros bancos de dados *Ingres*/STAR. Um único servidor pode ser utilizado para múltiplos bancos de dados distribuídos, como também podem existir múltiplos servidores na rede. O acesso aos bancos de dados distribuídos é feito de modo transparente no sentido que uma vez um banco de dados seja criado, os usuários do banco de dados não precisam conhecer nada sobre a existência de bancos de dados individuais.

Mermaid [TTCB+90] - *Data Integration, Inc.*

- É um sistema federado fortemente acoplado que suporta múltiplos esquemas federados [SL90]. Na verdade, o *Mermaid* não é um sistema gerenciador de bancos de dados, mas uma interface que localiza e integra dados que são mantidos por um SGBD local. Partes do SGBD local podem ser compartilhadas pelos usuários globais. O usuário utiliza uma linguagem de consulta única, SQL, para acessar e integrar os dados de bancos de dados diferentes. O sistema automaticamente localiza os dados, abre conexões para os SGBDs, envia as consultas na linguagem adequada para cada banco de dados e integra os dados das múltiplas fontes. O *Mermaid* suporta alguns níveis de heterogeneidade, entre eles: *hardware*, sistema operacional, conexão a rede, tipos de SGBD e linguagens de acesso, modelo de dados e esquema de dados. O sistema permite a recuperação através dos bancos de dados e atualizações em um único banco de dados. Uma característica interessante deste sistema é que uma transação de leitura pode ver um estado inconsistente do banco de dados,

já que nenhuma atualização local pode ocorrer nos bancos de dados locais durante a execução de uma consulta.

MULTIBASE [TTCB+90, SBD+81] -Xerox Advanced Information Technology

- É um sistema federado fracamente acoplado que oferece a definição de múltiplos esquemas locais e múltiplos esquemas ou visões [SL90]. Esquema locais descrevem os dados disponíveis nos SGBDs locais. Visões descrevem as integrações dos dados dos esquemas locais. Os usuários podem consultar qualquer combinação de esquemas locais ou visões, assim como múltiplos esquemas ou visões podem ser referenciados em uma única consulta. O mecanismo de visão do MULTIBASE também é usado para solucionar incompatibilidades que frequentemente aparecem quando os bancos de dados desenvolvidos e mantidos separadamente são acessados em conjunto. Incompatibilidades incluem diferenças de nomes, estrutura de dados, representações dos dados ou escala, falta de dados e conflito entre os valores dos dados. Quando define uma visão, o administrador do banco de dados aplica o conhecimento das bases de dados locais para determinar que incompatibilidades podem surgir e que regras devem ser utilizadas para solucioná-las. As regras são incluídas nas definições da visão, utilizadas pelo sistema para gerar as respostas as consultas. O MULTIBASE realiza a otimização de consultas tanto a nível global quanto a nível local. A nível global o sistema cria estratégias de modo a minimizar a quantidade de dados movidos entre os nós e maximizar o processamento paralelo que é inerente quando sistemas distribuídos são acessados. A arquitetura do MULTIBASE pode ser vista em [LR82].

SYBASE [TTCB+90, OPSY98] - Sybase, Inc.

- Em 1990, a *Sybase* introduziu o *Open Server*, um produto que estende as possibilidades de distribuição do SYBASE para fontes de dados heterogêneas. O SYBASE é um SGBD federado fracamente acoplado [SL90], que tenta abrir a arquitetura o mais possível, de modo a permitir que qualquer banco de dados, aplicação ou serviço possa ser integrado em sua arquitetura cliente/servidor num ambiente heterogêneo. Nenhum modelo de dados global ou esquema é forçado. Até certo ponto, operações distribuídas podem ser suportadas através de programas de aplicações ou através de RPCs (*Remote Procedure Calls*) entre os servidores SQL. Isto provê um alto grau de autonomia do nó. Nos sistemas tradicionais de bancos de dados centralizados, não é dado aos usuários de aplicações o acesso direto para realizar atualizações no banco de dados, em vez disso, dá-se a possibilidade de comunicação com um programa de aplicação que protege o banco de dados dos usuários. Este método comum pode ser chamado de *application-enforced integrity.* A legalidade de qualquer atualização é determinada principalmente por regras forçadas pelos programas de aplicação. *Application-enforced integrity* é um método falho em sistemas de bancos de dados distribuídos e heterogêneos já que a aplicação pode ser

escrita em departamentos diferentes ou cidades diferentes pelo administrador do banco de dados que está sendo atualizado. Uma alternativa melhor em SGBD distribuídos é forçar a integridade a partir do próprio banco de dados. Neste caso, uma aplicação de um nó remoto se comunica diretamente com um banco de dados que é capaz de decidir quando uma transação viola qualquer regra de integridade. Para isto, são armazenados procedimentos nos banco de dados. O SYBASE suporta o segundo tipo de integridade. Em particular este sistema oferece um método consistente de recebimento de consultas de uma aplicação SYBASE e a transmite para um banco de dados ou aplicação diferente. SYBASE suporta atualizações distribuídas que espalham-se em múltiplas localizações. O protocolo de *two-phase commit* assegura o controle de transações distribuídas sobre múltiplos servidores SQL.

Existe um grande número de sistemas citados na literatura. Aqui nós citamos apenas os mais referenciados. Dentre outros, não orientados a objetos, podemos citar: DQS (*Distributed Query System* - CRAI) [Bel88], o EDDS (*Experimental Distributed Database System* - Universidade de Ulster) [BGL87], o HD-DBMS (*Heterogeneous Distributed DBMS* - UCLA) [Car87], JDDBS (*Japanese Distributed Database System* - Japan *Information Processing Development Center*) [Tak83], NDMS (*Network Data Management System* - CRAI) [Sta85], Preci (Universidade de Aberdeen) [DAT87], o Proteus (Universidades britânicas) [Str84], Scoop (Universidade de Paris e Turin) [Spa82], Sirius-Delta (INRIA - Franca) [Esc84], Unibase (*Institute for Scientific, Technical, and Economic Information*) [Brz84], XNDM (*Experimental Network Data Manager - National Bureau of Standards*) [Kim81], o Heimbigner (Universidade do Colorado) [HM85], Calida (*GTE Research Labs*) [LZ88], etc.

3.6.2 Orientados a Objetos

Os sistemas orientados a objetos não são tão citados na literatura. Estão ainda na fase da infância. Desta forma, o número de projetos orientados a objetos é apresentado em um número bem reduzido em relação aos projetos não orientado a objetos.

Pegasus [SADD+93, ASDK+91] - Hewlett-Packard Laboratories

- É um sistema *multidatabase* desenvolvido pelo departamento de tecnologia de banco de dados nos laboratórios da *Hewlett-Packard*. *Pegasus* fornece o acesso a bancos de dados criados no próprio sistema *Pegasus* (com esquema e dados gerenciado pelo *Pegasus*) como também a bancos de dados externos. Os bancos de dados externos são acessíveis ao *Pegasus*, mas não são controlados diretamente por ele, os quais podem ter diferentes modelos de dados (orientado a objetos, relacional, hierárquico, etc.), diferentes linguagens de acesso, e diferentes estruturas para representar a mesma informação (diferentes visões do mundo). A definição e manipulação dos dados no *Pegasus* é feita através

da linguagem HOSQL (*Heterogeneous Object* SQL). De uma forma geral, o Pegasus define um modelo orientado a objetos para unificar os modelos de dados de sistemas externos. Suporta acessos transparentes a sistemas externos múltiplos, autônomos, heterogêneos e distribuídos através de uma interface uniforme.

ViewSystem [KDN91]- KODIM

- *ViewSystem* é um ambiente orientado a objetos que foi desenvolvido como primeiro protótipo do KODIM (*Knowledge Oriented Distributed Information Management*). Está mais concentrado na dinâmica da integração de bases de informações heterogêneas e autônomas. Fornece uma metodologia completa para a criação de classes virtuais baseadas em um conjunto de construtores.

Operational Integration System (OIS) [GGO90]

- É uma ferramenta de integração geral que oferece à aplicação um ambiente com uma *interface* uniforme de acesso aos dados gerenciados pelos sistemas heterogêneos. Estes sistemas são sistemas de arquivos, SGBDs, sistemas de recuperação de informação, serviços de bancos de dados remotos e aplicações *ad hoc*. Uma importante contribuição deste sistema é a introdução do conceito de mapeamento operacional ao longo da implementação.

Comandos Integration System (CIS) [BNPS88, BNPS89]

- Utilizado para integrar alguns ambientes de aplicação diferentes, incluindo SGBDs relacionais e bancos de dados gráficos. A contribuição deste sistema é a mesma apresentada para o OIS.

Distributed Object Management System (DOMS) [BOHGM92, MHGHB92]

- Desenvolvido pela GTE, é um ambiente orientado a objetos no qual sistemas locais autônomos e heterogêneos podem ser integrados e objetos nativos podem ser implementados. O sistema local não está limitado ao sistema de banco de dados, mas pode ser sistemas convencionais, sistemas hipermídia, programas de aplicações, etc.

UniSQL/M [KGKR+93]

- É um sistema de banco de dados heterogêneos, desenvolvido pela *UniSQL* que permite a integração de sistemas de banco de dados relacionais baseados em SQL e sistemas de bancos de dados objeto-relacionais *UniSQL*/X. Possui como características importantes o tratamento de conflitos de esquema e semânticos.

TSIMMIS (The Standford-IBM Manager of Multiple Information Sources) [CGHI+94,GHIP+95] - IBM Almaden Research Center

- O objetivo do projeto é desenvolver uma ferramenta que facilite a integração real de fontes de dados heterogêneas que podem conter tanto dados estruturais como não estruturais. As disponibilidades das fontes, os conteúdos e os significados dos conteúdos podem mudar freqüentemente. Outra característica deste sistema é que o ambiente de integração requer uma maior participação humana. Num caso extremo, a integração é realizada manualmente pelo usuário. O objetivo do TSIMMIS não é realizar a completa integração de forma automatizada, mas sim oferecer uma ferramenta para auxiliar pessoas no processamento de informações e nas atividades de integração.

3.7 *Middleware* de Bancos de Dados

Existem duas categorias principais de *database middleware* disponíveis: *middleware software* e *gateway*. Enquanto o *middleware software* pode estabelecer uma conexão entre algumas fontes de dados, um *database gateway* estabelece uma conexão ponto a ponto para um banco de dados remoto.

Middleware é um termo genérico para referenciar uma camada de sistema de *software* que tenta resolver o problema da heterogeneidade. O objetivo do *Middleware* é simplificar a interface do usuário oferecendo uma visão uniforme e transparente dos serviços que divergem por serem oferecidos por diversos fornecedores, ou por terem sido desenvolvidos de acordo com protocolos diferentes ou porque estão sendo usados para suportar as necessidades de diferentes aplicações.

Os *Gateways* tornam disponível a comunicação entre dados de fontes heterogêneas, possivelmente originárias de diferentes fornecedores e utilizando a mesma ou talvez plataformas diferentes, protegendo programadores e usuários finais das diferenças nos vários serviços e recursos usados pelas aplicações [RH98, RHS98]. Os *gateways* variam de acordo com o nível de transparência que eles oferecem. Eles podem oferecer pouca ou nenhuma transparência de localização, sendo preciso que os usuários saibam a localização da fonte de dados na rede. Ainda, como *gateway* conecta apenas dois bancos de dados, eles requerem múltiplas e distintas conexões quando uma aplicação requer dados de múltiplas fontes.

Em [RH98] é feito um estudo entre três produtos que oferecem a tecnologia *middleware* para integrar fontes de dados heterogêneas. Os três produtos são o *DataJoiner*, *Transparent Gateway* e *EDA/SQL*[1]. O *Transparent Gateway* é baseado na solução típica da técnica de *gateways* para bancos de dados. Ele provê a ligação

1. *Datajoiner* é marca registrada da IBM. *Transparent Gateway* é marca registrada da ORACLE e o *EDA/SQL* é marca registrada da *Information Builders*.

entre um banco de dados ORACLE para outros bancos de dados de outros fornecedores através da linguagem SQL. Ele estabelece uma ligação ponto-a-ponto com um banco de dados remoto, trabalhando sob o controle de um servidor ORACLE, que é chamado de *Integrating Server* [ORA97].

A solução da *Information Builders, EDA/SQL*, apresenta um *middleware* que é formado de alguns componentes. Uma instalação pode ser formada, por exemplo, de *Relational Gateways* e *Full-Function* e *Hub Servers*. O *Relational Gateway* permite o acesso a bancos de dados relacionais. *Full-Function Servers* combinam alguns serviços como *stored procedures* ou serviços de meta-dados em um único servidor. *Hub Servers* dão o suporte a transparência de localização e distribui as consultas SQL para outros servidores EDA/SQL para processamento sobre as fontes de dados locais. A estrutura do *middleware* requer a instalação de um componente EDA/SQL em cada nó que contém uma fonte de dados para ser integrada [IBI97].

O *DataJoiner* pode ser caracterizado como uma solução *middleware* de banco de dados. O servidor (*DataJoiner Server*) é construído no topo do DB2 (versão 2) e portanto oferece as funcionalidades do DB2 [IBM97, IBM97a].

As diferenças mais importantes entre os três produtos pode ser resumida como segue. O gerenciamento de transação é bem suportado pelo DataJoiner que implementa o protocolo *two-phase commit* (2PC) para processar transações distribuídas. O EDA/SQL suporta um grande número de operações DDL e DML e muitas plataformas e fontes de dados. O DataJoiner emprega uma refinada técnica de otimização de consultas e os benefícios podem ser sentidos através da performance do produto. O DataJoiner apresentou problemas para trabalhar com atributos não indexados, mas mostrou-se imbatível na realização de *selects* e especialmente de *joins*. O processamento de visões também é bem suportado pelo DataJoiner [RHS98].

3.8 Considerações

Como foi visto neste capítulo, recuperar informação de uma coleção de banco de dados independentes não é uma tarefa muito fácil. Os bancos de dados que fazem parte desta coleção geralmente têm esquemas diferentes, são expressos em diferentes modelos de dados, e são gerenciados por diferentes sistemas gerenciadores de bancos de dados, cada um dos quais, possuindo sua própria linguagem de recuperação. Formular e implementar consultas que requerem dados de mais de um banco de dados gera muitos problemas para os usuários. Estes problemas incluem resolver discrepâncias entre os banco de dados, tais como diferenças na representação e conflito de nomes, resolver inconsistência entre cópias da mesma informação armazenada em diferentes bancos de dados e transformar a consulta expressa

na linguagem do usuário em um conjunto de consultas expressas nas diferentes linguagens de recuperação suportadas pelos diferentes SGBDs [DH84].

É importante notar que o *middleware* providencia apenas a interação entre as fontes de dados heterogêneas, através da transparência de localização e de linguagem de consulta. Mas, as diferenças estruturais e representacionais entre os esquemas não são tratadas. Essa tecnologia facilita o trabalho de integração, já que não é necessário a tradução dos esquemas para um modelo comum, necessários tanto no modelo do esquema global quanto no modelo federado. Utilizando esta tecnologia todos os esforços de integração podem ser voltados somente para o aspecto de diferenças representacionais que existem nos objetos relacionados nos diferentes componentes do sistema.

4 MENTAS

4.1 Introdução

MENTAS - *MotorEntwicklungsAssistent*- é um projeto de inovação da *Daimler-Chrysler AG Research and Technology*, que está sendo desenvolvido para enge nheiros mecânicos do desenvolvimento de motores da Mercedes-Benz, possuindo as seguintes motivações:

- incrementar a capacidade de reação ao mercado, bem como o potencial inovador;
- reduzir o tempo de desenvolvimento dos motores e conseqüentemente os custos; e
- paralelizar o trabalho de desenvolvimento.

Hoje, o ambiente de desenvolvimento de motores na Mercedes-Benz é caracterizado pelo isolamento, pelo uso de *softwares* e banco de dados de vários fornecedores. Estes *softwares* são, na maioria, ferramentas de cálculos e simulações, e sistemas CAD, os quais não conseguem entender ou se comunicar uns com os outros. Além disso, os sistemas de bancos de dados infelizmente não têm nenhuma habilidade para relacionar dados de fontes heterogêneas. Essencialmente, cada departamento envolvido no desenvolvimento de um motor constitui uma ilha de informação com suas próprias ferramentas e fontes de dados.

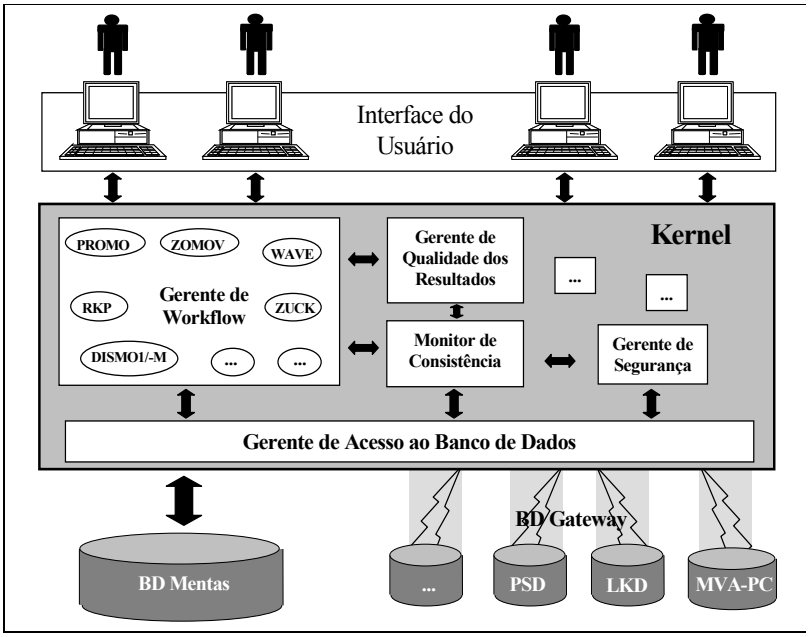
MENTAS tem por objetivo realizar a interconexão das fontes de dados e ferramentas, montar um ambiente de desenvolvimento orientado a engenharia para a rápida concepção e análise comparativa dos motores. Para atingir este objetivo, deve ser oferecido um acesso automático e integrado aos bancos de dados heterogêneos e às várias ferramentas de simulação.

O principal problema encontrado pelos engenheiros neste ambiente completamente heterogêneo diz respeito a habilidade de recuperar as informações das várias fontes dos dados para realizar a comparação e correlacioná-las. Os bancos de dados são encapsulados pelas aplicações, de forma que os engenheiros não têm acesso direto aos dados armazenados. Portanto, eles devem utilizar diferentes *interfaces* oferecidas pelas aplicações para cada banco de dados. Outra limitação, é que tais *interfaces* não possibilitam a criação de consultas *ad hoc*, sendo limitadas a um número de consultas pré-definidas para acessar os bancos de dados locais. Como o projeto e desenvolvimento de um novo motor é um processo

extremamente criativo, os engenheiros sempre vêem-se limitados na sua criatividade porque as *interfaces* não oferecem um método adequado para recuperar informações nem das fontes de dados locais, e muito menos de fontes de dados heterogêneas. Portanto, não é oferecido o suporte para que comparações e correlações entre os dados sejam realizadas.

Na Figura 4.1 podemos ver a arquitetura do projeto MENTAS. No topo da arquitetura encontra-se a GUI que permite aos usuários consultar o sistema. Através desta interface homogênea, os usuários têm acesso às ferramentas e fontes de dados que fazem parte do MENTAS. Os dados produzidos por uma ferramenta podem ser consumidos por outras ferramentas utilizadas no processo de desenvolvimento de motores. Essa produção e consumo de dados entre as ferramentas gera um *workflow* que é controlado por um Sistema de Gerenciamento de *Workflow*. Para garantir a consistência e segurança do sistema são utilizados o monitor de consistência e o gerente de segurança. As fontes de dados são acessadas através da tecnologia *middleware*. No contexto que está inserida esta dissertação, não é levada em consideração a problemática referente à integração das ferramentas no MENTAS. Ao contrário, todas as atenções do trabalho são voltadas para a integração das fontes de dados

Figura 4.1 A Arquitetura Geral do MENTAS.



A visão do usuário em relação aos bancos de dados do MENTAS é que se trata de um esquema global no qual as consultas podem ser formuladas como se todos os dados residissem em um único banco de dados local, mas na verdade, os dados são distribuídos sobre fontes de dados remotas. O interessante da nossa solução é que o usuário pode navegar através dos bancos de dados fazendo comparações, e relacionar informações das várias bases através de uma interface homogênea e simples, e com isso, recebendo informações mais confortável e o mais importante, mais rapidamente que nos ambientes usuais de desenvolvimento.

4.2 Tecnologia Middleware de Banco de Dados

Database middleware traz muitos benefícios na integração de banco de dados heterogêneos, por isso, decidimos empregar esta tecnologia no MENTAS. Antes de decidir por um produto particular, foi feita uma análise crítica, testes e comparações entre os mais populares *middleware* de banco de dados existentes atualmente no mercado. Decidiu-se pelo uso do DataJoiner (IBM), que mostrou-se imbatível no processamento de consultas e especialmente entre operações de *joins*, sendo esta umas das regras cruciais para o uso desta tecnologia no MENTAS [RH98].

Com o DataJoiner [CHKR98, IBM95], os usuários formulam suas consultas numa única versão de SQL, como se todos os dados residissem em um único banco de dados local quando, de fato, alguns ou todos os dados estão distribuídos através de fontes de dados remotas e heterogêneas. Para oferecer a imagem de uma única localização para os dados, DataJoiner implementa a tecnologia do gerenciamento de bancos de dados distribuídos. Prover o suporte a *joins,* uniões e visões distribuídas em conjunto com outros operadores relacionais que são necessários no suporte de consultas complexas. Para realizar estas operações, o DataJoiner através de um otimizador global minimiza o movimento dos dados através da rede. Desta forma, é conseguido um eficiente processamento de consultas *ad hoc*.

Apesar do *middleware* de banco de dados proporcionar a interação entre fontes de dados heterogêneas, as diferenças semânticas entre as fontes de dados devem ser tratadas separadamente. Portanto, devem ser analisados os banco de dados "parcialmente" integrados pelo DataJoiner procurando sempre identificar quais atributos possibilitam a integração entre as bases de dados e quais as possíveis mudanças que são feitas nos valores para que a comparação entre eles nos retorne os valores desejados.

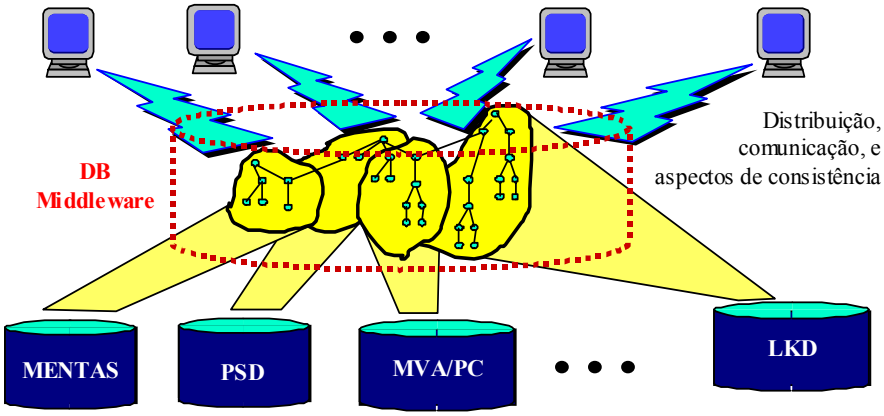
4.3 Integração dos Esquemas

É importante notar que a autonomia local das fontes de dados dos departamentos deve ser mantida após a integração das mesmas. Na prática, estas fontes de dados variam principalmente entre diversas versões de bancos de dados ORACLE e diferentes versões de bancos de dados DB2. Foram analisados os modelos de dados de cada base de dados a fim de identificar os dados cruciais para o MENTAS, reconhecer diferenças semânticas, ambigüidades, sinônimos, homônimos, etc. Nesta fase, encontramos os pontos em comum entre os bancos de dados que são utilizados no momento que uma consulta de um usuário ultrapassa os limites de um banco de dados local para obter informação de outro banco de dados.

Após essa fase, foi construído um esquema global, virtual, o qual só contém dados relevantes para o MENTAS [Rez98, Rez98a, Rez98b]. Essas visões são criadas no topo do banco de dados original pelo administrador do departamento ao qual o banco de dados corresponde, mantendo assim a autonomia local. Por outro lado, informamos ao sistema *middleware* localizado no servidor de banco de dados que existem algumas visões definidas nos bancos de dados remotos em alguns nós da rede. Isto é feito através da definição de *nicknames* no sistema *middleware*.

Assim, uma vez construído o esquema global, virtual, é feito uso do *middleware* para cobrir as diferenças entre os esquemas heterogêneos. Portanto, MENTAS pode formular pedidos a todos os dados residentes em um simples banco de dados local quando, de fato, muitos dos dados são distribuídos sobre fontes de dados remotas heterogêneas. A Figura 4.2 mostra como funciona a interconexão entre os bancos de dados heterogêneos no MENTAS.

Figura 4.2 Integração de Esquemas Heterogêneos como um Esquema Global e Virtual.



4.4 Considerações

MENTAS providencia a integração das várias fontes de dados dos vários departamentos envolvidos no processo de desenvolvimento de motores utilizando a tecnologia *middleware*. Os usuários têm uma visão de um esquema global já que o *middleware* oferece a total transparência de localização dos dados através do uso de *nicknames*. Além da transparência de localização é necessário oferecer aos usuários uma visão homogênea dos esquemas e dados. Isto é feito através do conceito de integração de esquema e é provido pelo módulo gerenciador de consistência presente na arquitetura de acesso aos bancos de dados do MENTAS.

Através da GUI do MENTAS os usuários são guiados no processo de montagem das consultas SQL sem se preocupar com a localização e muito menos com o formato dos dados e estruturas nos vários bancos de dados. Dessa forma, o processo de concepção de motores é otimizado já que disponibilizamos para os engenheiros uma interface homogênea onde eles podem consultar e comparar valores entre os vários bancos de dados.

5 Integração de Bancos de Dados no MENTAS

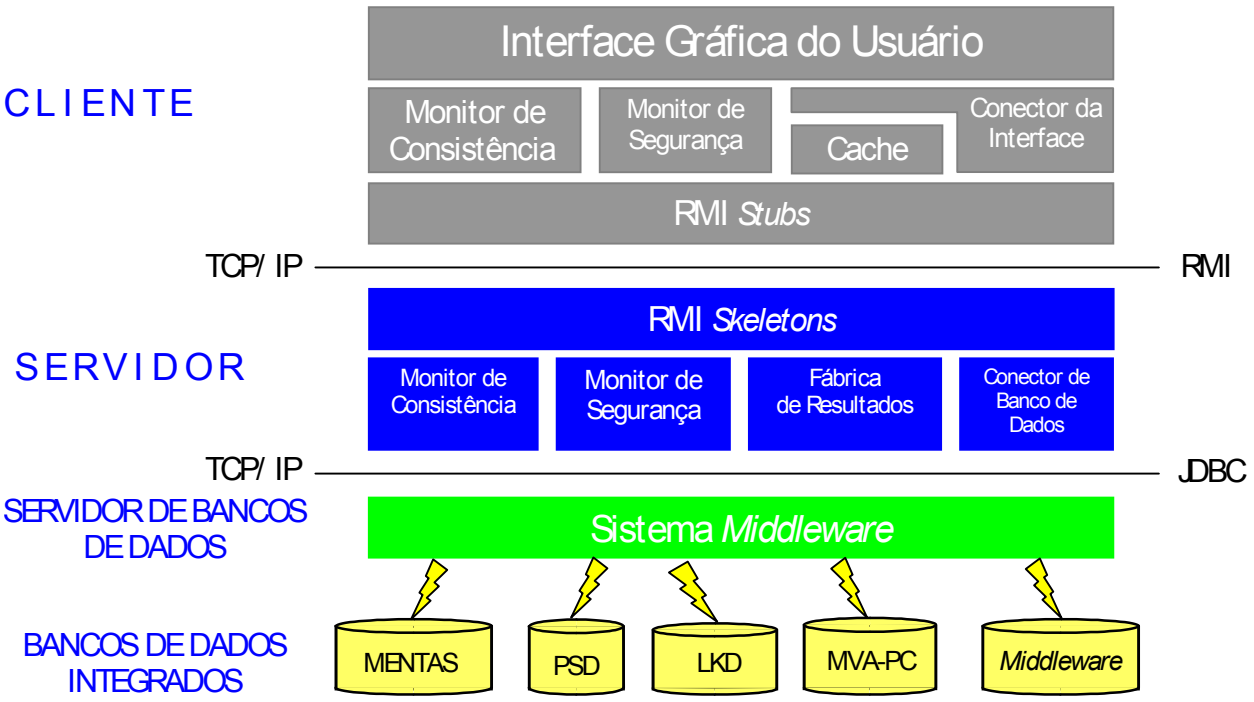
5.1 Introdução

Como vimos, MENTAS deve oferecer para os usuários uma interface amigável e eficiente, de modo que o trabalho dos engenheiros flua com maior rapidez. Ou seja, MENTAS deve prover um ambiente homogêneo para acessar bancos de dados heterogêneos. Este é o propósito deste capítulo: apresentar como está estruturada a arquitetura do MENTAS, como é a ligação de todos os módulos que a compõe e como essas informações chegam até a interface. Terminamos o capítulo apresentando como a interface apresenta-se ao usuário, dando ênfase à interação desta com o monitor de consistência.

5.2 Arquitetura de Acesso aos Banco de Dados

O MENTAS utiliza a arquitetura cliente-servidor para oferecer o acesso aos bancos de dados (Figura 5.1). A arquitetura cliente-servidor foi desenvolvida para tratar novos ambientes de computação, onde computadores pessoais, estações de trabalho, servidores de arquivo, impressoras e outros equipamentos são interconectados através de uma rede de comunicação. O objetivo principal desta arquitetura consiste em definir servidores com funcionalidades específicas de modo que os recursos destes servidores especializados possam ser acessados por vários clientes. Além de aplicar-se a recursos de *hardware*, esta idéia pode ser estendida a recursos de *software*. A decisão por esta arquitetura deu-se principalmente devido a três razões: *escalabilidade*, o *paralelismo* e a *característica de multi-camadas* oferecida por esta arquitetura.

Figura 5.1 Arquitetura de Acesso aos Banco de Dados.



A escalabilidade da arquitetura é um fator de grande importância a ser considerado. A arquitetura cliente-servidor provê a escalabilidade, uma vez que para o sistema crescer é preciso apenas adicionar novo *hardware* ou novos componentes de *software* quando necessário. Estas arquiteturas providenciam por si só o paralelismo: muitos clientes submetem pedidos independentes para o servidor os quais podem ser processados em paralelo. Com isso, existe o aumento de performance conseguindo-se resultados mais rápidos. Ainda, essas arquiteturas têm a característica de multi-camadas. Isso significa que elas podem ser integradas e fazerem parte de outras arquiteturas e novos componentes podem ser adicionados ou retirados.

Arquiteturas cliente-servidor podem ser diferenciadas de acordo com o modo como os a dados são transferidos e como a distribuição das tarefas é organizada: As três mais importantes formas são: *Servidor de Página, Servidor de Objetos e Servidor de Consultas* [HMNR95, RH96, DMFV90, Rez97]. A arquitetura cliente-servidor no MENTAS é um exemplo de uma simples mas efetiva metodologia de servidor de consultas.

Essencialmente, o paradigma da arquitetura servidor de consultas trabalha baseada em contextos. Um contexto normalmente compreende um conjunto de objetos

complexos e pode ser especificado, por exemplo, através de MQL ou SQL/XNF [MPPLS93]. No caso do MENTAS, o contexto pode ser visto como as consultas SQL2 criadas pelos engenheiros através da GUI. A execução de tais consultas acontece no servidor, onde os objetos são armazenados no *buffer* de transferência. Isso significa que com operações como projeções, seleções, e joins, os objetos são transferidos de acordo com a necessidade dos usuários. No MENTAS, a funcionalidade essencial e o poder da linguagem SQL2 são colocadas à disposição dos engenheiros para criarem tais operações. Portanto, o volume de informações a ser transferido para o *cache* do cliente é significativamente reduzido e otimizado.

A implementação dos módulos que compõem esta arquitetura foi totalmente desenvolvida em Java, uma linguagem orientada a objetos desenvolvida pela *Sun Microsystems*. No início do projeto, pensávamos em desenvolver apenas a GUI em Java, já que esta deveria ser independente de plataforma, e a única linguagem de programação que possibilita essa característica atualmente é Java. O núcleo do MENTAS iria ser implementado utilizando C ou C++ que são linguagens cujo desempenho e funcionalidade para este tipo de aplicação são testadas e conhecidas. Entretanto, Java adquiriu de C++ as melhores características e descartou as mais problemáticas e suscetíveis a erros, como por exemplo os ponteiros existentes em C++ os quais são fontes comuns de dor de cabeça para os programadores dessa linguagem. O resultado é que Java é simples, elegante, poderosa e fácil de utilizar [HCF97, Jav97, Roc96]. Dessa forma, devido às facilidades oferecidas por Java e também pelo bom desempenho, decidimos utilizá-la não só no cliente, mas também no servidor, e esta mostrou-se uma boa decisão, já que o tempo de desenvolvimento foi bastante reduzido e a performance do sistema não foi prejudicada.

De acordo com a Figura 5.1, podemos notar que a arquitetura é composta de quatro camadas: O Cliente, o Servidor, o Servidor de Banco de Dados, e os Bancos de Dados Integrados. Apesar de constituírem duas camadas da nossa arquitetura, os bancos de dados integrados e o sistema *middleware*, podemos considerá-los como uma simples camada, já que o sistema *middleware* provê a total transparência de localização para as fontes de dados integradas. Os principais componentes são [RHO+98]:

- A **Interface Gráfica do Usuário** (*Graphic User Interface* - GUI) está na camada superior da arquitetura. Esta é a parte que é percebida pelo usuário e na qual ele interage para formular as consultas em SQL2. Todos os outros componentes da arquitetura são transparentes para o usuário e têm o objetivo de dar suporte à interface;
- O **Monitor de Consistência** está presente tanto no lado do cliente quanto no lado do servidor. Porém, a grande maioria das funcionalidades providas concentra-se na parte do servidor. O monitor de consistência cuida de alguns aspectos relacionados à consistência dos dados tanto num único BD como providencia todos os aspectos necessários para que uma consulta atravesse as

fronteiras de um banco de dados e compare informações entre as bases de dados da federação;

- O ***Cache*** armazena temporariamente os conjuntos de resultados das consultas SQL;
- O **Conector de Interface** intercepta a consulta SQL produzida pela GUI e envia para processamento no servidor. Através de RMI (*Remote Method Invocation* [SUN98a]), objetos distribuídos podem ser implementados simples e elegantemente;
- O **Conector de Banco de Dados** é responsável pelo gerenciamento das conexões providenciando também a comunicação com o sistema *middleware*. Neste nível, foi empregado o JDBC (*Java Database Connectivity* [SUN98, HCF97]) para a comunicação com o sistema *middleware*. O poder e funcionalidade da API (*Application Programming Interface*) do JDBC corresponde ao padrão ISO SQL2. O JDBC emprega o protocolo TCP/IP padrão ISO para a comunicação;
- **Fábrica de Resultados** produz conjuntos de resultados de objetos contendo parte dos resultados das consultas SQL2. Este módulo divide o resultado completo em pequenos pacotes e envia assincronamente para o cliente para ser visualizado pelo usuário através da GUI;
- O **Monitor de Segurança** cuida de todos os aspectos relacionados à segurança do sistema bem como da segurança dos dados no MENTAS, em ambos os lados, Servidor e Cliente. É responsável por gerenciar os usuários, autorizações, acessos e criptografar os dados para transporte entre clientes e o servidor;
- O **Sistema *Middleware*** oferece uma visão uniforme e transparente das divergências dos serviços e recursos dos BDs que são integrados pelo MENTAS;
- Os **Banco de Dados Integrados** são as fontes de dados integradas no MENTAS. No estágio atual do desenvolvimento do MENTAS, todas as fontes de dados são relacionais [Cod70, Cod90]. A integração de outras fontes de dados não relacionais, como por exemplo hierárquico ou em rede, não deve causar problema ao MENTAS, já que o *middleware* empregado também suporta estes tipos de fontes de dados.

Apresentaremos a arquitetura do nível mais baixo (banco de dados integrados) até o nível mais alto (GUI). Nesta última camada, apresentaremos a funcionalidade enfatizando as características de como é feita a **navegação** entre os bancos de dados integrados. Os módulos são apresentados de acordo com os três níveis da nossa arquitetura: *Servidor de Banco de Dados*, *Servidor* e *Cliente*.

5.3 Servidor de Banco de Dados

5.3.1 Banco de Dados Integrados

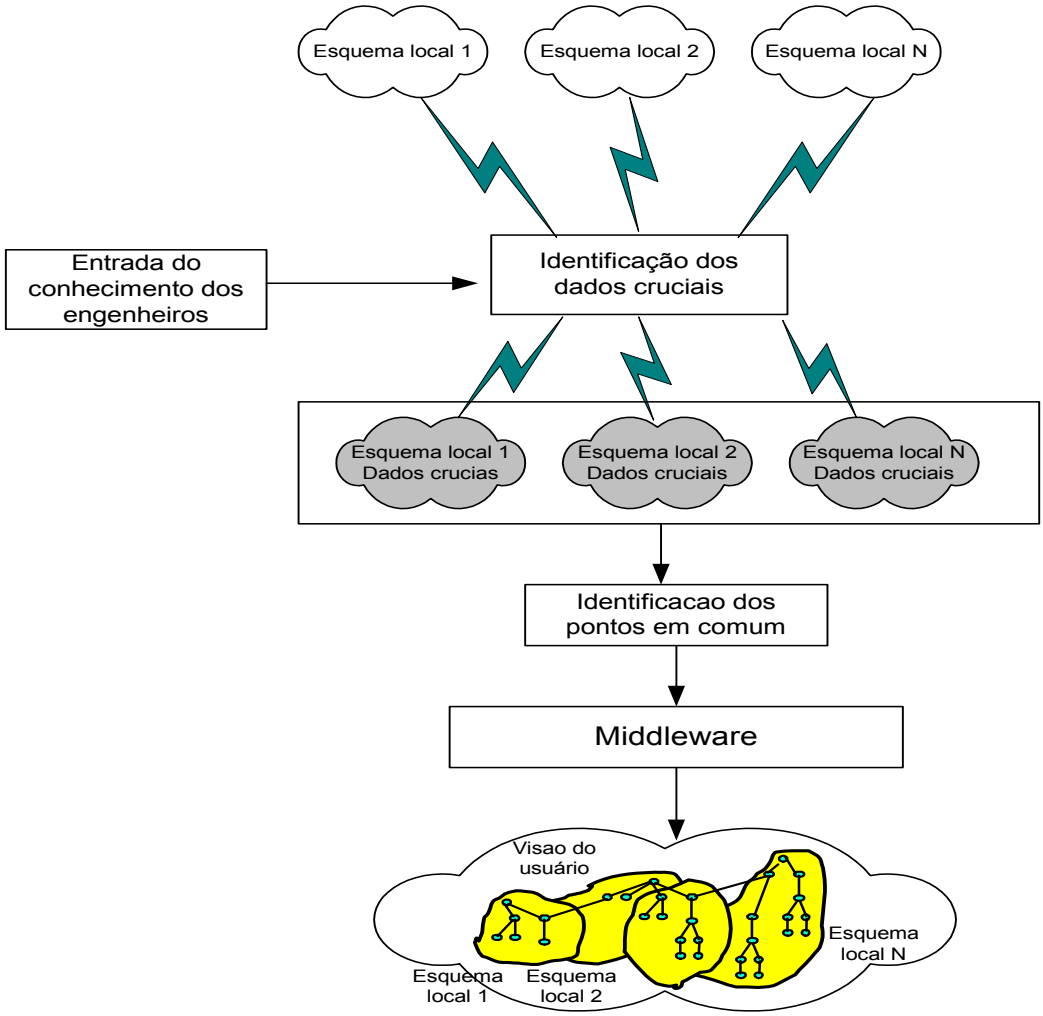
Nesta camada estão todos os bancos de dados integrados pelo MENTAS. Na prática, estas fontes de dados variam principalmente entre diversas versões de bancos de dados ORACLE e diferentes versões de bancos de dados DB2. A Figura 5.2 mostra os passos utilizados durante a integração dos bancos de dados heterogêneos no MENTAS. Algumas fontes de dados remotas possuem informações a respeito de vários componentes de um carro, não apenas de motores (que é o que realmente interessa ao nosso projeto). Após a definição de quais bancos de dados iriam ser integrados pelo MENTAS, foram analisados os modelos de dados de cada base de dados a fim de identificar os dados cruciais para o MENTAS. Essa etapa do projeto foi acompanhada pelos engenheiros mecânicos, de modo a garantir que todas as informações necessárias à construção de um motor estivesse presentes, e mais, deixando os esquemas resultantes o mais enxuto possível.

Uma vez possuindo os esquemas definidos para cada banco de dados participante, analisamos de modo a reconhecer os pontos em comum entre os bancos de dados. Identificamos as diferenças semânticas, os sinônimos, homônimos, etc. O projeto do **monitor de consistência** foi totalmente formulado de acordo com os resultados recolhidos nesta fase do projeto. Através destes pontos em comum, é que está baseada a operação de integração dos bancos de dados no MENTAS.

O próximo passo foi a definição das visões para cada banco de dados, refletindo os esquemas conseguidos em etapa anterior. Todas essas visões são criadas no topo do banco de dados original pelo administrador do banco de dados do departamento correspondente, mantendo assim a autonomia do banco de dados local. Por outro lado, informamos o sistema *middleware* em nosso servidor de banco de dados que existem algumas visões definidas nos bancos de dados remotos em alguns nós da rede. Isto é feito através da definição de *nicknames* no sistema *middleware* [IBM95].

Dessa forma, foi criado um esquema global, virtual, o qual só contém dados relevantes para o MENTAS [Rez98, Rez98a, Rez98b]. Portanto, MENTAS pode formular pedidos a todos os dados residentes em um simples banco de dados local quando, de fato, muitos dos dados são distribuídos sobre fontes de dados remotas heterogêneas.

Figura 5.2 Passos na Integração dos Esquemas Heterogêneos no MENTAS.



5.3.2 Sistema *Middleware*

Como vimos, o MENTAS utiliza a tecnologia *middleware* para conectar as fontes de dados. Segundo [Rym96], *middleware é um software que habilita aplicações interagirem nos nós da rede, escondendo diferenças de comunicação (protocolos), arquitetura de sistemas, sistemas operacionais, banco de dados e outros serviços de aplicações*. Em particular, MENTAS utiliza o Datajoiner da IBM como sistema *middleware*. Esta escolha foi feita depois de um estudo detalhado [RH98, RHS98, HR98a], no qual foi demonstrado a superioridade do DataJoiner sobre os

outros produtos comparados, principalmente em relação a operações de *join* que são de grande importância no ambiente do MENTAS.

Entretanto, faz-se necessário que fique claro que a arquitetura cliente-servidor do MENTAS é independente do sistema *middleware* utilizado. Isto ocorre porque a comunicação entre o servidor e o servidor de banco de dados, onde o *middleware* está localizado, é realizado através do JDBC (*Java Database Connectivity*), um padrão Java para acesso a banco de dados relacionais suportando a funcionalidade e o poder da linguagem SQL2. Dessa forma, MENTAS pode utilizar qualquer sistema *middleware* que ofereça a API do JDBC. Atualmente, praticamente todos os sistemas *middleware* fornecem a API do JDBC.

5.4 Comunicação

Entre Servidor e Servidor de Banco de Dados

No momento, MENTAS integra apenas bancos de dados relacionais. Somando-se a isto o fato de que MENTAS é um produto 100% Java, a comunicação entre o servidor e o servidor de banco de dados é feita utilizando o JDBC [HCF97]. Além deste fato, outra característica que foi levada em consideração é que praticamente todos os fabricantes de bancos de dados oferecem o *driver* JDBC correspondente para os seus produtos. Os tipos de *driver* JDBC podem ser classificados de quatro maneiras [SUN98]. No MENTAS foi empregado o *driver* tipo 3, que é implementado totalmente em Java [RHO+98] e ainda suporta o paralelismo no processamento de consultas. MENTAS ainda suporta o driver tipo 2 devido a alguns casos de inviabilidade no do tipo 3. Maiores detalhes sobre este tópico podem ser encontrados em [RHO+98].

Entre Clientes e Servidor

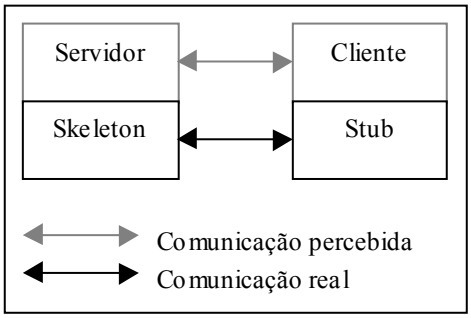
A comunicação entre cliente e servidor é feita através do *Remote Method Invocation* (RMI).O termo RMI é utilizado para descrever o ato de invocar métodos remotos, entre máquinas virtuais *Java*. RMI habilita um método de uma máquina virtual para ser invocado por outra máquina virtual com a facilidade de uma chamada a um método local. Isto faz de RMI uma atraente alternativa em comparação com *sockets*, por exemplo [SUN98a].

Uma das vantagens do uso de RMI é a abstração providenciada para os detalhes de comunicação entre os processos. Parte desta abstração é oferecida por classes especiais chamadas *stubs* e *skeletons*. Um *stub* é o lado cliente que implementa os métodos remotos de um objeto remoto. Sendo **objeto remoto** definido como *um objeto com métodos que podem ser chamados por outras máquinas virtuais Java*.

Skeleton é o lado servidor que aceita uma chamada de método do cliente e dispara a invocação ao método fonte no servidor [Jav97].

A Figura 5.3 mostra o relacionamento entre essas classes especiais e as porções cliente e servidor de uma aplicação RMI.

Figura 5.3 Cliente/Servidor de uma Aplicação RMI.



Quando um cliente invoca um método remoto, existe a ilusão da chamada direta ao método do objeto remoto. Na realidade, a chamada ao método remoto começa como uma chamada a um método local no *Stub*. O *stub* empacota os parâmetros e envia o pedido para o objeto remoto no *skeleton*. O *skeleton* desempacota-os e dispara uma chamada ao método fonte. O *stub* e o *skeleton* também são responsáveis por retornar os resultados.

Em comparação com outros métodos utilizados, como por exemplo CORBA [OHE94,OMG92, OMG95], a performance de RMI é muito boa. Adiciona-se a isso ainda o fato de MENTAS possuir uma arquitetura implementada completamente em Java (isso significa que os objetos distribuídos no cliente e servidor são homogêneos), o poder de CORBA para integrar objetos heterogêneos em ambientes distribuídos seria irrelevante no nosso caso.

5.5 Servidor

5.5.1 Conector de Banco de Dados

As principais tarefas do conector de banco de dados são gerenciar os recursos dos bancos de dados bem como as consultas SQL recebidas dos clientes.

Gerenciamento dos Recursos dos Bancos de Dados

Com a centralização do gerenciamento das conexões neste módulo, é possível aumentar a velocidade do sistema e ao mesmo tempo possuir uma visão global para controlar os recursos do mesmo. Estas vantagens são conseguidas através do compartilhamento de objetos instanciados, no caso, as conexões. Como a mesma conexão é utilizada por várias chamadas, não é gasto tempo para a criação destes objetos. Após o uso, as conexões são devolvidas para o *buffer*, poupando o tempo do *garbage collection*, por sua vez. Este método melhora a performance e minimiza o uso de memória.

Todos os módulos que precisam utilizar uma conexão com o sistema *middleware* devem requisitá-la ao conector de banco de dados. Após a execução da consulta SQL, esta conexão deve ser devolvida ao *buffer* de modo que o conector de banco de dados possa gerenciar o tempo de vida das conexões, bem como criar novas quando necessário. As conexões são utilizadas um número de vezes pré-definido e após isso, são fechadas e outra conexão é aberta. Em [Dav98], é sugerido uma proposta similar utilizando *timestamps*. O pedido da conexão é feito diferenciando entre leitura e escrita. Através dessa diferenciação é possível aumentar o paralelismo no lado do servidor [RHO+98].

Gerenciamento de Consultas

Um dos requisitos da implementação de *drivers* JDBC é que qualquer objeto Java que não esteja sendo utilizado e que consuma recursos do banco de dados deve ser recolhido pelo *garbage collector* liberando imediatamente os recursos. Ainda, devem liberar ou reutilizar os recursos usados pela execução de uma consulta. Infelizmente, existem implementações de *drivers* que não preenchem esses requisitos totalmente. Além disso, o *garbage collector* apresenta algumas desvantagens na prática. Como é sabido, o *garbage collector* em Java é inicializado automaticamente para liberar espaço de memória, quando é preciso. Neste momento, ele libera também os recursos dos bancos de dados. Mas, o contrário não é verdade. Ou seja, quando os recursos dos bancos de dados estão esgotados o *garbage collector* não é acionado. No MENTAS, nós seguimos o método de liberar explicitamente os recursos dos bancos de dados após cada operação de execução, independente do *driver* que estiver sendo utilizado. Dessa forma, não ficamos sujeitos a falhas de implementações de alguns *drivers*, e ainda, possuímos um melhor controle sobre o uso dos recursos dos bancos de dados. Técnicas diferentes são utilizadas pelos vários módulos da nossa arquitetura para liberação desses recursos. O monitor de consistência e o controlador de segurança podem liberar estes recursos localmente[1] logo após o processamento da execução da consulta. No caso do conector

1. Isto significa que o objeto Statement, utilizado para executar uma consulta SQL e obter os resultados retornados por ela são fechados após a execução da consulta.

de banco de dados esta tarefa não é tão simples visto que o processamento dos resultados depende da visualização pelo usuário [RHO+98].

5.5.2 Fábrica de Resultados

Após o processamento de uma consulta SQL, a fábrica de resultados é acionada. Este módulo é o responsável por criar conjuntos de resultados que são mostrados através da GUI aos usuários. Os resultados de uma consulta SQL não são transportados de uma só vez para o cliente e nem muito menos tupla por tupla. Ao contrário, o conjunto do resultado é dividido em pequenas partes e enviados assincronamente para o cliente. Dessa forma, o usuário não é penalizado com o tempo de espera, já que, enquanto é visto o primeiro conjunto dos resultados, os demais são apanhados no servidor e armazenados no cliente. Além de buscar os resultados do servidor e levar para o cliente, é também responsabilidade deste módulo formatar os dados para a apresentação através da GUI aos usuários (janela de apresentação dos resultados - Figura 5.15).

5.5.3 Controlador de Segurança

O monitor de segurança do servidor é responsável por manipular os dados dos usuários, bem como os acessos aos bancos de dados efetuados por esses usuários. É responsabilidade deste módulo também criptografar os dados que são transportados entre o cliente e o servidor. A construção deste módulo é baseada nos mecanismos de segurança providos pelo sistema operacional (UNIX) no lado do servidor e Java [Rüt 99].

5.5.4 Monitor de Consistência

Com o uso da tecnologia *middleware*, nós conseguimos transparência de localização, de sistemas operacionais, de protocolos de rede, etc. Infelizmente, o *middleware* não cuida dos aspectos relacionados à semântica dos dados e nem das heterogeneidades de esquemas utilizadas para representar as informações nos diversos bancos de dados integrados. O principal objetivo do monitor de consistência é justamente resolver os problemas provenientes dessa “falha” dos sistemas *middleware*.

O monitor de consistência visa tão somente proporcionar a GUI um ambiente necessário para a condução da navegação entre os bancos de dados. É função do mo-

nitor de consistência todo o processo de **integração de esquemas**[1] entre os bancos de dados integrados.

Depois do reconhecimento dos atributos que possuem o mesmo significado semântico chamados de pontos de entrada nos bancos de dados, de estabelecer as regras necessárias para que essas diferenças sejam tratadas, o monitor de consistência faz as devidas transformações nos dados de forma a garantir que possa ser executada uma navegação.

Checagem da Possibilidade de Navegação

Outra tarefa importante executada por este módulo é a verificação da possibilidade de ocorrer ou não uma navegação. Esta verificação é efetuada como primeiro passo durante o processo da navegação. Para isso, é enviada ao servidor de banco de dados a consulta formulada pelo usuário através da GUI. Se esta consulta retornar pelo menos um resultado o monitor de consistência sinaliza como possível a navegação e parte para a manipulação dos pontos de entrada. Caso nenhum resultado seja retornado da execução da consulta, é retornado um código para a GUI que o interpreta e envia uma mensagem de aviso ao usuário sinalizando não ser possível a navegação.

Manipulação dos Pontos de Entrada

É necessária a identificação de pontos semelhantes entre os diversos bancos de dados para que possa ser construído o esquema global virtual que é percebido pelo usuário no processo de navegação. Estes são os chamados *pontos de entrada* dos bancos de dados. O monitor de consistência é o responsável por fazer as transformações necessárias entre os pontos de entrada no momento de uma navegação.

Parser

Se todos os bancos de dados fossem modelados seguindo o mesmo padrão, muitos dos problemas de integração de esquemas seriam resolvidos. Acontece que os bancos de dados são modelados de acordo com as necessidades de um conjunto de usuários e mais, projetados por pessoas diferentes que possuem visões diferentes sobre a mesma informação do mundo real. Devido a essas características, encontramos nos bancos de dados do MENTAS diferentes modelagens para uma mesma informação. Um exemplo concreto é o atributo que identifica o motor. No banco de dados Mentas essa informação foi modelada seguindo o padrão da Mercedes-Benz. Por esse padrão um identificador de motor é composto de 3 partes: um tipo, seguido de uma identificação e finalmente seguido por um modelo de construção.

1. Integração de esquemas aqui é o processo de desenvolvimento de um esquema conceitual, livre de duplicações ou heterogeneidades, que integre uma coleção de esquemas locais [Ham94].

Dessa forma, criamos no banco de dados Mentas três atributos para armazenar tal informação. Mas, nos demais bancos de dados esta mesma informação é representada de forma diferente. Em um banco de dados, o tipo é seguido da especificação sem possuir entretanto o modelo de construção. Em outro banco de dados, o tipo é seguido do modelo de construção, sendo que este dois atributos na maioria da vezes (não em todos os casos) são concatenados através de um ponto. Algumas vezes a especificação (no primeiro banco de dados) e o modelo de construção (no segundo banco de dados) é seguida de seqüências de caracteres que não seguem nenhum padrão, sendo totalmente aleatórias. Portanto, mesmo nesses dois bancos de dados em que o identificador do motor é representado por um único atributo não é possível realizar a comparação. Para resolver este problema foi criado um parser que separa um único objeto dos bancos de dados que apresentam outro formato que não o da Mercedes-Benz, no formato recomendado por esta. Para isso, o parser trabalha com o auxílio de regras gramaticais definidas para homogeneizar o identificador do motor transformando todos para um modelo comum no qual possam ser comparados entre si.

Gerenciamento das Tabelas de Mapeamento

O monitor de consistência precisa de um formato padrão para todos os pontos de entrada para que seja possível a ocorrência da navegação. Como a autonomia dos bancos de dados é uma propriedade que deve ser garantida pelo MENTAS após a integração, não é possível que haja modificação dos bancos de dados. Por outro lado, as informações dos usuários formatadas não podem ser comparadas com as informações originais do banco de dados. O monitor de consistência também não pode formatar esses dados cada vez que ocorre uma navegação pois seria altamente ineficiente realizar este procedimento a cada ocorrência de uma navegação. Devido a todas essas características, nós utilizamos tabelas intermediárias chamadas por nós de *Tabelas de Mapeamento*. Essas tabelas são armazenadas no banco de dados do sistema *middleware*, e nela são armazenados todos os pontos de entradas.

As tabelas de mapeamento são geradas automaticamente no momento da primeira inicialização do servidor. Neste momento são buscados das fontes de dados remotas todos os pontos de entrada para serem armazenados nas tabelas de mapeamento. É também acionado o parser para formatar os pontos de entrada. Após essa fase, são armazenados todos os pontos de entrada já com formato comum. Adicionalmente é armazenado nas tabelas de mapeamento o identificador do ponto de entrada no formato original (sem passar pelo parser). Este atributo serve de ponteiro para as tuplas das tabelas originais. Portanto, no momento da navegação, todos os pontos de entrada são tratados e comparados a nível do banco de dados do sistema *middleware*, não sendo necessário o acesso a fontes de dados remotas. Com isso, ganhamos na performance, uma vez que é poupado o tempo de comunicação da rede. E como é armazenado um ponteiro para as tabelas originais, ao final da navegação podem ser recuperadas no banco de dados remoto as tuplas referentes ao dado ponto de entrada. Claro que isso só ocorre se o usuário enviar a consulta para

o processamento. Deste modo, nossas tabelas de mapeamento permitem uma navegação eficiente, já que todos os pontos de entrada se encontram presentes e no mesmo formato.

Como as tabelas originais podem sofrer mudanças, damos a possibilidade ao próprio engenheiro de requisitar que um novo mapeamento seja gerado. O monitor de consistência gerencia informações como dia e hora que foi gerado o último mapeamento e através da GUI esta informação é disponibilizada ao usuário.

Outra característica importante do monitor de consistência é o suporte ao mecanismo de sombra das tabelas de mapeamento. Para cada banco de dados que necessita que uma tabela de mapeamento seja gerada, o monitor de consistência manipula duas tabelas - a versão corrente e a sombra. A tabela de mapeamento corrente dá suporte a todos os clientes do MENTAS durante a navegação dos bancos de dados. Isso significa que estas tabelas devem estar sempre disponíveis. Por outro lado, a requisição da geração de um novo mapeamento pode levar ao bloqueio de todos os usuários. Por isso, quando é requisitado um novo mapeamento, o monitor de consistência dispara *threads* de forma assíncrona gerando o novo mapeamento na tabela sombra. No momento que todos os dados são armazenados na tabela, o monitor de consistência simplesmente troca a informação de qual é a tabela corrente a partir daquele momento, passando a tabela corrente ser a sombra e vice-versa. Dessa forma, a navegação é paralisada apenas no momento da atualização de qual tabela é a corrente, e não em todo o tempo que é gerado o mapeamento completo. Mais uma vez ganhamos na performance do sistema.

5.6 Cliente

5.6.1 Conector de Interface

O Conector de Interface possui duas tarefas principais. A primeira é prover à GUI informações sobre os meta-dados dos bancos de dados integrados e a segunda é manipular as consultas SQL criadas pelo usuário através da GUI.

Gerenciamento das informações dos meta-dados

O conector de interface começa o seu trabalho quando a GUI é inicializada no cliente. Neste momento, ele estabelece uma conexão com o servidor. Através do registro RMI, o cliente recebe uma referência para o objeto servidor que será usado como ponto inicial para todas as outras conexões com o servidor. Através deste objeto, muitos outros objetos contendo informações sobre os meta-dados dos bancos de dados integrados no servidor de bancos de dados são carregados para o cliente. O esquema global das meta-informações é compreendido do nome, atributos e comentários das relações e do nome, comentário e tipo dos atributos.

É criado então uma interface bem definida para que a GUI tenha acesso aos metadados, que são utilizados por esta para fornecer informações de descrição de entidades e atributos aos usuários através do *help*, controlar os operadores na montagem da cláusula de condição, etc.

Gerenciamento de Códigos de Erros e Mensagens

A manipulação de mensagens de erro no MENTAS é feitas através de uma classe própria que contém informação tanto para o usuário final como para o DBA (mensagens detalhadas como exceções SQL2). Os diferentes tipos de falhas que podem ocorrer são caracterizados por um código de erro e a mensagem de erro correspondente. Este código e mensagem são armazenados no banco de dados do sistema *middleware* e carregados nos *properties object* de Java durante a inicialização do cliente.

Gerenciamento de Consultas

No momento que o usuário finaliza a formulação de sua consulta, a GUI produz um *string* da consulta SQL correspondente. A partir de então, o conector de interface cria um objeto (*ExecuteSelect*) que envia assincronamente a consulta para ser processada no servidor de banco de dados. Nesse momento, o conector de interface pára a execução e espera por uma resposta do servidor. Se a consulta for executada com sucesso, o conector de interface passa para a GUI o primeiro conjunto de resultados que é apresentado através da janela de resultados ao usuário. Caso a consulta falhe, um código é retornado à GUI e esta reporta através de mensagens ao usuário.

Após apresentar o primeiro subconjunto dos resultados da consulta, o objeto *ExecuteSelect* busca em paralelo os próximos subconjuntos dos resultados no servidor, armazenando em *cache* no cliente. Dessa forma, quando o usuário requisita a visualização do próximo subconjunto de resultados da sua consulta, ele não precisa ficar esperando por muito tempo, uma vez que estes resultados já estão disponíveis em *cache* para a GUI.

Dependendo da complexidade, o processamento de uma consulta pode levar bastante tempo. Por isso, é dada ao usuário a possibilidade de interromper o processamento da consulta através de uma janela intermediária entre a janela principal e a janela de resultados. Entretanto, interromper a consulta a nível da GUI não significa a interrupção da operação correspondente ao nível do banco de dados (transação). JDBC não oferece nenhum comando que interrompa uma transação ativa, e devido a isso, infelizmente nenhum recurso do banco de dados é liberado. O SGBD sempre executa a consulta até o final.

Em relação às consultas, é ainda de responsabilidade do conector de interface, através do objeto *ExecuteSelect*, armazenar no banco de dados do sistema *midd-*

leware as consultas SQL produzidas pelo usuário (desde que isso seja requisitado pelo mesmo), bem como armazenar em arquivo os resultados das consultas retornados pelo servidor após o processamento das mesmas. Através desta funcionalidade, o usuário não precisa formular consultas que são frequentemente utilizadas, bastando apenas abrir a consulta através da GUI e esta é imediatamente enviada para o servidor para processamento. Assim como as consultas, os resultados também podem ser salvos em arquivo para visualizações futuras pelo usuário. Os usuários podem salvar tanto os resultados parciais apresentados pela janela de resultados no momento da requisição, como também os resultados completos do processamento de uma consulta.

5.6.2 Cache

Este módulo do cliente é gerenciado pelo conector de interface para armazenar temporariamente os resultados das consultas. O método de implementação do *cache* do MENTAS provê funcionalidade suficiente para a primeira versão do nosso sistema, mas existem alguns problemas de capacidade de armazenamento quando uma consulta produz grande quantidade de resultados. No futuro, esperamos mudar a atual funcionalidade do nosso *cache* implementando a metodologia CAOS [Her98].

5.6.3 Monitor de Consistência

O trabalho de ambos os monitores de consistência, tanto o do servidor como o do cliente, é voltado para possibilitar a navegação entre os banco de dados, sendo que, o MCC (Monitor de Consistência do Cliente) se preocupa em formatar os dados fornecidos pelo usuário à GUI e o MCS (Monitor de Consistência do Servidor) em formatar os dados originados das fontes de dados remotas.

A única tarefa do MCC é formatar os dados fornecidos pelo usuário através da GUI. Assim como a parte do servidor, o cliente também conta com um parser, sendo que este é utilizado para formatar os dados informados pelo usuário para os pontos de entrada. Mas, este parser manipula as mesmas regras gramaticais do servidor na hora da formatação. Dessa forma, conseguimos ajustar os valores para um formato padrão, possibilitando que ocorra a navegação.

O perfil dos usuários do MENTAS, em relação a entrada dos dados, varia de acordo com o BD que estes trabalham. Uma preocupação do MENTAS é deixar o usuário trabalhar da maneira mais natural. Dessa forma, não deve ser exigido deste a responsabilidade de saber em qual formato ele deve realizar uma consulta. Muito pelo contrário, ele deve interagir com o sistema como se estivesse trabalhando apenas com o BD usual. Para garantir a transparência das representações dos atributos utilizadas pelos bancos de dados, o monitor de consistência do cliente se en-

carrega de formatar algumas entradas para o usuário. São coisas simples, mas que se não forem cuidadas pode ocasionar uma falha na consulta.

5.7 Funcionalidade da Interface

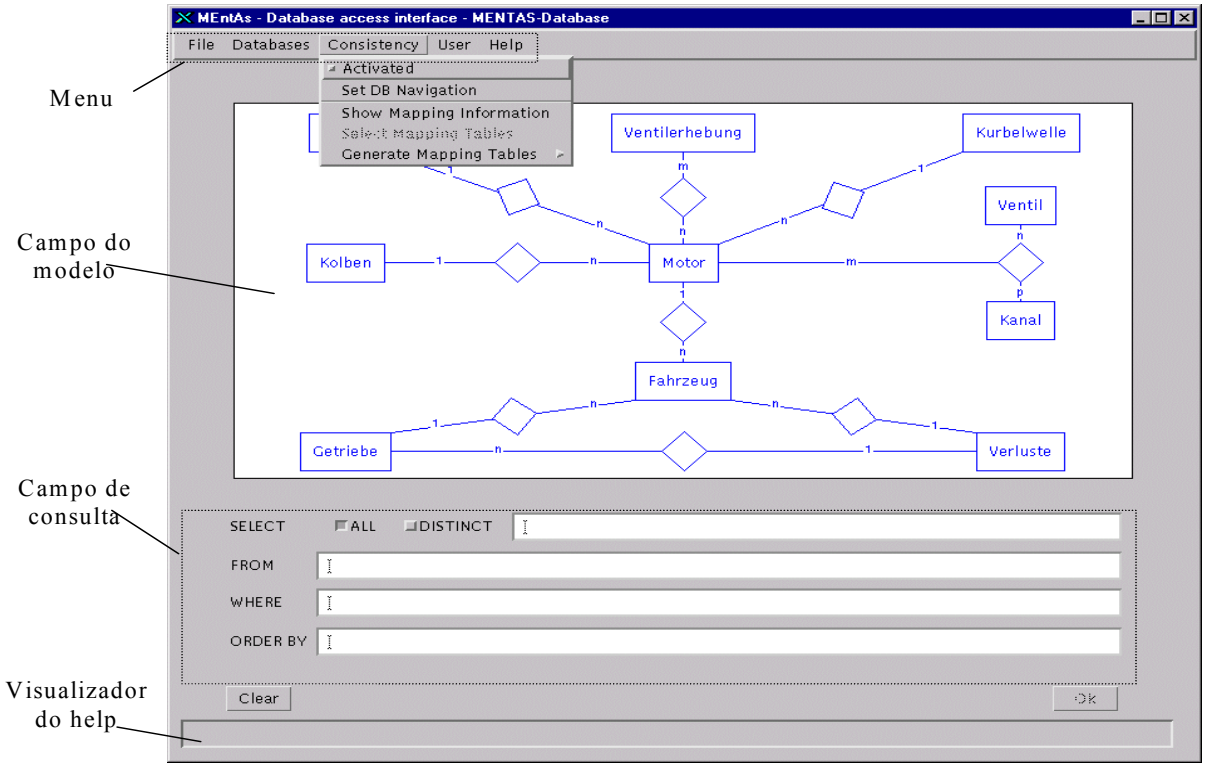
Todos os componentes descritos até o momento trabalham com a única e exclusiva função de oferecer à GUI todo o ambiente necessário para providenciar ao usuário final uma forma rápida, fácil e eficiente de acessar dados tanto num único banco de dados quanto quando a consulta envolver mais de um banco de dados.

Devido ao fato dos usuários estarem distribuídos em departamentos diversos e, assim, possuírem o seu próprio computador, esta GUI foi projetada com a finalidade de possibilitar a sua execução em diferentes plataformas de *hardware* (entre PCs e *workstations* como IBMs, HPs, SUNs, etc.) e *software* (sistemas operacionais como Windows-NT, AIX, HP-UX, Solaris, etc.). Desta forma, a GUI do MENTAS foi programada em *Java*, a qual permite atingir a independência de plataforma desejada [Oli99].

Quando um cliente inicializa uma sessão, são pedidos o login e a senha para que o usuário possa ter o acesso aos bancos de dados integrados pelo MENTAS. Caso o usuário seja autorizado, é apresentada a janela principal da interface que pode ser vista através da Figura 5.4[1].

1. Sendo que a janela principal é inicializada sem a presença de nenhum modelo de dados no campo de modelo.

Figura 5.4 Janela Principal do MENTAS - Interface de Acesso aos Bancos de Dados.



Através da GUI, os usuários podem criar suas consultas SQL para acessar desde um simples banco de dados como também expandir o contexto da consulta para os outros bancos de dados, podendo assim comparar valores e adquirir informações complementares nos outros BDs. A GUI proporciona ao usuário uma visão que todas as fontes de dados não passam de um sistema homogêneo, quando na verdade o acesso é feito a banco de dados remotos heterogêneos. Nesta seção, abordaremos a funcionalidade da interface, não entrando em detalhes de implementação e técnicas utilizadas. Maiores informações podem ser encontradas em [Oli99].

Os dados são apresentados para o usuário através da interface utilizando o modelo Entidade-Relacionamento - Modelo ER - [Che76] (ver Figura 5.4). Através da visualização do Modelo ER de um BD, o usuário pode identificar os atores e cenas que fazem parte do dia-a dia no ambiente de trabalho - como o motor, as válvulas - e através de um clique do *mouse* no Modelo ER apresentado pela GUI, o usuário

pode formular suas consultas SQL, sem que para isso ele tenha a menor noção da linguagem SQL.

A GUI apresenta o Modelo ER da forma mais enxuta possível, apenas entidades e relacionamentos são mostrados diretamente para o usuário. Os atributos foram omitidos visto que algumas entidades possuem mais de cem atributos, o que torna inviável fazer a apresentação completa.

Como podemos observar na Figura 5.4, os principais componentes da janela principal são:

- Menu: apresenta cinco títulos, sendo eles:

  File, onde o usuário pode abrir consultas previamente salvas e sair do sistema.

  Databases, onde o usuário pode escolher qualquer um dos bancos de dados integrados pelo MENTAS. Quando a GUI é iniciada, não apresenta qualquer modelo no campo esquema (ver Figura 5.4), só sendo apresentado para o usuário o Modelo ER após a escolha através deste item de menu.

  Consistency, dá ao usuário a possibilidade de gerenciar alguns ítens do sistema, como: definir se deseja que a navegação ocorra com ou sem checagem de consistência (através do *check box Activated*), definir os pontos de entrada que serão verificados no decorrer da navegação com checagem de consistência, gerar novas tabelas de mapeamento para a navegação entre os BDs, criar as consultas acessando diretamente as tabelas de mapeamento ou simplesmente recebendo informações a respeito dessas tabelas. Este item de menu está totalmente relacionado ao trabalho do **monitor de consistência**.

  User, através deste item, o usuário pode trocar a *password* que dá acesso ao sistema.

  Help, apresenta ao usuário, através de um *browser*, uma documentação completa do sistema em HTML.

- Campo do modelo: mostra ao usuário o Modelo ER do banco de dados corrente. Esse campo é atualizado, mudando o modelo ER apresentado, caso o usuário escolha outro BD a partir do menu *Databases* ou no caso de navegação entre os banco de dados, onde é automaticamente apresentado o novo modelo ER do BD para o qual a consulta do usuário está dirigindo-se ao final da navegação.

- Campo da consulta: composto por quatro campos editáveis, onde o usuário vai visualizando a consulta que está sendo construída através do Modelo ER, e no caso de usuários mais experientes, onde será digitado diretamente a consulta SQL.

- Visualizador de *help*: mostra ao usuário um texto de ajuda *on-line*, dependendo do posicionamento do *mouse* na interface. Assim, para receber uma

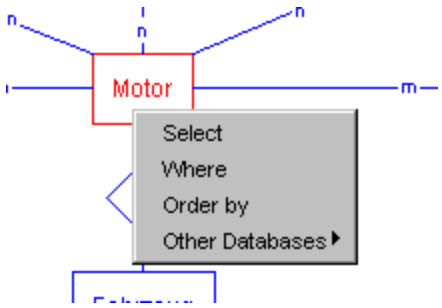
descrição de um objeto presente na interface, tudo que o usuário deve fazer é colocar o *mouse* em cima do objeto e um texto é mostrado neste campo.

- Botões *OK* e *Clear*: Através do botão *OK* o usuário envia a consulta corrente apresentada no campo da consulta para ser processada pelo servidor. O Botão *Clear* limpa a consulta corrente dando início a uma nova sessão.

5.7.1 Formulação de Consultas

O uso da interface é bastante simples. Com apenas um toque no botão esquerdo do *mouse* em cima de qualquer entidade ou de alguns relacionamentos do diagrama entidade-relacionamento, surge um *pop up menu* com as opções que aparecem na Figura 5.5. O usuário pode montar a sua consulta bastando para isso fazer uso das opções que lhe são apresentadas no *pop up menu*, as quais são automaticamente habilitadas ou não, dependendo da situação em que se encontra a consulta do usuário.

Figura 5.5 *Pop up Menu para a Formulação de Consultas.*



De acordo com a figura acima, podemos perceber que existem 4 opções no *pop up menu*, são elas:

Select: especifica a cláusula de projeção para a consulta SQL.

Where: define a cláusula de condição da consulta SQL.

Order by: especifica a ordenação dos resultados da consulta SQL.

Other Databases: habilita a navegação entre os bancos de dados.

A inserção de qualquer um dos atributos de uma entidade ou de um relacionamento na cláusula de projeção da consulta SQL faz com que haja mudança na cor do diagrama, possibilitando assim uma melhor visualização do usuário em relação a interação da sua consulta no banco de dados, representado através do diagrama ER.

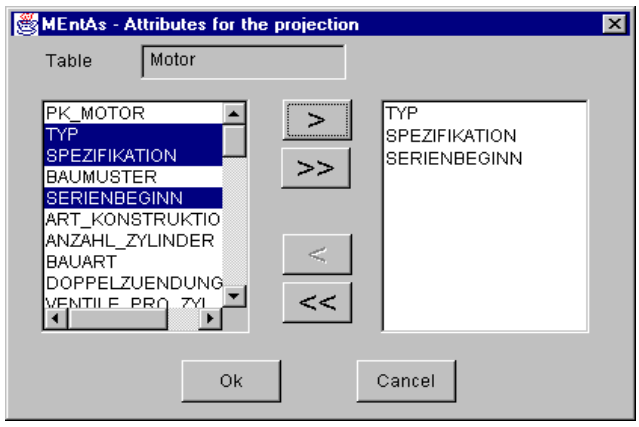
5.7.1.1 Especificando a Cláusula de Projeção

Ao escolher qualquer um dos ítens que compõem o *pop up menu*, o usuário depara-se com novas janelas. Através da escolha do *Select*, ele recebe uma janela contendo todos os atributos da entidade, como pode ser visto na Figura 5.6. Basicamente a janela é composta de duas listas: uma na parte esquerda contendo todos os atributos disponíveis para a projeção e outra no lado direito, contendo os atributos que foram escolhidos pelo usuário a partir da lista da esquerda. Estas janelas são chamadas de janelas de seleção.

A movimentação dos atributos entre as duas listas é feita utilizando os botões ">", ">>", "<", "<<". Antes da movimentação, é necessário que o usuário escolha o atributo que deseja movimentar. Isto é feito posicionando o *mouse* sobre o nome do atributo e clicando o botão esquerdo do *mouse*. Porém, não é dada a possibilidade ao usuário para passar o mesmo atributo mais que uma vez para a lista dos atributos selecionados. Ainda podemos perceber a presença de um *label* na parte superior da janela com o nome da entidade ou relacionamento aos quais os atributos pertecem. Ainda, a barra superior da janela traz a informação sobre a operação que pode-se realizar através dela - *Attributes for the projection.*

Ao selecionar os atributos nessa janela e clicar no botão *OK*, os dois primeiros campos do campo de consulta da janela principal são preenchidos. O botão *Cancel* por sua vez cancela as operações realizadas nessa janela e volta para a janela principal. Através dessa janela são preenchidos os dois primeiros campos do campo de consulta da janela principal, ou seja, o *select* e o *from*.

Figura 5.6 Janela da Projeção.



No momento que o usuário confirma os atributos selecionados por esta janela, é habilitado no *pop up menu* a opção de navegar para os outros bancos de dados (opção *other databases* (Figura 5.5).

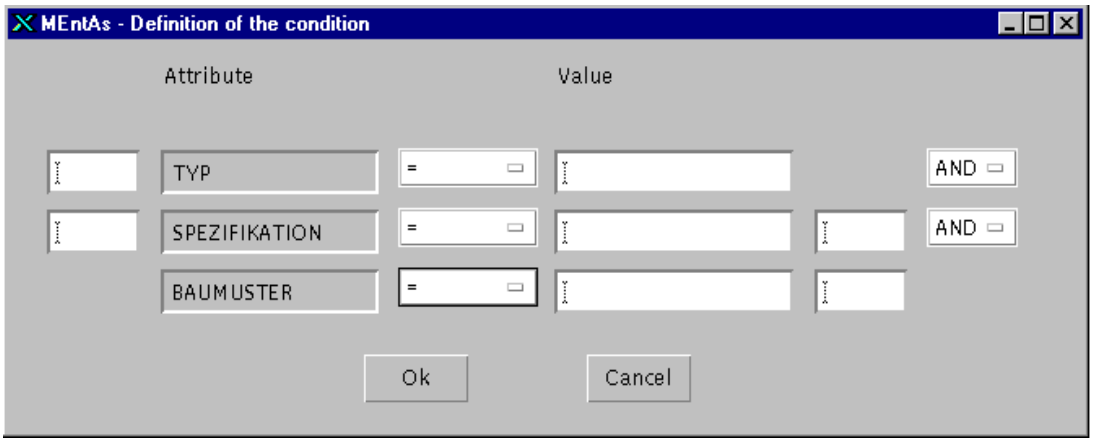
5.7.1.2 Especificando a Cláusula da Condição

A escolha do item *Where* do *pop up menu* origina uma janela idêntica à apresentada anteriormente (Figura 5.6), sendo que serão selecionados nela os atributos que irão preencher a cláusula *where* da consulta do usuário.

A primeira diferença na funcionalidade das janelas é a possibilidade do usuário passar o mesmo atributo mais de uma vez para a lista dos ítens selecionados, possibilitando ao usuário então, construir consultas do tipo: "selecione todos os motores que utilizam diesel **ou** gasolina na combustão".

A segunda diferença está na funcionalidade do botão *OK* que ao contrário da janela anterior, aciona uma nova janela (Figura 5.7) que possibilitará ao usuário estabelecer as condições para os atributos escolhidos nesta primeira fase.

Figura 5.7 Janela de Montagem da Cláusula de Condição.

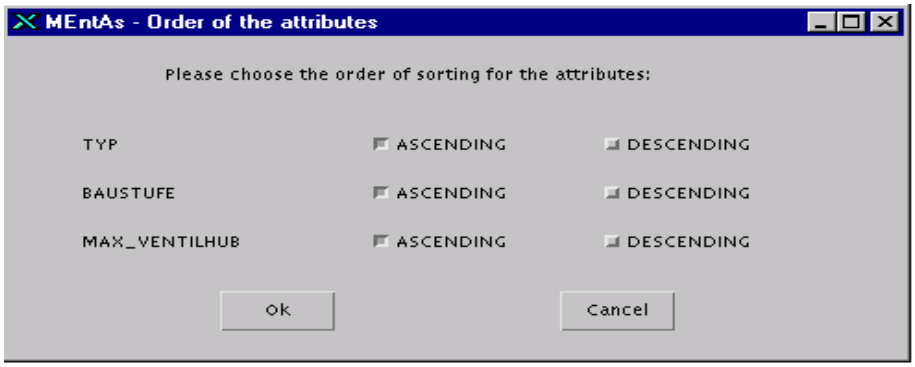


Através desta janela o usuário pode escolher o operador da condição, o valor, aninhar as condições através do uso de parênteses e escolher o operador relacional (*and* ou *or*). São mostrados apenas os operadores de condição - através de uma lista de seleção - compatíveis ao domínio do atributo. Esse controle é feito pela GUI através das meta-informações recebidas pelo conector de interface, evitando que o usuário cometa erros dessa natureza.

5.7.1.3 Especificando a Ordenação dos Resultados

Através da escolha do item *Order by* no *pop up menu*, o usuário tem acesso a uma janela similar à apresentada na Figura 5.6, sendo que na lista da esquerda aparecem apenas os atributos pertencentes à cláusula de projeção. Os atributos selecionados nessa janela irão ordenar os resultados da consulta. Após pressionar o botão *OK* o usuário recebe uma nova janela (Figura 5.8), com dois *check-box* para cada atributo selecionado na janela anterior, classificando-os de forma ascendente ou descendente no resultado. Como nas outras janelas, o botão *OK* leva o usuário à janela principal do MENTAS, atualizando de forma apropriada a consulta SQL.

Figura 5.8 Janela de Ordenação dos Resultados.



5.7.1.4 *Others Databases*

Este é o mais importante tópico da Interface no que diz respeito ao monitor de consistência. Através deste item do *pop up menu* o usuário pode inicializar a navegação entre os bancos de dados. A palavra navegação no MENTAS está implicitamente ligada ao monitor de consistência. É através desta opção no *pop up menu* que será realizada a integração das fontes de dados no MENTAS. Esta tarefa é realizada de forma transparente para o usuário.

No momento em que o usuário especifica uma cláusula de projeção na sua consulta, a opção de *other databases* torna-se ativa (a navegação é habilitada). Note que a opção torna-se ativa para a entidade a qual possui atributos inclusos na projeção e não para todas as entidades. Existem três possibilidades de consultas que são relevantes para o monitor de consistência no momento da navegação, que são:

- consultas sem cláusula de condição
- consultas com cláusula de condição, mas sem pontos de entrada contidos nas condições;

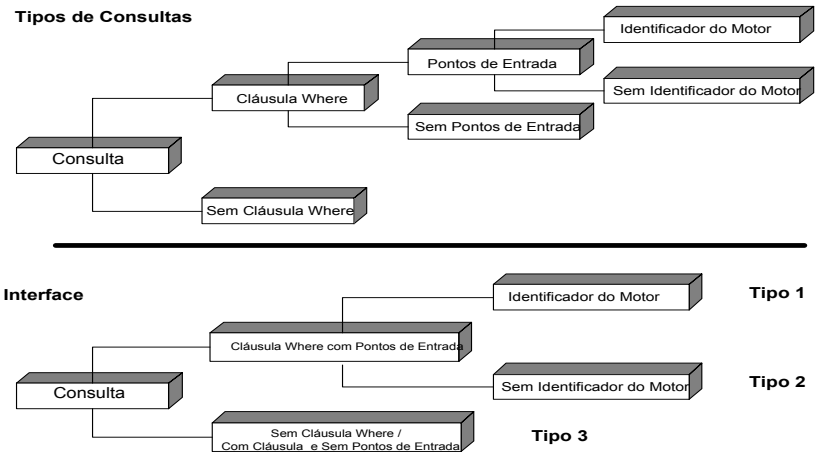
- consultas com cláusula de condição contendo ponto(s) de entrada

Com relação à interface - os detalhes do monitor de consistência são tratados no Capítulo 6 - apenas dois tipos são levados em consideração, que são consultas com pontos de entrada na cláusula de condição e consultas sem pontos de entrada inclusos na cláusula de condição ou consultas sem cláusula de condição (Figura 5.9). Para a interface é transparente o fato de haver um ou mais pontos de entrada presentes na cláusula de condição. As janelas apresentadas são as mesmas.

De acordo com a interface, definimos três tipos de navegação (Figura 5.9):

- Tipo 1: A consulta no momento da navegação possui uma cláusula de condição como também pontos de entrada inclusos nesta cláusula. Mais especificamente, dentre os pontos de entrada está presente o atributo que identifica o motor;
- Tipo 2: Apesar de possuir pontos de entrada na cláusula de condição, o atributo que identifica o motor não está presente;
- Tipo 3: A consulta pode ter duas formas: (i) Não possuir uma cláusula de condição ou (ii) possuir uma cláusula de condição, sendo que esta não contém ponto(s) de entrada.

Figura 5.9 Tipos de Consultas *versus* Interface na Navegação.

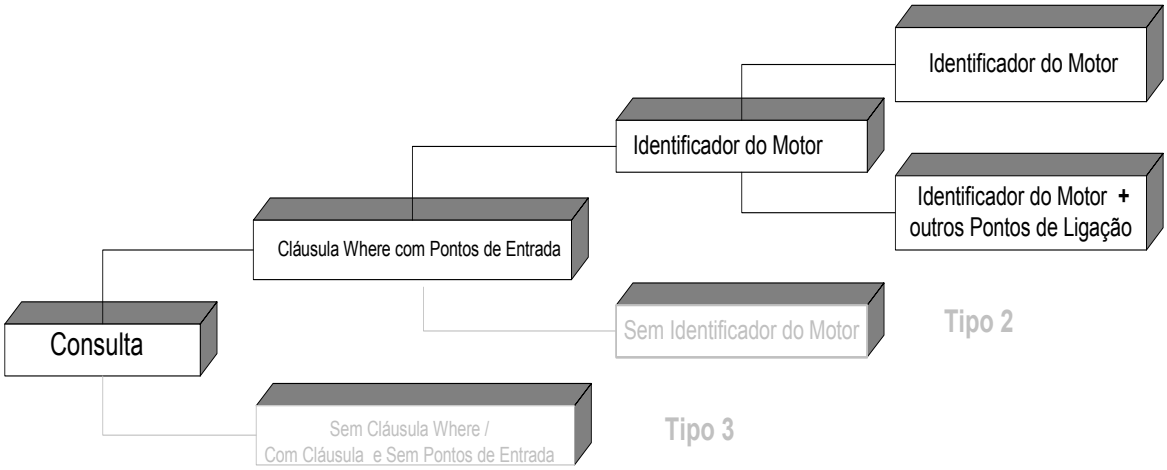


Apresentaremos como se comporta a interface em cada uma das consultas acima, e qual o relacionamento da interface com o monitor de consistência em cada caso.

5.7.1.5 Navegação Tipo 1

Este tipo de navegação pode ser dividido ainda em dois casos distintos, que são apresentados na Figura 5.10. Em ambos os casos será apresentada ao usuário a mesma interface durante a navegação. A distinção desses dois tipos de consultas está apenas no relacionamento da GUI e o monitor de consistência (servidor e cliente).

Figura 5.10 Consulta com Cláusula de Condição Contendo Pontos de Entrada.



Quando existe a presença do atributo que identifica o motor na consulta do usuário (condição), é necessário procedimentos intermediários, antes de apresentar a consulta final ao usuário. Na verdade, a consulta final é montada com o auxílio do usuário. Não é possível automatizar completamente este processo, visto que as diferenças das representações nos atributos que identificam o motor são muitas.

O identificador do motor possui características que o diferenciam dos demais pontos de entrada entre os bancos de dados integrados. Essas características serão exploradas no Capítulo 6. Esse ponto de entrada é representado de formas diferentes em cada banco de dados. Em alguns casos faltam informações para que esses atributos sejam comparados. Por esta razão, não foi possível a completa automatização do processo de mapeamento deste atributo entre os bancos de dados no momento da navegação.

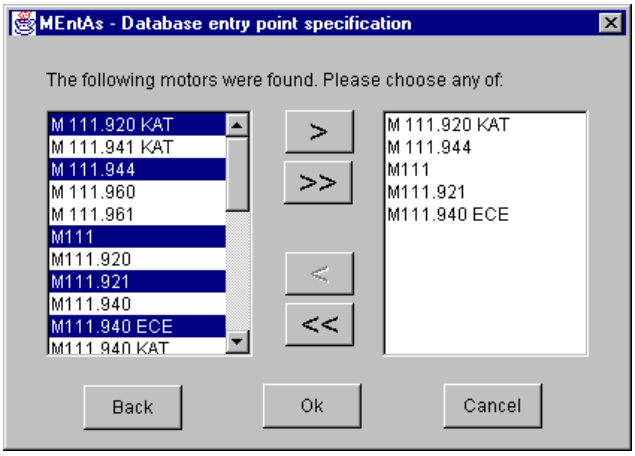
Assim, no momento que o usuário especifica esse ponto de entrada e deseja que a navegação entre os bancos de dados ocorra através dele, a GUI faz uma chamada ao parser do monitor de consistência do cliente que transforma o identificador do

motor para um formato padrão. A partir de então, são montadas as consultas intermediárias pelo monitor de consistência do servidor, para que sejam recuperados os motores no banco de dados para onde está seguindo a navegação que corresponde ao especificado pelo usuário na consulta corrente. O percurso de uma navegação com a presença do identificador do motor será apresentado no Capítulo 6.

Caso seja encontrado pelo menos um resultado para a consulta intermediária, apresentamos ao usuário uma nova janela contendo tais resultados (Figura 5.11). A funcionalidade desta janela é a mesma apresentada pela janela de projeção (Figura 5.6). Sendo que a lista da esquerda apresenta todos os motores que possuem as mesmas definições do motor especificado pelo usuário. Na lista da direita são apresentados os motores que o usuário escolheu como sendo o(s) motor(es) equivalente(s) para o banco de dados para onde está seguindo a navegação.

Quando o usuário clica o botão de *OK*, é montada a consulta final do usuário de acordo com os motores escolhidos, retornando para a janela principal. A janela principal é apresentada com a nova consulta no "campo de consulta" e o modelo de dados do novo banco de dados para onde ocorreu a navegação.

Figura 5.11 Busca da Consulta Intermediária para o Identificador do Motor.



Esta é a primeira tentativa de acharmos o motor especificado pelo usuário no banco de dados posterior[1]. Caso o engenheiro tenha especificado no banco de dados corrente um motor que não possui correspondente direto no banco de dados posterior, ainda oferecemos uma segunda alternativa chamada por nós de *Other Matches*. A idéia do *Other Matches* é procurar no banco de dados posterior os motores que são mais semelhantes ao especificado pelo usuário ao banco de dados corren-

1. Como convenção, adotamos o termo banco de dados posterior para significar o banco de dados para onde está ocorrendo a navegação. Do mesmo modo, o termo banco de dados corrente é o banco de dados inicial de onde parte a navegação.

te. Esta metodologia é uma nova tendência dos métodos de busca e baseia-se na diretiva de que se não é possível encontrar a especificação completa do usuário, deve-se procurar por resultados mais semelhantes ao especificado. Dessa forma, o monitor de consistência monta novas consultas intermediárias que vão buscar no banco de dados posterior os motores mais semelhantes ao especificado pelo usuário.

Other Matches

Com a falha da primeira consulta intermediária, apresentamos ao usuário a janela da Figura 5.12. Nesta janela, informamos que não existe motor correspondente no banco de dados posterior para o especificado no banco de dados corrente e damos a opção para o usuário decidir se deseja realizar a procura mais refinada (através do botão *other matches*). Se o usuário não estiver interessado neste método, pode optar pelo botão *Back* que vai retornar à janela principal.

Se o usuário desejar visualizar os motores mais semelhantes ao especificado, o monitor de consistência monta novas consultas intermediárias que vão buscar no banco de dados posterior tais motores. Isto é feito procurando cada parte que compõe o motor em separado (o identificador do motor no formato padrão). Para facilitar o trabalho do usuário, antes de apresentar uma janela semelhante à Figura 5.11, separamos os resultados de acordo com o atributo que foi correspondente. Após a execução dessas novas consultas intermediárias apresentamos ao usuário a janela da Figura 5.13. Nesta janela são apresentadas a quantidade de motores encontrados para cada parte componente do identificador do motor.

Figura 5.12 Falha da Primeira Consulta Intermediária.

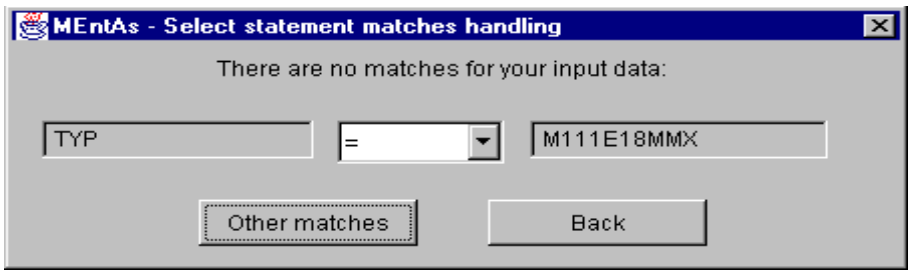
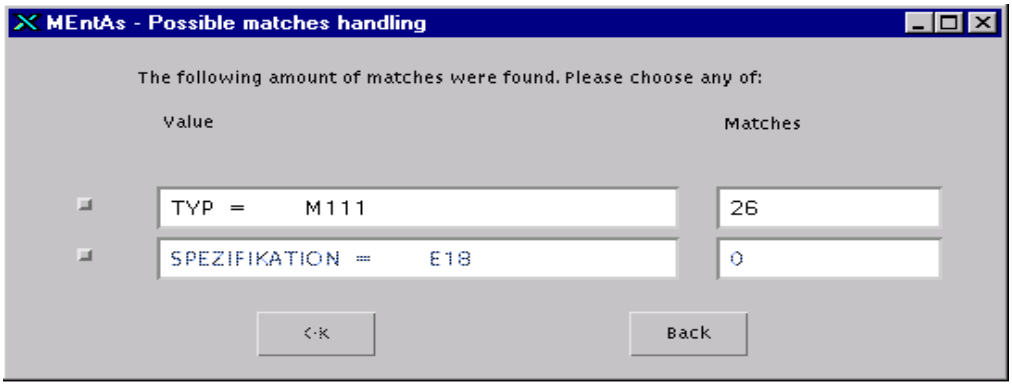


Figura 5.13 Janela de *Other Matches*.



Como podemos notar através da Figura 5.12, o usuário especificou um motor M11E18MMX para o banco de dados corrente e ao navegar para o banco de dados posterior não foi encontrado nenhum motor correspondente. Através da opção de *other matches*, conseguimos encontrar 26 motores que possuem o mesmo tipo do motor igual a M111 e nenhum motor com a especificação E18. A seqüência MMX é ignorada por não fazer parte da gramática por nós definida (Capítulo 6). Escolhendo o botão *Back* o usuário retorna para a janela principal sem efetuar nenhuma mudança. Mas, escolhendo um dos *check box* à esquerda da janela e clicando no botão de *OK* o usuário depara-se com a mesma janela apresentada na Figura 5.11. Na lista da esquerda são apresentados os motores referentes à busca de *other matches* e de acordo com o *check box* escolhido na fase anterior. O botão *OK* (da Figura 5.11) retorna para a janela principal do MENTAS que agora já deve apresentar a consulta final ao usuário junto com o novo campo de modelo. O botão *Back* (da Figura 5.11) volta para a janela Figura 5.13. Caso escolha o *Back,* o usuário tem a possibilidade de clicar em outro *check box* (caso exista) para ver os resultados mais aproximados segundo a perspectiva de outro componente do identificador do motor.

5.7.1.6 Navegação Tipo 2

Esta alternativa de consulta é resolvida sem necessidade de novas janelas na interface. A GUI envia os pontos de entrada presentes na consulta corrente do usuário para o monitor de consistência e recebe deste os mapeamentos adequados que serão acrescidos à consulta final do usuário. O campo de modelo é atualizado para o novo banco de dados corrente e a nova consulta é apresentada ao usuário no campo de consultas.

Por exemplo, suponha que o usuário encontra-se no banco de dados *BD1* e realize a navegação através da entidade *EntidadeBD1* para o banco de dados *BD2*. Para

isso, ele define uma condição através do ponto de entrada desta entidade (*AtributoEntBD1*). Ao partir para a navegação então, ele possui uma consulta do tipo:

SELECT EntidadeBD1.Atributo1, EntidadeBD1.Atributo2, ..., EntidadeBD1.AtributoN
FROM EntidadeBD1
WHERE EntidadeBD1.AtributoEntBD1 <comparação> valor

Que será transformada para a consulta[1]:

SELECT EntidadeBD1.Atributo1, EntidadeBD1.Atributo2, ..., EntidadeBD1.AtributoN, *EntidadeBD2.AtributoEntBD2*
FROM EntidadeBD1, *EntidadeBD2*
WHERE EntidadeBD1.AtributoEntBD1 <comparação> valor ***AND*** *EntidadeBD2.AtributoEntBD2 <comparação> valor*

5.7.1.7 Navegação Tipo 3

Caso o usuário resolva expandir os limites de sua consulta a outro banco de dados, especificando a cláusula de condição sem a presença de pontos de entrada ou simplesmente sem a presença da cláusula de condição, será apresentado ao usuário uma janela semelhante às apresentadas na Figura 5.14. Esta janela varia de acordo com a entidade na qual o usuário está realizando a navegação, pois são apresentados os pontos de entrada para a entidade.

O usuário precisa escolher pelo menos uma das opções apresentadas e a seguir clicar no botão de *OK* para que seja efetuada a navegação para o próximo banco de dados. O botão de *OK* só é habilitado depois de pelo menos uma das opções ser escolhida. Ao contrário, o botão *Cancel* cancela a operação sem efetuar a navegação, voltando para o mesmo ambiente anterior à escolha da opção *other databases*.

Dessa forma, a única diferença da interação da interface com o monitor de consistência é que, se a consulta não possuir uma cláusula de condição, a consulta final é imediatamente montada, sem a necessidade da chamada de nenhum outro módulo. Mas, se a consulta possuir um cláusula de condição, a GUI deve antes de tudo, testar se a consulta corrente montada pelo usuário é válida - por válida, entenda-se que retorna pelo menos uma tupla do banco de dados. Se a consulta não possui uma cláusula de condição definida, não é necessário efetuar tal teste, já que trata-se apenas de uma projeção de uma relação de algum dos bancos de dados, e portanto, sempre retornará resultados[2].

O exemplo a seguir mostra uma navegação na qual não existe uma cláusula de condição na consulta inicial. Considere que o usuário encontra-se no banco de da-

1. As modificações são apresentadas em itálico.
2. Considerando o ambiente do Mentas em que todas as tabelas dos bancos de dados já estão preenchidas.

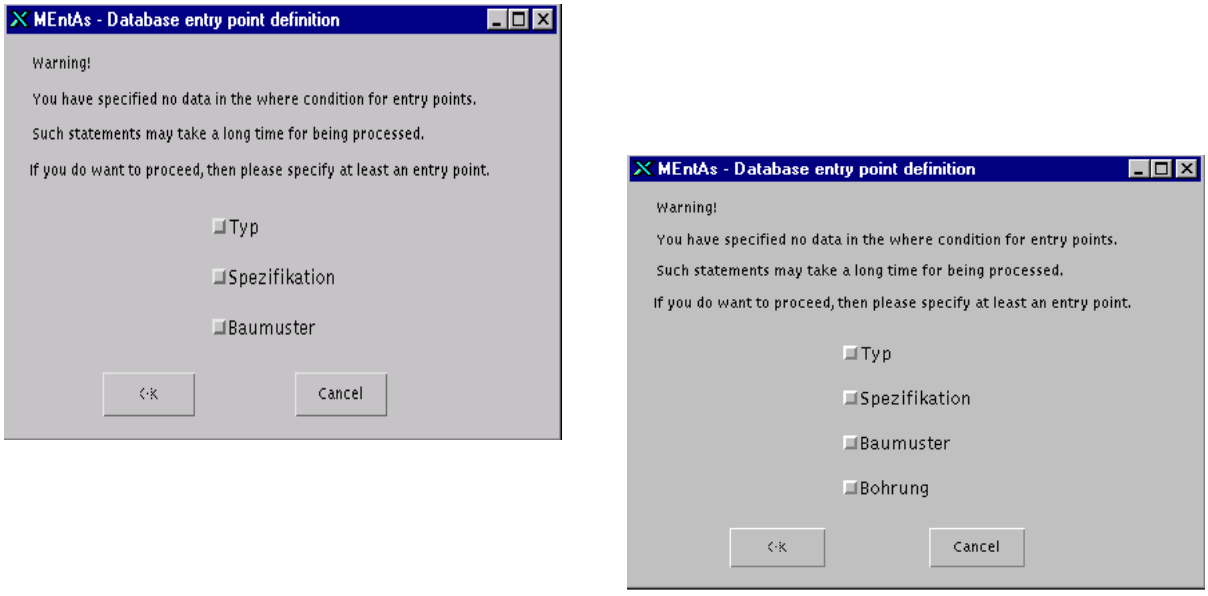
dos *BD1* e deseja expandir sua consulta ao banco de dados *BD2*, tendo como consulta inicial

SELECT EntidadeBD1.Atributo
FROM EntidadeBD1

No momento que o usuário escolhe a opção *other databases* do *pop up menu* da entidade *EntidadeDB1*, é mostrada uma janela semelhante às apresentadas na Figura 5.14 montada pela GUI com os pontos de entrada para a entidade sobre a qual será efetuada a navegação. Supondo que o usuário escolheu dois pontos de entrada (*A* e *B*) dos apresentados nesta janela, e a seguir, escolheu o botão *OK*, é mostrado o novo campo de modelo com o novo diagrama (do banco de dados *BD2*) e com a nova consulta no campo de consulta, que pode ser vista a seguir:

SELECT EntidadeBD1.Atributo, EntidadeBD2.Atributo
FROM EntidadeBD1, EntidadeBD2
WHERE EntidadeBD1.A = EntidadeBD2.A **AND** EntidadeBD1.B = EntidadeBD2.B

Figura 5.14 Janela de Especificação de Atributos.



5.7.2 Janela de Resultados

Como foi explicado anteriormente, após cada passo da montagem da consulta e ao final de uma navegação, o usuário retorna à janela principal do MENTAS. Assim que a consulta SQL atinge o ponto em que é possível enviá-la ao servidor de bancos de dados, o botão *OK* da janela principal é habilitado. Uma vez selecionando o botão *OK*, a GUI passa a consulta ao conector de interface, o qual cuida de todos os passos requeridos para a execução da consulta. A GUI simplesmente recebe o

resultado da execução da consulta e apresenta ao usuário através da janela de resultados (Figura 5.15).

Esta janela é composta de quatro componentes:

- Menu: apresenta três títulos, *File*, *Options* e *Help*.

  File: através deste item de menu o usuário pode salvar consultas, salvar os resultados correntes apresentados pela interface e salvar o resultado completo da consulta.

  Options: Através desse item de menu é possível receber informação das tabelas de mapeamento, da mesma forma que na janela principal.

  Help: apresenta ao usuário, através de um *browser*, uma documentação completa do sistema em HTML.

- Campo da consulta: mostra ao usuário a consulta que foi executada pelo servidor de banco de dados do MENTAS.

- Campo dos Resultados: É dividido em duas partes, a superior apresenta o nome dos atributos e das entidades e a segunda, apresenta os resultados no formato de uma tabela.

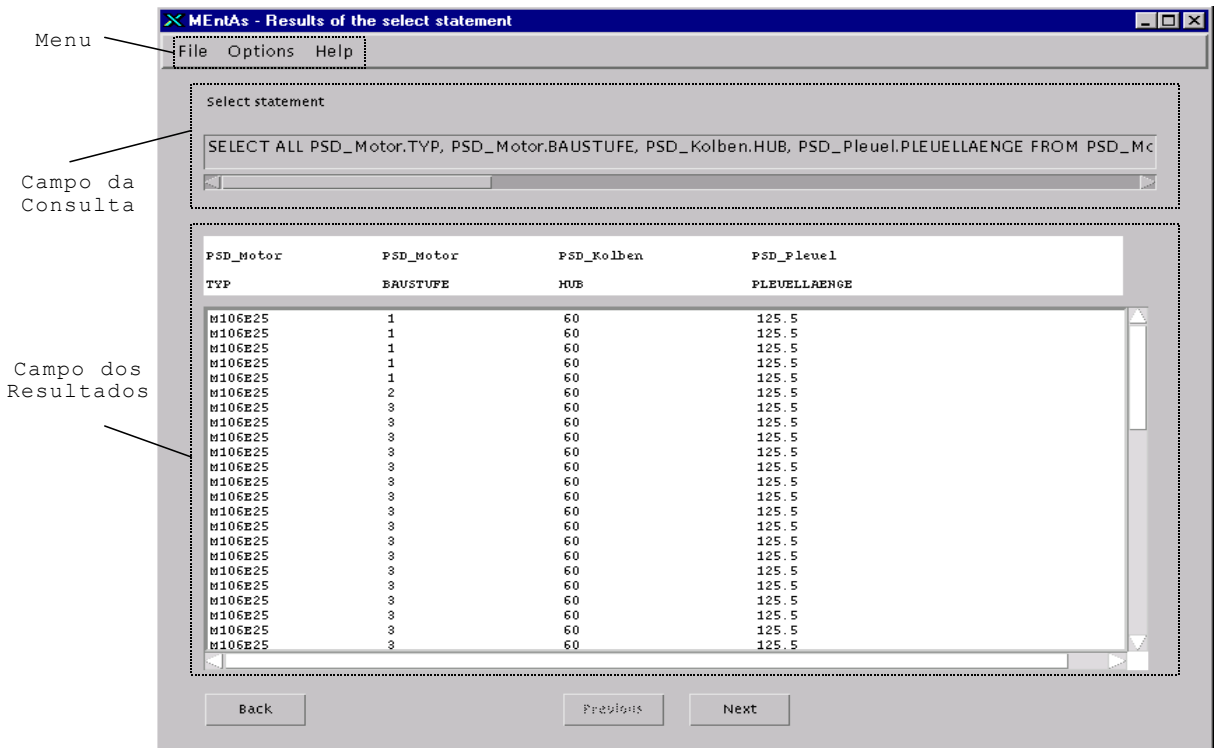
- Botões:

  Back: Através do botão *Back* o usuário retorna à janela principal do MENTAS.

  Next: Apresenta ao usuário o próximo conjunto de resultados armazenados no *cache*.

  Previous: Apresenta ao usuário o conjunto de resultados anterior.

Figura 5.15 Janela de Apresentação dos Resultados.



5.7.3 Menu *Consistency e o Monitor de Consistência*

O menu *Consistency* faz parte da janela principal e é composto de cinco ítens (Figura 5.4): *Check box Activated, Set DB Navigation, Show Mapping Information, Select Mapping Tables, Generate Mapping Tables*. Os dois primeiros ítens estão relacionados à verificação de consistência durante a navegação. Os três últimos relacionam-se à manipulação das tabelas de mapeamento.

5.7.3.1 *Other Databases versus* Menu *Consistency*

As navegações do Tipo 1 (envolvendo o identificador do motor) podem ser realizadas com ou sem checagem de consistência. Este fator não influencia o ambiente de navegação com relação às janelas apresentadas. Por este motivo não foi feita referência à checagem de consistência na seção anterior.

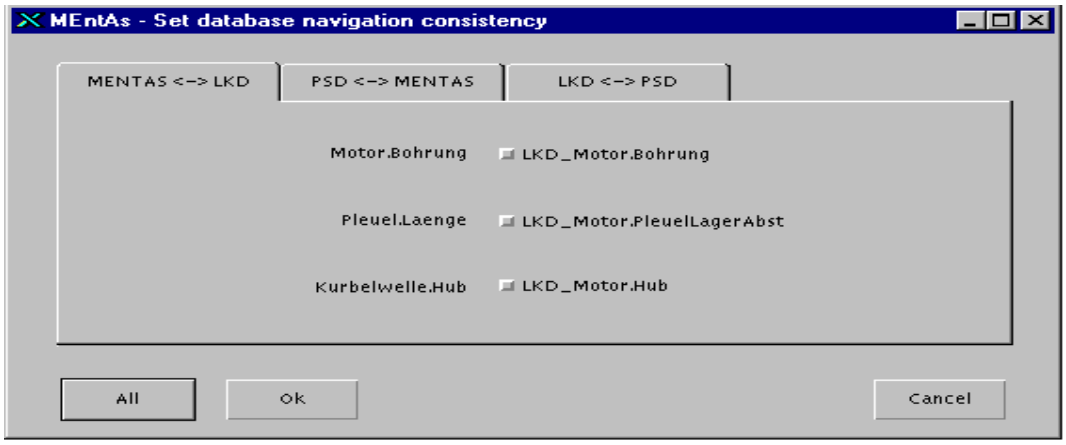
Como já mencionamos, os identificadores de motor são modelados de formas diferentes nos vários bancos de dados. Devido a essas diferenças, precisamos do auxílio do usuário para fazermos o mapeamento de valores entre os bancos de dados.

Através do *menu consistency* o usuário tem a possibilidade de deixar um pouco da responsabilidade sobre o MENTAS. Isto ocorre quando ele deseja realizar a navegação através dos requisitos de verificação de consistência.

Entretanto, quando o usuário deseja definir por quais pontos de entrada vai ocorrer a navegação, surge uma nova janela (Figura 5.16). Esta janela possui três "abas" representando as possíveis formas de navegação entre os pares de banco de dados. Em cada aba são apresentados os pontos de entrada para o par de banco de dados, onde o usuário pode escolher os mais importantes no seu ponto de vista. Esta informação é armazenada no banco de dados do sistema *middleware*, junto com outras informações do usuário. Esta janela será apresentada na primeira vez que o usuário marcar o *check box* do menu *consistency* (*Activated*), ou então através do sub-item de menu *Set DB Navigation*, caso ele já tenha definido em algum momento a verificação de consistência.

Pela janela abaixo notamos ainda a presença dos botões *All*, *OK* e *Cancel*. O primeiro marca todos os pontos de entrada de todas as abas da janela. Neste caso, uma nova janela de aviso é mostrada ao usuário, alertando-o que o processo de navegação ficará mais lento que o usual. O botão *OK* envia as opções escolhidas pelo usuário para serem salvas no banco de dados[1] e o botão *Cancel* simplesmente ignora qualquer operação sobre a janela.

Figura 5.16 Janela de Especificação dos Pontos de Entrada para Checagem de Consistência.

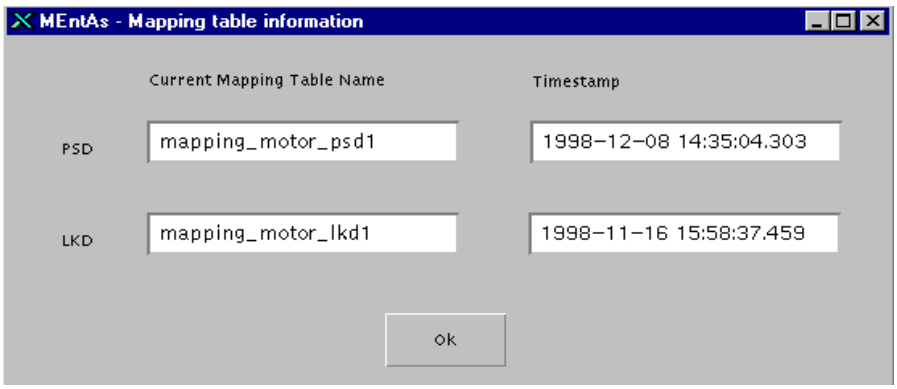


1. A GUI envia as opções marcadas pelo usuário e o monitor de consistência salva as opções no banco de dados.

5.7.3.2 *Show Mapping Information*

A escolha deste item de menu pelo usuário, faz com que a GUI envie um pedido ao monitor de consistência. Este acessa o servidor de banco de dados e recupera informações sobre as tabelas de mapeamento que são enviadas de volta para a GUI montar a janela da Figura 5.17.

Figura 5.17 Janela de Informações sobre as Tabelas de Mapeamento.



5.7.3.3 *Select Mapping Tables*

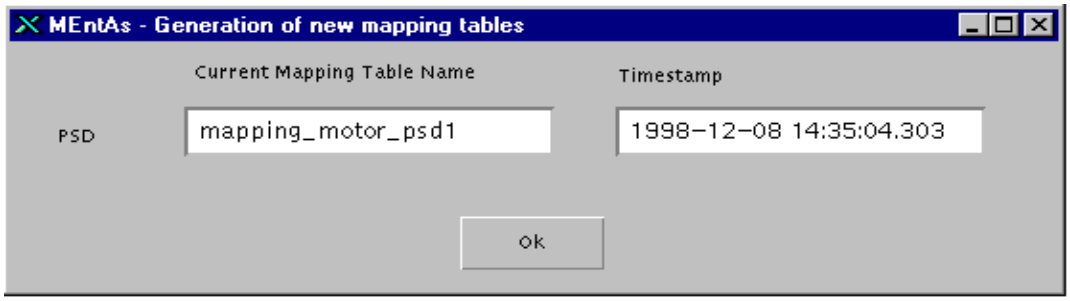
Este item de menu aciona uma janela semelhante à Figura 5.6, sendo sua funcionalidade semelhante à montagem da cláusula de condição. Na verdade, o que o usuário faz através deste item é montar a cláusula de condição da sua consulta através dos valores já formatados presentes nas tabelas de mapeamento. Esta facilidade foi criada para ajudar o engenheiro a encontrar o motor correto para o banco de dados local. Com isso, é oferecida maior segurança na montagem das consultas locais, já que esta deve estar correta para que haja navegação.

5.7.3.4 *Generate Mapping Tables*

Se o usuário escolher este último item, a GUI envia um pedido ao monitor de consistência para que seja gerada uma nova tabela de mapeamento. É enviado em conjunto, o nome do banco de dados escolhido pelo usuário (através do sub-item de menu que apresenta todos os bancos de dados para o qual é possível gerar uma nova tabela de mapeamento). Neste momento, o monitor de consistência dispara *threads* assíncronos para gerar a nova tabela de mapeamento, de forma que o usuário pode continuar o trabalho sem que seja necessário a espera pela resposta. Assim que o monitor de consistência termina o trabalho, ele envia para a GUI a informação que o novo mapeamento já está pronto juntamente com as informações sobre a nova tabela. Estas informações são apresentadas ao usuário através da janela da Figura 5.18. Esta janela tem função apenas informativa, trazendo o

nome do banco de dados para o qual foi feito o mapeamento (nesse caso PSD), o nome da tabela no banco de dados que possui o mapeamento corrente (devido a técnica de tabela corrente e tabela sombra utilizada pelo monitor de consistência) e a data e hora que o novo mapeamento foi gerado.

Figura 5.18 Informação ao Final da Geração de um Novo Mapeamento.



5.8 Considerações

A interface apresentada pelo MENTAS é bastante fácil de usar. Isto possibilita que o usuário sinta-se completamente à vontade no momento da criação das suas consultas SQL. O mais importante é que os usuários são capazes de criarem consultas SQL complexas sem a necessidade do conhecimento dessa linguagem. A funcionalidade completa da interface bem como os aspectos relacionados à implementação da mesma podem ser encontrados em [HR98a, Oli99].

No momento que dois bancos de dados são envolvidos na mesma consulta tanto a montagem da consulta quanto as transformações necessárias nos dados são realizadas sem o conhecimento do usuário. Este precisa apenas informar que deseja acessar um segundo banco de dados e por qual entidade e atributo deseja que seja feita a integração.

A GUI consegue oferecer um ambiente deste nível aos usuários devido às relações mantidas com os demais componentes da arquitetura do MENTAS. Neste capítulo nós demos atenção especial à interação da GUI com o monitor de consistência por ser esse o tópico de interesse desta dissertação.

6 Monitor de Consistência

6.1 Introdução

Na modelagem de um banco de dados um mesmo conceito pode ser representado de diferentes formas. Essas diferenças podem ser em nomes, estruturas, tipos de dados, etc. Devido a essas diferenças é impossível manipular dados em diferentes bancos de dados com uma linguagem como SQL que foi projetada para ser usada em um banco de dados homogêneo. Como podemos notar, a integração de banco de dados heterogêneos não é uma tarefa fácil. Através desta integração, deve-se oferecer para os usuários um ambiente em que ele possa formular consultas únicas a uma federação de N bancos de dados em vez de uma seqüência de N consultas, uma para cada banco de dados.

Para o usuário é transparente o fato dos bancos de dados estarem distribuídos sobre fontes de dados remotas. MENTAS proporciona a este uma visão de um esquema global no qual as consultas podem ser formuladas como se todos os dados residissem em um único banco de dados local. Entretanto, a integração das fontes de dados acontece no MENTAS apenas quando o usuário navega por entre os bancos de dados. Dessa forma, garantimos ao usuário a total independência de acesso aos dados de um único banco de dados, e ainda, caso deseje, sua consulta pode atravessar as fronteiras do banco de dados corrente.

Ao nível de integração das fontes de dados, a principal tarefa do Monitor de Consistência é oferecer à GUI todo o ambiente necessário para que o usuário consiga ultrapassar os limites do banco de dados corrente no qual está realizando uma consulta, estendendo-a a outras fontes da federação. Isto inclue fazer o mapeamento entre os diversos atributos correspondentes nos bancos de dados, além de gerenciar a entrada de dados no sistema. Ou seja, é tarefa do monitor de consistência resolver os problemas existentes ao **integrar os bancos de dados heterogêneos**.

O trabalho de ambos os monitores de consistência, tanto o do servidor como o do cliente, é voltado para possibilitar a **navegação** entre os bancos de dados. O MCC (Monitor de Consistência do Cliente) se preocupa em formatar os dados fornecidos pelo usuário à GUI e o MCS (Monitor de Consistência do Servidor) em formatar os dados originados das fontes de dados remotas. Além disto, este último cuida da verificação de consistência entre as fontes de dados no momento da navegação.

Iniciamos o capítulo com a definição do que vem a ser a navegação no MENTAS (seção 6.2). A seguir, apresentamos os passos utilizados pelo monitor de consistência para integrar as fontes de dados: na seção 6.3 apresentamos os pontos em comum que foram identificados no MENTAS, na seção 6.4 apresentamos quais os conflitos existentes nestes pontos de entrada e na seção 6.5 são mostradas as regras que devem ser seguidas para resolver os conflitos entre os bancos de dados. Partimos então para apresentar como foi solucionado o problema da integração dos bancos de dados apresentando as soluções para as heterogeneidades dos pontos em comum: na seção 6.6 apresentamos o monitor de consistência do servidor e na seção 6.7 apresentamos o monitor de consistência do cliente. Na seção 6.8 apresentamos um exemplo de uma navegação no MENTAS. Na seção 6.9 é feito um paralelo entre consistência e performance no MENTAS. Finalmente, na seção 6.10, um breve resumo é apresentado.

6.2 Navegação entre os bancos de dados

O termo navegação no ambiente MENTAS está implicitamente relacionado a integração das fontes de dados. É através desta funcionalidade que o usuário pode comparar e integrar dados que estão em fontes de dados distintas. Realizar uma navegação no MENTAS é muito simples. Primeiro o usuário deve formular uma consulta ao banco de dados no qual ele está trabalhando e depois escolher um segundo banco de dados para o qual sua consulta será expandida, e escolher sobre quais pontos de entrada entre estes bancos de dados será feita a ligação entre os bancos de dados. O MENTAS encarrega-se de todo o processo de conversão entre os pontos comuns que ligam este dois bancos de dados e expande a consulta inicial do usuário ao nível do próximo banco de dados.

A navegação entre os bancos de dados no MENTAS é unidirecional, significando que cada banco de dados só pode ser visitado uma única vez durante uma navegação completa. Dizemos que uma navegação está completa quando todos os bancos de dados já foram visitados. Como um dos objetivos do MENTAS é proporcionar uma interface amigável e homogênea ao usuário final, não possibilitamos ao usuário a volta a um banco de dados que já foi visitado. Isso deixaria o usuário confuso podendo atrapalhar o processo de montagem da consulta. É preciso que fique claro que em termos de implementação é completamente possível apresentar esta funcionalidade ao usuário.

Como MENTAS possui três bancos de dados, é possível realizar a navegação de seis formas diferentes, apresentadas na Tabela 6.1.

Tabela 6.1 Possíveis Navegações no MENTAS.

BD Inicial	*BD Intermediário*	*BD Final*
PSD	LKD	MENTAS
PSD	MENTAS	LKD
LKD	PSD	MENTAS
LKD	MENTAS	PSD
MENTAS	PSD	LKD
MENTAS	LKD	PSD

Note porém, que a navegação é feita entre pares de bancos de dados, como apresentado na Tabela 6.2. Através desses pares é que torna-se possível realizar uma navegação completa. Por exemplo, a primeira navegação completa apresentada na Tabela 6.1 (PSD[1] - LKD - Mentas), ocorre da seguinte forma: inicialmente o banco de dados corrente é o PSD e o banco de dados posterior é o LKD. Uma vez a consulta do usuário seja estendida do domínio do PSD para o LKD, este último passa a ser o banco de dados corrente. O outro ciclo da navegação ocorre com a extensão da consulta para o BD Mentas, onde o banco de dados corrente é o LKD. Ao final deste outro ciclo, o Mentas passa a ser o BD corrente e, como não existe nenhum banco de dados que não tenha sido visitado, dizemos que ocorreu uma navegação completa.

Tabela 6.2 Possíveis Navegações entre os Pares de Bancos de Dados no MENTAS.

Navegação entre os BDs	
BD Corrente	**BD Posterior**
PSD	LKD
PSD	MENTAS
LKD	PSD
LKD	MENTAS
MENTAS	PSD
MENTAS	LKD

1. Nesta dissertação o termo PSD é utilizado para representar o banco de dados PSD. São utilizados também os termos LKD e Mentas significando os bancos de dados que eles representam.

O princípio da navegação é simples. Seja N o conjunto de elementos que representam os bancos de dados onde pode ocorrer a navegação. No MENTAS, N é representado inicialmente por:

N = {PSD, LKD, Mentas}

A cada banco de dados visitado é retirado do conjunto o nome correspondente. Sendo assim, no momento da inicialização do campo do modelo, o conjunto N acima só terá dois elementos. O controle de quais elementos devem fazer parte deste conjunto é feito pela GUI, e é mostrado ao usuário através da ativação ou não de ítens de menu.

6.3 Pontos de Entrada

O primeiro passo na integração de esquemas é identificar nos bancos de dados que irão compor o esquema global quais os atributos que possuem significado semelhantes. Esta tarefa não é nada trivial. Requer um grande entendimento das modelagens de todos os bancos de dados envolvidos. Na maioria das vezes, cada banco de dados componente foi modelado para atender uma situação específica. Portanto, é comum que tais bancos de dados apresentem modelagens distintas para representar uma mesma informação.

No MENTAS, em primeiro lugar, foram analisados todos os esquemas de todos os bancos de dados componentes de modo a identificar os dados cruciais ao sistema. Depois de definidos estes dados procuramos os pontos que possuíam significado semelhantes entre os bancos de dados. Estes pontos são referenciados por nós como **pontos de entrada** da navegação. Isto porque é através deste pontos em comum que a consulta do usuário a um banco de dados consegue **entrar** nos limites de um segundo banco de dados, possibilitando a integração dessas fontes.

Uma definição mais formal para pontos de entrada no MENTAS é *atributos existentes em pelo menos dois dos bancos de dados e que representam a mesma informação, embora possam ser armazenados em formatos e/ou representações diferentes*. A identificação dos pontos de entrada pode ser vista na Tabela 6.3.

Tabela 6.3 Pontos de Entrada.

PSD		LKD		Mentas	
Entidade	*Atributo*	*Entidade*	*Atributo*	*Entidade*	*Atributo*
PSD_Motor	Typ	LKD_Motor	Typ	Motor	Typ Spezifikation Baumuster
PSD_Zylinder	Durchmesser	LKD_Motor	Bohrung	Motor	Bohrung
PSD_Zylinder	Hubvolumen	LKD_Motor	ZylinderHubRaum	Motor	ZylinderHubvolumen
PSD_Ventil	Ventil_DM_Aussen_1 Ein_Auslass_Kanal	LKD_Motor	Event_TellerDurchm Avent_TellerDurchm	Ventil Motor_Kanal_Ventil	Durchmesser Ein_Auslass_Ventil
PSD_Pleuel	Pleuellaenge	LKD_Motor	PleuelLagerabstand	Pleuel	Laenge
PSD_Kolben	Hub	LKD_Motor	Hub	Kurbelwelle	Hub
PSD_Motor PSD_Kanal	Max_Ventil_Hub Ein_Auslass_Kanal	LKD_Motor	Event_Max_Hub Avent_Max_Hub	Ventil Motor_Kanal_Ventil	Max_Hub Ein_Auslass_Ventil

6.4 Identificação dos Conflitos entre os Pontos de Entrada

O atributo *Typ* da entidade *PSD_Motor* do banco de dados PSD corresponde ao atributo também denominado *Typ* da entidade *LKD_Motor* do banco de dados LKD. A ligação, tanto do PSD quanto do LKD, com o banco de dados Mentas através do atributo *Typ* se dá pela junção de três atributos neste último: *Typ, Spezifikation e Baumuster* (conflitos de esquemas, um para muitos atributos [KGCS95, KS91]).

No caso da entidade *PSD_Ventil*, dependendo do valor do atributo *Ein_Auslass_Kanal*, vai haver a correspondência do atributo *Ventil_DM_Aussen_1* com o atributo *Event_TellerDurchm* ou com *Avent_TellerDurchm* no LKD. Em relação ao Mentas, esse relacionamento se dá através da comparação dos dois atributos, onde *Ventil_DM_Aussen_1* corresponde a *Motor_Kanal_Ventil* quando *Ein_Auslass_Kanal* é igual a *Ein_Auslass_Ventil*. O mesmo caso ocorre para os atributos *Max_Ventil_Hub* e *Ein_Auslass_Kanal* das entidades *PSD_Motor* e *PSD_Kanal* respectivamente. Nos demais casos, cada atributo de um BD corresponde a apenas um único atributo nos outros BDs.

Todas as entidades que possuem um ponto de entrada são automaticamente habilitadas para a navegação assim que o usuário especifica algum atributo na projeção da sua consulta SQL. Caso o usuário deseje navegar para outro banco de dados nesse instante, uma nova janela é apresentada com todos os pontos de entrada da entidade pela qual está realizando a navegação, de forma que para continuar, deve

ser escolhido pelo menos um dentre os apresentados. Por exemplo, suponha que o usuário esteja no PSD e resolve navegar através da entidade *PSD_Zylinder*. No momento em que o usuário escolhe qualquer um dos atributos dessa entidade e o coloca na cláusula de projeção, a opção de navegação dessa entidade é habilitada. Caso o usuário resolva navegar nesse instante, uma janela com o *Durchmesser* e o *Hubvolumem* é automaticamente apresentada, de forma que o usuário precisa escolher pelo menos um dos apresentados (ver seção 5.7.1.7).

Nesta mesma janela é dado um aviso ao usuário sobre o custo de formular uma consulta SQL desse tipo, pois o processamento de consultas nas quais valores não são especificados é o produto cartesiano entre duas ou mais tabelas dos dois BDs envolvidos. Sabendo que algumas relações dos bancos de dados envolvidos possuem mais de vinte mil tuplas, temos como resultado um tempo de espera razoável para o usuário caso resolva optar por esse tipo de navegação.

Através da Tabela 6.3 podemos notar que algumas entidades possuem mais de um ponto de entrada definidos. São elas:

- **no PSD**, apenas a entidade *PSD_Motor* através dos atributos *Typ* e *Max_Ventil_Hub*;
- **no Mentas**, as entidades *Motor* (através dos atributos *Typ*, *Spezifikation*, *Baumuster* e *Bohrung*) e *Ventil* (através dos atributos *Motor_Kanal_Ventil* e *Max_Hub*);
- **no LKD** só é possível realizar a navegação através da entidade *LKD_Motor* - utilizando qualquer um dos atributos da Tabela 6.3.

O usuário pode realizar a navegação entre os bancos de dados de acordo com um dos casos abaixo:

- Sem cláusula de condição;
- Caso não especifique nenhum ponto de entrada na cláusula de condição;
- Especificando apenas um ponto de entrada na cláusula de condição;
- Especificando mais de um ponto de entrada na cláusula condição;

Outro ponto importante sobre a navegação é que só é permitido ao usuário atravessar de um banco de dados para outro se a consulta ao banco de dados corrente retornar algum resultado. Caso não tenha resultados, conseqüentemente a nova consulta que será originada com a navegação também não terá resultados. Isto porque a nova consulta é apenas uma extensão da antiga, acrescida dos novos atributos e entidades de acordo com os pontos de entrada especificados pelo usuário (ver seção 6.8.1 e seção 6.8.2). Ou seja, não será a navegação que mudará o quadro. Nesse caso, uma mensagem de aviso é enviado pela GUI ao usuário e a navegação não é feita.

Identificador do Motor - Um Ponto de Entrada com Características Especiais

O identificador do motor (Tabela 6.4) possui algumas características próprias em cada um dos bancos de dados integrados. Por isso, foi necessário utilizar alguns artifícios para possibilitar ao usuário utilizar este atributo como ponto de entrada.

Tabela 6.4 Identificação do Motor e a Representação nos Bancos de Dados.

PSD		**LKD**		**Mentas**	
Entidade	*Atributo*	*Entidade*	*Atributo*	*Entidade*	*Atributo*
PSD_Motor	Typ	LKD_Motor	Typ	Motor	Typ Spezifikation Baumuster

São muitas as diferenças apresentadas entre os bancos de dados para o atributos que representam o identificador do motor. A primeira delas é no nível de representação dos atributos. No banco de dados Mentas (ver Tabela 6.4) a identificação do motor é representada por três atributos, enquanto nos demais bancos de dados (PSD e LKD) é representado apenas por um atributo. Este problema é abordado como sendo conflito de integração de esquema do tipo um para muitos atributos, sendo este tratado como um sub caso dos conflitos originados de muitos para muitos atributos (ver Figura 3.5). *Kim et al.* [KGCS95] propõe várias técnicas para a resolução de conflitos de esquemas, e dentre elas, é apresentado o problema apontado anteriormente. Abaixo, apresentamos a definição do problema, a solução proposta e um exemplo, segundo [KGCS95].

***Problema**: As informações podem ser representadas em diferentes níveis de detalhes, especialmente quando são representadas como strings de caracter. Dessa forma, conflitos de um para muitos atributos podem surgir se a informação capturada de um único atributo de um banco de dados corresponde a mais que um atributo em outro banco de dados.*

***Solução:** Concatenar os atributos que aparecem em mais de um nível.*

***Exemplo:** Em um banco de dados, o nome de uma pessoa é representado através dos atributos primeiro_nome e último_nome, e em outra fonte de dados essa mesma informação é representada apenas por um único atributo nome. Dessa forma, na hora da integração dos esquemas, deve-se definir um operador de concatenação, para que a informação torne-se homogênea na integração dos dados.*

Embora nosso problema se enquadre perfeitamente na definição acima, a solução no nosso caso não é tão simples assim. Essa solução poderia ser adotada, caso todos os bancos de dados integrados tivessem um padrão para a seqüência de caracteres que compõe um identificador de motor, o que não é o caso.

Na nossa realidade, cada banco de dados possue uma representação própria para esse atributo, e com a concatenação, disponibilizaríamos apenas que uma consulta fosse feita envolvendo os três bancos de dados, mas de forma alguma retornaríamos um resultado satisfatório/adequado para os usuários. Em vez da concatenação dos atributos, a solução adequada para o nosso caso é justamente o contrário da que foi proposta por *Kim et al.*[KGCS95]. Ou seja, desagrupar o atributo deixando com o formato do Mentas.

De acordo com os engenheiros da *Mercedes-Benz*, um motor é identificado levando em consideração três características: Tipo, Especificação e Modelo de Construção. No projeto do banco de dados Mentas, essas informações foram modeladas nos atributos *Typ*, *Spezifikation* e *Baumuster*, nesta ordem. Infelizmente, os outros bancos de dados não seguem este padrão. Além da própria modelagem do banco de dados ser diferente, utilizando apenas um único atributo, muitos caracteres diferentes são usados para separar as diferentes partes do identificador do motor.

Como o Mentas foi projetado para armazenar os dados dos motores fabricados atualmente ou que ainda serão fabricados nos próximos anos pela *Mercedes-Benz*, e principalmente por seguir a padronização da *Mercedes-Benz*, decidimos usá-lo como modelo padrão para a comparação com os outros bancos de dados. Depois de analisar os dados do Mentas, estabelecemos uma **gramática** através da qual devemos comparar os motores dos outros bancos de dados, identificando semelhanças e diferenças.

Através de informações dos engenheiros mecânicos da *Mercedes-Benz*, podemos distinguir a princípio o motor de duas maneiras básicas: motores a gasolina e motores a diesel. Dessa forma, transcrevemos a informação para nossa gramática, onde, o significado do tipo do motor é associado à letra que inicia a descrição. Motores a gasolina são representados pela letra M seguida de três dígitos. Já os motores a Diesel são representados pelas letras OM seguidas novamente de três dígitos -Tabela 6.5.

Tabela 6.5 Gramática - Tipo do Motor.

Gramática – Tipo	
Gasolina	***Diesel***
M <dígito><dígito><dígito>	OM<dígito><dígito><dígito>

Já a especificação é usada como o próprio nome indica, para dar características específicas aos motores, e segue a gramática apresentada na Tabela 6.6.

Tabela 6.6 Gramática - Especificação.

Gramática – Especificação
<letra><dígito><dígito>
<letra><dígito><dígito> <letra><letra>
<letra><letra><dígito><dígito> <letra><letra>
<letra><letra><dígito><dígito> <letra>

O modelo de construção (*baumuster*) é usado para dar características de como será a construção do motor. O exemplo clássico é o caso da localização do câmbio nos carros da *Mercedes-Benz*. Dependendo do país onde é fabricado, o câmbio pode estar localizado em locais diferentes: ou entre os bancos do motorista e passageiro ou junto a direção. É representado por três dígitos - Tabela 6.7.

Tabela 6.7 Gramática - *Baumuster.*

Gramática – Baumuster
<dígito><dígito><dígito>

Nem todos os motores do Mentas têm necessariamente os três atributos. Existem alguns identificadores de motores que são a combinação do tipo e especificação, outros que combinam tipo e a modelo de construção, e ainda, outros que combinam o tipo, especificação e modelo de construção. Mas, em todos os casos, existe a presença do tipo.

Todos os demais bancos de dados devem seguir este padrão, para que seja possível a comparação entre os valores e, conseqüentemente, a navegação. Apesar de o LKD e o PSD utilizarem apenas um atributo para representar essa informação, são tantas as diferenças semânticas entre os valores dos dados, que se fizermos uma consulta envolvendo esses dois BDs e procurarmos por todos os motores que são iguais, só iremos encontrar dois motores distintos. Este resultado não é nada razoável. Isto acontece porque os bancos de dados são utilizados por departamentos diferentes, tendo cada um, sua própria representação para os motores. Por exemplo, um motor representado em um departamento por OM668.940 pode ser igual ao motor representado em outro departamento por OM668DE17. Apesar de visualmente não ser o mesmo motor, o significado da interpretação pelos engenheiros da *Mercedes-Benz* é o mesmo.

Então, com a separação do identificador do motor no formato apresentado pelo Mentas, é possível procurar por informações de um determinado motor em mais de um banco de dados tanto através do motor completo caso ele exista em ambos, ou através de um dos atributos que compõe a identificação do motor após a formatação - tipo, especificação ou modelo de construção.

Analisando os outros bancos de dados tomando como parâmetro o formato do Mentas, notamos que existem algumas particularidades e similaridades em cada um. Por exemplo, tanto o PSD como o LKD possuem a seqüência de caracteres que definem o tipo para um motor. Entretanto, a especificação é mais comum nos motores do PSD, aparecendo poucas vezes nos motores do LKD. E o modelo de construção, por sua vez, faz-se presente em quase todos os motores do LKD, mas em nenhum do PSD. Ou seja, no PSD os motores basicamente são constituídos do tipo mais a especificação e no LKD pelo tipo e modelo de construção.

6.5 Definição de Regras para Solucionar os Conflitos

A solução para a heterogeneidade nas fontes de dados utilizadas pelo MENTAS é baseada na tecnologia *middleware* (ver Capítulo 5). Apesar do *middleware* utilizado -DataJoiner [IBM95, RH98]- proporcionar a interação entre fontes de dados heterogêneas (providenciando a transparência de localização das fontes de dados, transparência nas diferenças de comunicação, etc.), não é feito nenhum tratamento das diferenças estruturais e semânticas entre os dados. Estas diferenças foram tratadas separadamente pelo Monitor de Consistência. Sem realizar este tratamento os resultados conseguidos por consultas envolvendo dois bancos de dados não foram nem um pouco satisfatórios. Consultas que deveriam retornar dezenas ou centenas de tuplas, muitas vezes retornavam algumas unidades ou mesmo nada.

Portanto, devem ser analisados os banco de dados "parcialmente" integrados pelo DataJoiner procurando sempre identificar quais atributos vão possibilitar a integração entre as bases de dados. Além disso, devemos analisar tais atributos para

saber a necessidade de efetuarmos mudanças no momento da integração. Se realmente for necessário, partimos para uma nova fase, para saber quais mudanças serão feitas nos valores para que a comparação entre eles nos retorne os valores desejados.

O procedimento de integração de esquema apresentado por [RR99] (ver Figura 3.6) é composto por quatro fases: tradução de esquemas, identificação dos relacionamentos entre os esquemas, geração dos esquemas integrados e geração de esquemas de mapeamento. Comparando com procedimentos empregados no MENTAS, podemos notar que a primeira fase é feita implicitamente pelo DataJoiner. Podemos notar que das quatro fases propostas por [RR99] apenas a segunda - Identificação dos relacionamentos entre os esquemas - está realmente presente no nosso esquema.

Partindo deste princípio, a primeira tarefa do monitor de consistência foi definir algumas características de cada banco de dados, com o objetivo de construir regras que devem ser executadas na hora da navegação entre os bancos de dados. Esta tarefa foi realizada no momento de definir o esquema global e virtual do MENTAS (figura 3.2). Abaixo mostraremos as características e no final de cada uma, a regra que deve ser aplicada. As regras abaixo são utilizadas tanto pelo monitor de consistência do servidor, como também pela GUI e pelo monitor de consistência do cliente para fazer o tratamento dos valores de entrada do usuário.

Domínios dos atributos:

- O banco de dados LKD possue sempre o CHAR como domínio para seus atributos, independente do que possam representar;
- PSD e Mentas possuem domínios diferentes para os atributos de acordo com o que possam representar. O tipo dominante dos pontos de entrada nesses bancos de dados é o FLOAT. Apenas os atributos do identificador de motor, *Ein_Auslass_Kanal*, *Ein_Auslass_Ventil* possuem o tipo VARCHAR.

Regra 1: Entrada de Dados: Usar aspas simples nos valores dos atributos que possuem o tipo CHAR ou VARCHAR.

Regra 2: Navegação: Transformar o domínio dos atributos quando necessário, de forma que seja possível a comparação de valores entre eles.

Separador de Casas Decimais:

- O LKD utiliza a vírgula como separador de casas decimais;

- O PSD e o Mentas utilizam o ponto como separador de casas decimais.

Regra 3: Sempre que ocorrer a navegação entre o LKD e os demais bancos de dados, é necessário fazer a troca do separador de casas decimais, de vírgula para ponto e vice-versa.

Representação do Identificador do Motor

LKD:

- Utiliza apenas um único atributo;
- Raramente possue a seqüência de caracteres que representa a especificação de um motor, e quando isso acontece, não possue a seqüência que especifica o *baumuster*;
- A seqüência de caracteres que representa o tipo é separada pela seqüência de caracteres que representam o *baumuster* através de um ponto;
- Alguns motores apresentam espaços em branco no início, entre e no final da seqüência dos caracteres que identificam os motores.

PSD:

- Usa apenas um único atributo;
- Não possue a seqüência de caracteres que representa o *baumuster*;
- Alguns motores apresentam espaços em branco no início e no final da seqüência dos caracteres que identificam os motores.

Mentas:

- Utiliza três atributos para a identificação, seguindo o padrão estabelecido pela Mercedes-Benz;
- Apresenta, para a maioria dos motores, a presença das três seqüências que os compõem;

Regra 4: Para os bancos de dados que possuem apenas um atributo para o identificador do motor, fazer a separação utilizando o formato empregado no Mentas, retirando espaços em branco, pontos e todas as demais diferenças que possam existir. Deve-se utilizar a gramática definida na Tabela 6.5, Tabela 6.6 e Tabela 6.7.

Diferentes Escalas:

- Alguns atributos possuem diferentes escalas para representar os valores. Alguns consideram cinco casas decimais e outros apenas duas.

Regra 5: Para atributos com o número de casas decimais distintas, deve-se considerar apenas duas casas decimais no momento da comparação dos valores.

Uma vez identificado os pontos em comum entre os bancos de dados, as diferenças entre eles e realizando a definição das regras que devem ser seguidas no momento da integração, partimos para o último e indispensável passo que é a resolução de tais conflitos. As duas próximas seções apresentam como foi solucionado os conflitos entre os bancos de dados no MENTAS. Começamos por apresentar o monitor de consistência do servidor pelo fato deste apresentar mais funcionalidades que o monitor de consistência do cliente. Na verdade, o monitor de consistência do cliente possui um subconjunto das funcionalidades presentes no servidor sendo diferenciado apenas pelo acréscimo do tratamento de entrada de dados.

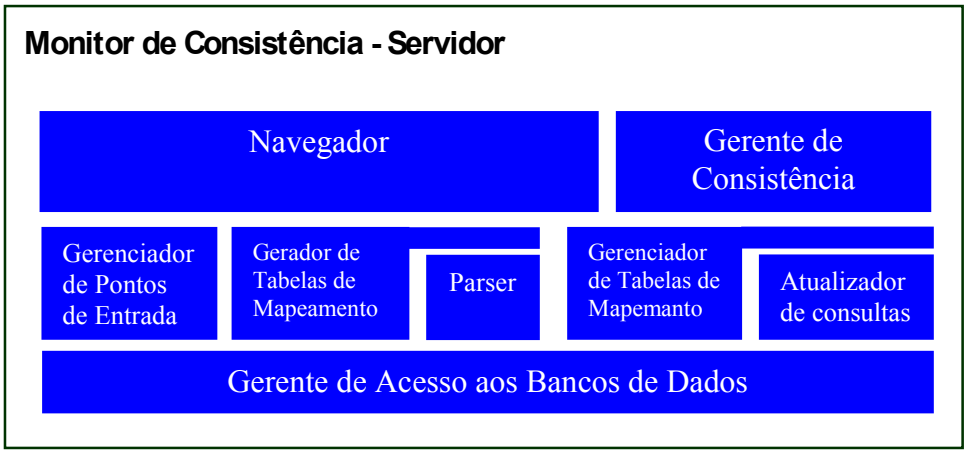
6.6 Monitor de Consistência do Servidor - MCS

Como já foi dito anteriormente, não é apenas com a formatação dos dados de entrada que o problema de incompatibilidade dos dados dos bancos de dados integrados está resolvido. Além disso, os dados das fontes remotas também devem ser postos no mesmo formato, de forma que no final haja um casamento perfeito entre eles. É para isso que existe o monitor de consistência do servidor. As principais tarefas executadas por este módulo são o gerenciamento dos pontos de entrada, o gerenciamento das tabelas de mapeamento e principalmente a realização da navegação de um banco de dados para outro. Essa tarefas são distribuídas entre os componentes que são apresentados na arquitetura do monitor de consistência (Figura 6.1).

6.6.1 Arquitetura

A arquitetura do MCS pode ser vista na Figura 6.1. O módulo de acesso aos bancos de dados é utilizado por todos os componentes e será abordado na seção 6.6.2. Logo a seguir, na seção 6.6.3 apresentaremos o Gerente de Pontos de Entrada que é responsável pelo mapeamento de quase todos os pontos de entrada apresentados na Tabela 6.3. A única exceção é o atributo que identifica o motor, que é tratado num caso a parte. Quatro dos oito componentes do Monitor de Consistência estão voltados diretamente para realizar o mapeamento desse atributo nos diversos bancos de dados. São eles: o Gerador de Tabelas de Mapeamento, o Parser, o Gerenciador de Tabelas de Mapeamento, o Atualizador de Consultas e o Gerente de Consistência, que são apresentados da seção 6.6.4 à seção 6.6.7 respectivamente. O principal módulo da nossa arquitetura é o Navegador. Todos os demais módulos existem para auxiliá-lo. Sendo assim, o explicaremos na seção 6.6.8. O Gerente de Consistência será abordado logo a seguir na seção 6.6.9, visto que esse módulo é uma extensão do Navegador.

Figura 6.1 Arquitetura do Monitor de Consistência.



6.6.2 Gerente de Acesso aos Bancos de Dados

O módulo de Acesso aos Bancos de Dados tem a função de interceptar qualquer consulta SQL referente à navegação, montada pelo usuário através da GUI e enviá-la para processamento no servidor de banco de dados.

Antes de cada consulta ser enviada para o processamento no servidor, uma conexão deve ser estabelecida ao sistema *middleware* de banco de dados. Isto é feito através de um pedido ao conector de banco de dados, responsável pelo gerenciamento das conexões, e esta é retornada (ao conector de banco de dados) tão logo haja o processamento da consulta. O pedido de conexão é diferenciado para operações de leitura e operações de escrita como foi visto na seção 5.5.1.

Neste nível, foi empregado o JDBC (*Java Database Connectivity* [SUN98, HCF97]) para a comunicação com o sistema *middleware* de banco de dados. O poder e funcionalidade da API (*Application Programming Interface*) do JDBC corresponde ao padrão ISO do SQL2. O JDBC emprega o protocolo TCP/IP padrão ISO para a comunicação.

A liberação dos recursos do banco de dados é outra tarefa do gerente de acesso. Existem duas formas principais onde podemos liberar recursos dos bancos de dados: através do ***fechamento de consulta*** ou do ***fechamento das conexões*** que não são mais necessárias. As ***conexões***, como foi explicado, são gerenciadas pelo conector de banco de dados e portanto não é tratado pelo gerente de acesso aos bancos de dados. A única tarefa necessária nesse caso é certificar-se que devolvemos

a conexão ao final do processamento da consulta. Por outro lado, as ***consultas*** podem ser fechadas ao final do processamento, e esta é a nossa contribuição a nível local.

Na fase inicial de implementação, o monitor de consistência gerenciava sua própria conexão com o sistema *middleware* de banco de dados, a qual era estabelecida no construtor da nossa classe. Entretanto, esta não era uma boa solução, visto que não deve-se manter uma conexão aberta por muito tempo, correndo o risco de esgotar os recursos do banco de dados. Não é uma boa prática também, abrir e fechar conexões a todo momento. Dessa forma, tornava-se difícil ter o controle da conexão a nível local, o qual foi delegado mais tarde ao conector de banco de dados.

6.6.3 Gerenciador de Pontos de Entrada

O papel do gerenciador de pontos de entrada - GPE - é fazer o mapeamento entre os atributos e entidades correspondentes em cada SGBD, bem como formatar os valores para que seja possível a comparação entre os mesmos. Além disso, é tarefa dele informar aos demais módulos do monitor de consistência, caso necessitem, os pontos de entrada correspondentes nos demais bancos de dados, de acordo é claro, com um ponto de entrada específico e a entidade (visto que alguns pontos de entrada possuem o mesmo nome). Mas, nem todos os pontos de entrada são tratados por este módulo. O gerenciador de pontos de entrada não faz tratamento dos atributos que identificam o motor (atributo *Typ* do PSD e LKD e *Typ, Spezifikation, Baumuster* no Mentas). Outros componentes da arquitetura são responsáveis por este mapeamento.

Funcionalidade do GPE

O gerenciador de pontos de entrada é ativado caso o usuário especifique qualquer um dos atributos da Tabela 6.3 na cláusula de condição da sua consulta e queira fazer a navegação para outro banco de dados. Neste momento, a GUI faz uma chamada ao servidor, mais precisamente ao gerenciador de pontos de entrada, passando como parâmetros a atual consulta feita pelo usuário, os pontos de entrada especificados no banco de dados corrente, os valores desses pontos, o banco de dados corrente e o banco de dados para o qual o usuário deseja navegar. Essas informações são necessárias para mapear os atributos e formatar seus valores de acordo com os banco de dados no qual o usuário está trabalhando e para o qual está navegando.

O gerente de pontos de entrada envia a consulta SQL do usuário para o servidor de banco de dados para que esta seja executada. Caso essa consulta não retorne nenhum resultado - dizemos então que a consulta não é válida - um código é retornado à GUI, que por sua vez, o reporta ao usuário como uma mensagem de aviso, e a navegação não é feita.

Caso a consulta SQL do usuário retorne algum resultado do banco de dados, é dado prosseguimento à navegação. É feito o mapeamento de acordo com os atributos (leia-se pontos de entrada) e os valores, as entidades, o banco de dados corrente e o banco de dados para o qual o usuário deseja navegar, que foram informadas no momento da chamada a este módulo.

Para fazer o mapeamento entre os valores especificados para o(s) ponto(s) de entrada do BD corrente e o correspondente no BD posterior, o GPE utiliza as regras definidas na seção 6.5. Aqui é feito então o tratamento de diferenças de tipos entre os atributos e diferenças nas representações de valores [KGCS95, HM93].

Tabela 6.8 Diferenças nas Representações dos Pontos de Entrada.

	PSD	*LKD*	*Mentas*
Tipos dos Atributos	FLOAT	CHAR(x)	FLOAT
Separador	Ponto	Vírgula	Ponto

O tratamento de entrada dos dados é feito pelo módulo Gerente de Entrada localizado no Cliente (seção 6.6.3), sendo este tratamento feito em relação ao banco de dados corrente, verificando as regras definidas na seção 6.5. O GPE faz o tratamento dos valores para o BD posterior, deixando transparente para o usuário qualquer diferença da representação. Mostraremos agora através de um exemplo, todos os passos do GPE numa suposta navegação entre o PSD e o LKD.

A situação mais simples de ocorrência de uma navegação no ambiente do MENTAS acontece quando o usuário especifica apenas por qual ponto de entrada será feita a ligação entre os bancos de dados, sem entretanto especificar qualquer valor para o atributo no banco de dados corrente (seção 5.7.1.7). Neste caso não é necessário qualquer transformação de dados de entrada (pois não foi informado nenhum valor para o atributo). Por exemplo, suponha que o usuário possua como banco de dados corrente o PSD e formulou a seguinte consulta:

```
SELECT PSD_Zylinder. Durchmesser
FROM PSD_Zylinder
```

Suponha ainda que o usuário escolheu a entidade PSD_Zylinder para dar início à navegação. E mais, que escolheu o atributo *PSD_Zylinder.Durchmesser* para ser o ponto de ligação com o banco de dados MENTAS. A GUI envia então as seguintes informações para o gerente de pontos de entrada:

- Pontos de entrada especificados pelo usuário (uma entidade pode possuir mais de um ponto de entrada), bem como a entidade que o atributo pertence;

- Banco de dados corrente e banco de dados para o qual o usuário deseja navegar;

Este caso é muito simples. Através da Tabela 6.3 é inferido qual é o atributo correspondente no banco de dados Mentas para o atributo *PSD_Zylinder.Durchmesser* do PSD (Motor.Bohrung). Dessa forma, montamos a consulta final originada através da montagem da cláusula de condição como a apresentada a seguir:

```
SELECT PSD_Zylinder.Durchmesser
FROM PSD_Zylinder, LKD_Motor
WHERE PSD_Zylinder.Durchmesser = Motor.Bohrung
```

Um exemplo um pouco mais complicado aparece se o usuário especificar um valor para o ponto de entrada. Por exemplo, suponha que o usuário esteja montando a sua consulta no banco de dados PSD (banco de dados corrente é o PSD) e especifique o atributo *Durchmesser* da entidade *PSD_Zylinder* na cláusula de condição da sua consulta com o valor 20.3. Agora, o usuário deseja realizar a navegação para o banco de dados LKD através deste ponto de entrada. O procedimento então é abrir o *pop up menu* da entidade PSD_Zylinder e escolher a opção *other databases*. Temos então a seguinte consulta montada pelo usuário através da GUI:

```
SELECT PSD_Zylinder. Durchmesser
FROM PSD_Zylinder
WHERE PSD_Zylinder. Durchmesser = 20.3
```

A GUI então dispara um pedido ao gerenciador de entradas passando as seguintes informações:

- Consulta SQL atual;
- Pontos de entrada especificados pelo usuário, bem como a entidade que o atributo pertence;
- Banco de dados corrente e banco de dados para o qual o usuário deseja navegar;
- Valor especificado para o ponto de entrada e o comparador (=, <, >, like, etc.).

Um vez com essas informações, começa então o trabalho do GPE. Suponha novamente que a consulta SQL retorne alguma tupla quando aplicada ao banco de dados. O que deve ser feito agora é o mapeamento do nome do atributo e valor especificado pelo usuário aos correspondentes para o banco de dados posterior. Assim, transformamos o valor 20.3 para '20,3' (note que o ponto foi trocado por uma vírgula e que foi acrescentado aspas ao valor, já que o BD posterior é o LKD) e mapeamos a entidade e o atributo *PSD_Zylinder* e *Durchmesser*, para os correspondentes no LKD, ou seja, *LKD_Motor* e *Bohrung*, respectivamente.

Terminado o trabalho, o GPE retorna para a GUI a entidade, o atributo e os valores já mapeados, os quais são utilizados para completar a consulta do usuário. É en-

cerrado então para o GPE o ciclo de navegação. Começa então o trabalho da GUI a qual acrescenta as informações necessárias à consulta do usuário como também troca o modelo ER antigo apresentado no campo de modelo da janela principal - no nosso exemplo o PSD - para o novo banco de dados corrente, no nosso caso, o modelo ER do LKD. Pronto, para o usuário a navegação está concluída, e a consulta SQL apresentada ao usuário através do campo de consultas passou a ser a seguinte:

```
SELECT PSD_Zylinder.Durchmesser
FROM PSD_Zylinder, LKD_Motor
WHERE PSD_Zylinder.Durchmesser = 20.3 AND LKD_Motor.Bohrung = '20,3'
```

Apresentamos nesta seção um exemplo simples no qual a ligação entre os bancos de dados foi feita utilizando um único ponto de entrada. No caso do usuário especificar mais de um ponto de entrada na sua consulta SQL, o mesmo procedimento é adotado para cada um dos pontos. Isso não implica em várias chamadas ao servidor. A GUI passa para o GPE um vetor de elementos que são tratados e, logo após isso, retornados. Não importa a quantidade de pontos de entrada definidas pelo usuário, o tratamento é homogêneo e a chamada ao servidor única.

O tratamento dispensado para o atributo *PSD_Zylinder.Durchmesser* é adotado também para os seguintes pontos de entrada: *PSD_Zylinder.Hubvolumen, PSD_Pleuel.Pleuellaenge, PSD_Kolben.Hub*. Todos esses atributos utilizam a **regra 1**, a **regra 2** e a **regra 3**. É claro que o mesmo é válido para os correspondentes destes pontos de entrada nos demais bancos de dados. No caso do *PSD_Zylinder.Hubvolumen* ainda é necessário o uso da **regra 5** porque os bancos de dados possuem diferentes níveis de precisões nos valores. Desta forma, no momento da comparação faz-se necessário que os valores sejam truncados em duas casas decimais. Isto é feito com o auxilio de funções presentes no próprio SGBD.

Os atributos *PSD_Ventil.Ventil_DM_Aussen_1, PSD_Motor. Max_Ventil_Hub e PSD_Kanal.Ein_Auslass_Kanal* (e todos os correspondentes nos demais bancos de dados) precisam de um pouco mais de cuidado no tratamento porque não existe um mapeamento de um para um entre estes atributos nos bancos de dados (ver Tabela 6.3). Como foi visto é preciso a definição de dois atributos nos bancos de dados PSD e Mentas para corresponder a um único no banco de dados LKD, e vice-versa. A problemática destes pontos de entrada ocorre porque no MENTAS os usuários podem montar as consultas de forma *ad hoc*. Isto implica que o usuário pode definir estes pontos sem uma ordem prévia. Para piorar o quadro, o usuário pode combinar vários valores de um atributo para um único valor de um segundo (no caso no PSD e Mentas no qual o ponto de entrada é constituído de dois atributos e não apenas de um único). Como exemplo, podemos ver as consultas abaixo:

```
SELECT PSD_Ventil.Ventil_DM_Aussen_1
FROM PSD_Ventil
WHERE (PSD_Ventil.Ventil_DM_Aussen_1 = 30 OR
PSD_Ventil.Ventil_DM_Aussen_1 = 30.5 OR
```

PSD_Ventil.Ventil_DM_Aussen_1 = 46.7) **AND**
PSD_Ventil.Ein_Auslass_Kanal = 'E'

Como um fator ainda mais complicador, o usuário pode definir outros atributos entre *PSD_Ventil.Ventil_DM_Aussen_1* e *PSD_Ventil.Ein_Auslass_Kanal*. Por todas essas características, deve ser montado um tradutor de forma a agrupar os valores antes de fazer o mapeamento. Note que o usuário não percebe essa dificuldade. Ele apenas monta a consulta da forma que lhe é mais conveniente e requisita a navegação. É responsabilidade do monitor de consistência e da GUI fazer todo o tratamento.

6.6.4 Parser

Como vimos, a modelagem do identificador do motor nos bancos de dados do MENTAS é variada e é necessário fazer a transformação dos atributos seguindo a gramática definida na Tabela 6.5, Tabela 6.6 e Tabela 6.7. O parser é o responsável por deixar os identificadores dos motores em um formato único, utilizando esta gramática. Para cada identificador de motor, o parser gera um vetor com quatro campos. O primeiro para o *tipo*, o segundo para a *especificação*, o terceiro contendo o *modelo de construção* e o quarto e último contendo seqüências que não pertencem à gramática - *desconhecido*. Na Tabela 6.9 e Tabela 6.10 são apresentados os resultados do parser para alguns identificadores de motor, no LKD e no PSD.

Note a variação entre as representações dos motores. Em um banco de dados, o tipo é seguido da especificação sem possuir entretanto o modelo de construção (PSD). Em outro banco de dados, o tipo é seguido do modelo de construção, sendo que este dois atributos na maioria da vezes são concatenados através de um ponto. Algumas vezes a especificação (no primeiro banco de dados) e o modelo de construção (no segundo banco de dados) é seguida de seqüências de caracteres que não seguem nenhum padrão, totalmente aleatórias. Outras vezes os motores só possuem o tipo seguido da seqüência desconhecida pela gramática.

Tabela 6.9 Parser - PSD.

Motor	*Tipo*	*Especificação*	*Arte de Construção*	*Desconhecido*
M106E25	M106	E25		
M111E18MMX	M111	E18		MMX
OM602DE29LA	OM602	DE29LA		
OM667-LOMIX	OM667			-LOMIX

Tabela 6.10 Parser - LKD.

Motor	*Tipo*	*Especificação*	*Arte de Construção*	*Desconhecido*
M 110.986	M110		986	
M102.910 ECE	M102		910	ECE
OM 605.910 KAT	OM605		910	KAT
OM601.9 AUT.	OM601			9 AUT.
M111E23	M111	E23		

Podemos notar que alguns motores possuem o campo desconhecido preenchido. No caso do Motor M111E18MMX por exemplo, conseguimos realizar o casamento entre o tipo e a especificação. Entretanto, o restante da seqüência - MMX - não se enquadra para o modelo de construção, sendo tratado como uma seqüência desconhecida da gramática. O mesmo ocorre para os demais motores que apresentam este campo.

Os dados já formatados pelo parser são a base para que ocorra a navegação por este ponto de entrada. Após a passagem dos identificadores do motor por este módulo, temos uma representação homogênea entre todos os bancos de dados. Com os dados formatados surgiu um novo problema: *como gerenciar esses dados de modo que as operações de join entre os bancos de dados possam ser feitas?* A primeira alternativa seria mudar a modelagem dos bancos de dados originais. Esta solução torna-se inviável porque iria infrigir a garantia de autonomia aos bancos de dados integrados. De forma alguma deve-se mudar qualquer banco de dados componente para se adequar ao ambiente do MENTAS. Por outro lado, se o identificador do motor não for transformado para uma representação homogênea não poderão ser comparados entre si.

Outra alternativa seria executar o parser a cada ocorrência da navegação e comparar os dados a nível de programação, sem a presença de banco de dados. Formataríamos os dados de entrada do usuário e faríamos a comparação com os resultados do parser. Mais uma vez não seria uma alternativa viável formatar todos os identificadores do motor a cada navegação porque a performance do sistema seria prejudicada. Por isso, decidimos utilizar tabelas intermediárias para armazenar os resultados do parser. Estas tabelas intermediárias são chamadas de *tabelas de mapeamento*.

De acordo com a arquitetura apresentada na Figura 6.1 podemos notar que o parser é na realidade um sub-módulo do *gerador de tabelas de mapeamento*. O funcionamento do parser do servidor está totalmente ligado ao gerador de tabelas de mapeamento. Este módulo envia para o parser os identificadores do motor no formato original que estão armazenados nos bancos de dados e recebe exatamente no formato que serão armazenados na tabela de mapeamento.

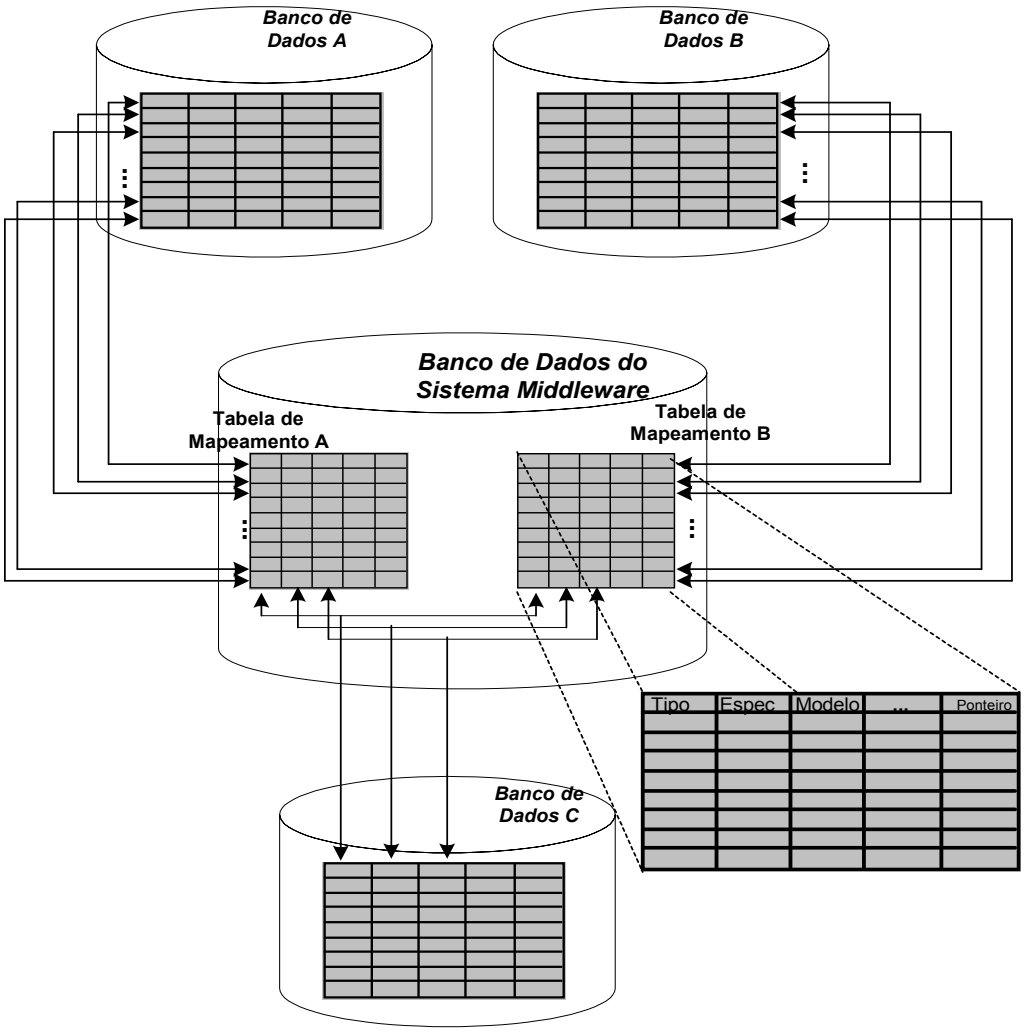
6.6.5 Gerador de Tabelas de Mapeamento - GTM

Este é o segundo módulo que trata do mapeamento entre os bancos de dados através do identificador do motor. Podemos vê-lo como a parte centralizadora que requisita e integra os serviços dos outros módulos. O gerador de tabelas de mapeamento tem a finalidade de recuperar os valores originais dos identificadores do motor, enviá-los para o parser, e salvar os resultados do parser nas tabelas de mapeamento. Além disso é responsabilidade deste módulo também recuperar todos os demais pontos de entrada e armazená-los nas mesmas tabelas de mapeamento.

As tabelas de mapeamento são geradas pela primeira vez na primeira inicialização do servidor. Neste momento são buscados todos os pontos de entrada das fontes de dados originais, o parser é chamado para formatar os valores dos identificadores do motor e a seguir todos os valores são armazenados nas tabelas de mapeamento. Estas tabelas de mapeamento são armazenadas em um banco de dados gerenciado pelo sistema *middleware*. Essas tabelas podem ser geradas também pelo usuário. Através da GUI o usuário pode requisitar informações sobre esta tabela e decidir se deseja ou não realizar um novo mapeamento.

Entretanto, para que esta técnica tenha o efeito esperado, é preciso que haja uma ligação eficiente entre os dados mapeados e as fontes de dados originais, de modo que as outras informações da relação que armazena o identificador de motor sejam recuperadas com sucesso. Para criar esta referência entre as tabelas, um novo atributo foi inserido nas tabelas de mapeamento, além dos dados formatados pelo parser. Este atributo é o motor no formato original (Figura 6.2). Em relação ao banco de dados Mentas, este relacionamento é feito através de três atributos -tipo, especificação e baumuster-, em conjunto ou separadamente.

Figura 6.2 Ligação entre as Tabelas de Mapeamento e as Fontes de Dados Remotas

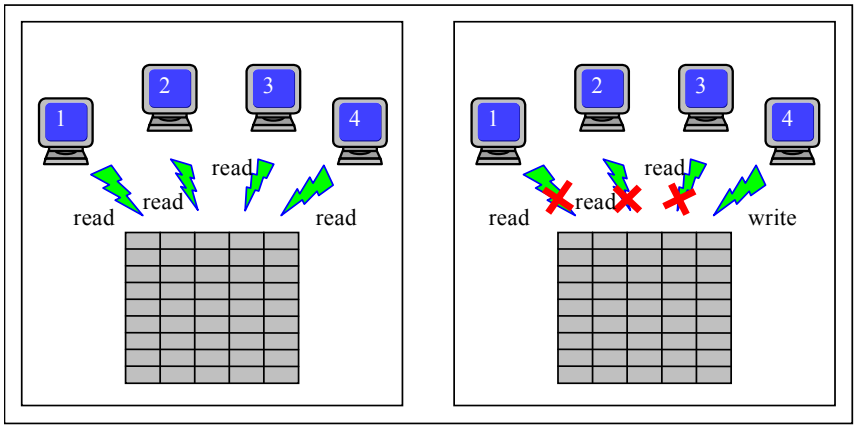


As tabelas de mapeamento têm uma importância vital para que a navegação ocorra com sucesso. Todas as consultas que envolvem os pontos de entrada que identificam o motor acessam-nas obrigatoriamente. O acesso também ocorre quando o usuário especifica que a navegação deve ocorrer seguindo as normas de consistência entre os pontos de entradas dos bancos de dados envolvidos. Portanto, uma propriedade que deve estar presente nestas tabelas é a sua disponibilidade aos usuários sempre que forem requisitadas. Por esta razão, decidimos utilizar duas tabelas, uma ativa e uma reserva, como veremos a seguir.

Considere o caso em que apenas uma tabela de mapeamento é utilizada. Como sabemos, operações de leitura e escrita devem ser cuidadosamente tratadas para garantir a consistência dos dados acessados. Dessa forma, quando um usuário requisitar uma operação que resulte em escrita de dados na tabela (no nosso caso,

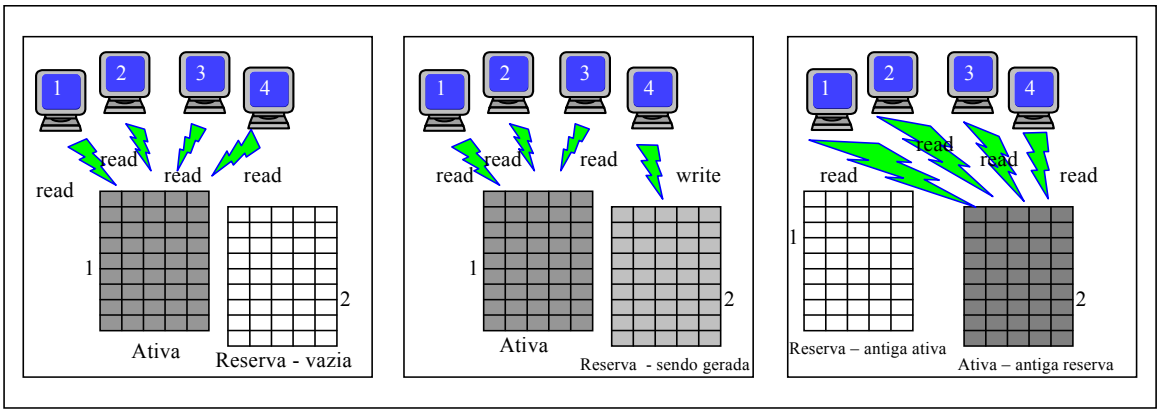
isto ocorre quando o usuário requisita através da GUI que um novo mapeamento deve ser feito) os demais devem aguardar até que a transação seja concluída. Porém, caso a transação não seja tão simples, o que significa que a tabela ficará bloqueada por um longo período de tempo, todos os demais usuários ficarão inabilitados a fazer acesso aos dados da tabela. Podemos ver o caso ilustrado na Figura 6.3. O primeiro quadro da esquerda apresenta o caso em que todos os usuários utilizam a tabela apenas para leitura, o que não ocasiona problema algum. O quadro da direita mostra exatamente o problema onde um usuário está realizando uma operação de escrita na tabela e os demais aguardam sem poder acessá-la. Isso significaria no MENTAS a paralisação da navegação por todos os usuários, até que um novo mapeamento fosse concluído por um usuário, sendo portanto, uma alternativa inviável.

Figura 6.3 Problema de Acesso as Tabelas de Mapeamento.



Para garantir que este tipo de problema não exista nas tabelas de mapeamento do MENTAS, nós usamos duas tabelas para cada banco de dados, sendo que apenas uma está ativa num dado instante. A Figura 6.4 ilustra a nossa solução:

Figura 6.4 Solução Adotada para as Tabelas de Mapeamento.



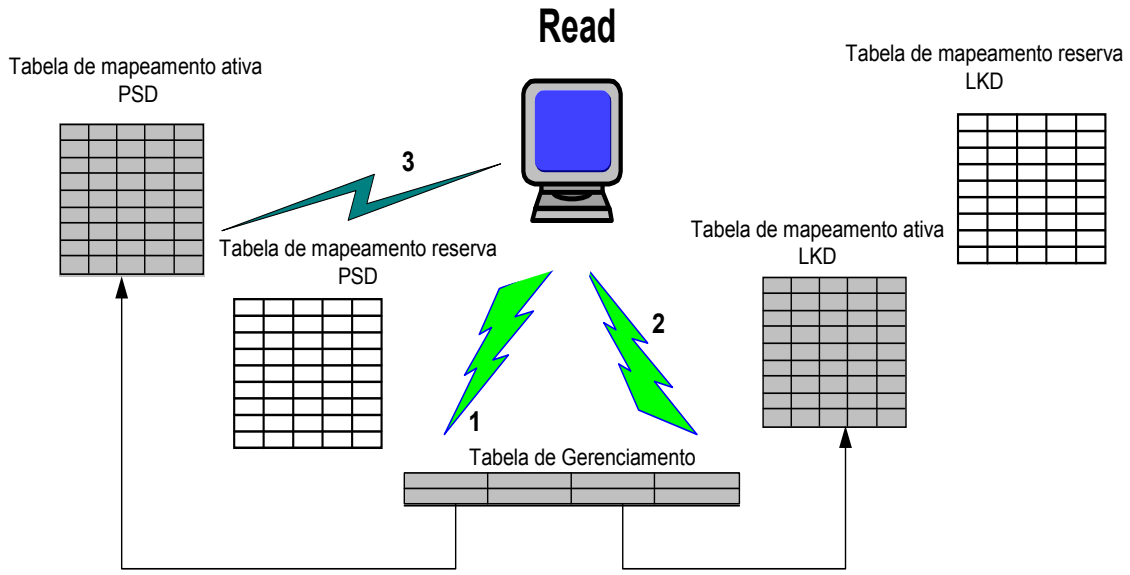
O primeiro quadro ilustra o caso em que todos os usuários estão acessando a tabela de mapeamento ativa, enquanto a outra está vazia - tabela reserva. Se algum usuário resolver gerar um novo mapeamento, ele o faz utilizando a tabela reserva (segundo quadro). Com isso, não existe o problema apontado anteriormente, dos demais usuários ficarem impossibilitados de navegar pelos bancos de dados integrados até que o novo mapeamento seja completado. A navegação entre os bancos de dados continua sendo realizada sem nenhum problema. Quando o novo mapeamento torna-se completo (quadro três da Figura 6.4), a tabela de reserva torna-se a tabela ativa e vice-versa.

Esta técnica introduz um novo problema: *como gerenciar as tabelas de mapeamento de modo que seja feito o acesso a tabela correta?* A Figura 6.5 apresenta como é feito o relacionamento entre as tabelas de mapeamento e as tabelas de gerenciamento e os usuários numa operação de leitura (*read*). Os nomes das tabelas de mapeamento ativas para cada banco de dados são armazenados numa outra tabela que é a encarregada de informar aos usuários qual é a tabela de mapeamento ativa no momento de um acesso. Dessa forma, qualquer atividade que envolver as tabelas de mapeamento deve antes de tudo consultar a tabela de gerenciamento para localizar a tabela ativa.

A operação de escrita (*write*) é similar a de leitura. Sendo que a escrita ocorrerá na tabela de mapeamento reserva e quando todos os dados são gravados na tabela reserva o usuário realiza um novo acesso à tabela de gerenciamento para atualizar o nome da nova tabela que acabou de ser gerada. Como podemos perceber, com esta técnica a navegação é interrompida apenas no momento de atualizar dois campos relativos à tabela de mapeamento que acabou de ser gerado (o nome e o *timestamp*). Este é o único instante em que as operações de navegação são interrompidas, apenas uma simples e única operação de escrita.

No momento que ocorre a atualização da tabela de gerenciamento, a tabela de mapeamento ativa torna-se a reserva e a tabela de mapeamento reserva torna-se ativa. Entretanto, a tabela que agora é a reserva ainda está populada com os valores do antigo mapeamento. Este valores devem ser deletados para que um próximo mapeamento possa ser feito da mesma maneira que o especificado anteriormente. Isto é feito por um *thread* assíncrono em *background*.

Figura 6.5 Tabela de Mapeamento *versus* Tabela de Gerenciamento (Leitura).



1 - usuário consulta a tabela de gerenciamento para saber qual tabela de mapeamento está ativa

2- usuário recebe a informacao de qual tabela é a ativa

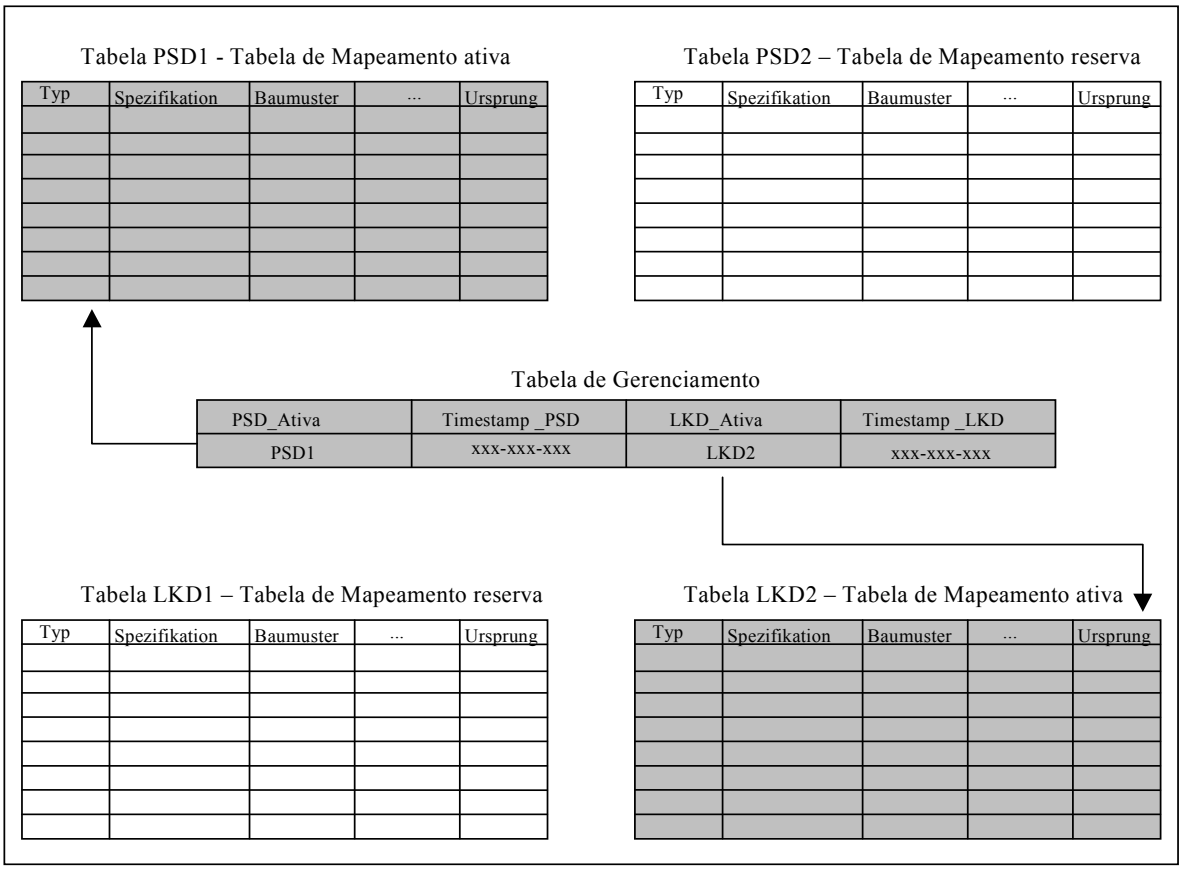
3 - usuário realiza a operacao de **leitura** da tabela de mapeamento (ativa)

6.6.6 Gerenciador de Tabelas de Mapeamento

A tarefa de atualizar o nome do novo mapeamento é do gerenciador de tabelas de mapeamento. Este módulo é capaz de informar aos usuários informações que podem ser requisitadas através da GUI como qual a tabela ativa e quando ocorreu o último mapeamento para um determinado banco de dados, ou seja, o dia e hora (*timestamp*) em que a tabela foi gerada. Estas informações o auxiliam na decisão de fazer ou não um novo mapeamento. Ainda mais importante que a informação ao usuário sobre a data do último mapeamento, é informar ao Atualizador de Consultas a tabela de mapeamento corrente (ver seção 6.6.7). Através desta informação, as consultas que envolvem tabelas de mapeamento são atualizadas no momento que são enviadas para o servidor de banco de dados, eliminando assim o problema de inconsistência da informação, já que o usuário pode gerar as tabelas de mapea-

mento a qualquer momento. O gerenciador de tabelas de mapeamento guarda a informação de qual tabela é a ativa numa relação no sistema *middleware* onde também se encontram as tabelas de mapeamento e pode ser visto na Figura 6.6.

Figura 6.6 Tabelas de Mapeamento e a Tabela de Gerenciamento.



Controle de Concorrência

No MENTAS, transações submetidas por vários usuários podem executar concorrentemente e podem acessar e atualizar os mesmos ítens nos bancos de dados. Mas se essa concorrência não for controlada, ela pode levar a problemas de inconsistência nos bancos de dados. As nossas tabelas de mapeamento são alvo de problemas causados pela concorrência. Dentre eles podemos destacar:

- Um usuário deseja gerar uma nova tabela de mapeamento antes da tabela antiga ser deletada.
- Dois usuários desejam gerar ao mesmo tempo um novo mapeamento para um dos bancos de dados.

No primeiro caso, a sincronização das tarefas deve acontecer de tal forma a garantir que apenas uma tabela de mapeamento possua valores armazenados por vez, de modo que a tabela reserva esteja sempre disponível para receber um novo mapeamento. Isso significa que deve ser garantida a remoção dos dados da tabela de mapeamento antiga, quando um novo mapeamento for criado. Para assegurar que apenas um usuário execute essas três operações -escrita do novo mapeamento, atualização da tabela de gerenciamento, e remoção dos dados da tabela de mapeamento antiga- utilizamos os *monitores*[1] a nível de implementação, de forma que, apenas o *thread* do usuário que obtiver o monitor será executado. Para garantir todas as funcionalidades necessárias à nossa aplicação, implementamos nossa própria classe de monitor. Para complementar, o método que realiza a remoção dos dados é interligado com o método que realiza a escrita na tabela de mapeamento através de um outro monitor, garantindo que não haverá problemas entre escrita e remoção dos dados, já que esta última é implementada através de um thread java. Para não prejudicar o paralelismo do banco de dados, atribuímos um monitor a cada banco de dados, de modo a garantir que dois mapeamentos sejam gerados simultaneamente para dois bancos de dados distintos, caso seja requisitado.

No momento em que ocorre a atualização da tabela de gerenciamento, o novo mapeamento está criado e o usuário que fez a requisição não deve ser penalizado com a espera da remoção dos dados da tabela antiga, que dependendo do tamanho, pode ser uma operação demorada. Assim, optamos por liberar o usuário assim que a tabela de gerenciamento é atualizada e realizar a operação de remoção dos dados da tabela antiga em *background*. Entretanto, o *thread* do usuário continua com a posse do monitor.

Caso dois usuários requisitem do sistema ao mesmo tempo a geração de um novo mapeamento para o mesmo banco de dados, damos a sensação aos dois de que seu pedido está sendo processado. Mas na realidade, só um dos usuários vai ter a posse do monitor para efetuar as operações sobre o banco de dados. Com isso, contribuímos para o aumento da performance do sistema, já que evitamos vários acessos

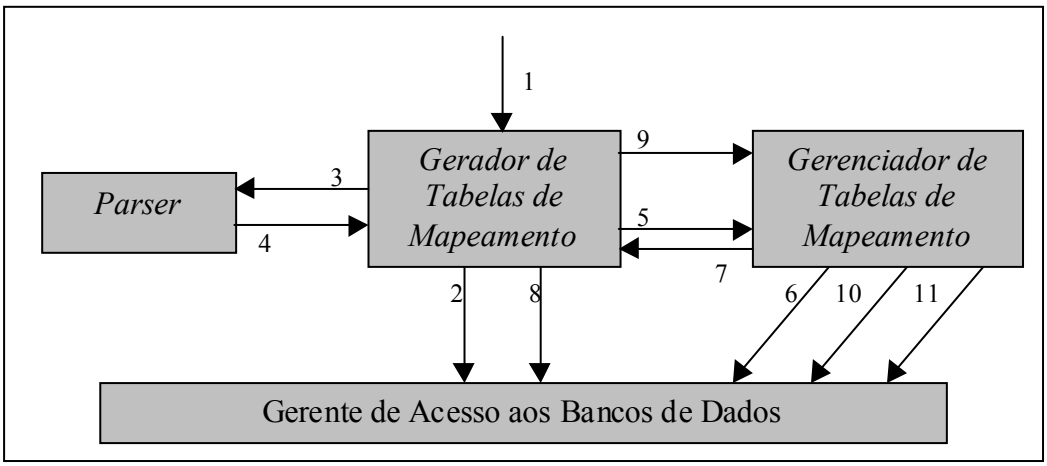
1. De uma forma geral, monitores garantem que em tempo de execução apenas um thread execute o trecho de programa que está sendo controlado por ele.

aos bancos de dados neste caso, além de que, se o primeiro usuário estiver no final da operação, o tempo de resposta ao segundo usuário será mais rápido.

O uso de monitores, caso não seja bem cuidado, pode introduzir o problema de *deadlock* no sistema, uma vez que um usuário pode manter a posse do monitor, tornando o processo de gerar novos mapeamentos inacessível aos demais usuários. Garantimos a total disponibilidade e confiabilidade do sistema assegurando a liberação do monitor caso algum problema ocorra com o usuário que o possui.

Na Figura 6.7, podemos ver o relacionamento dos componentes apresentados até então na ilustração de como é processado um pedido do usuário para gerar um novo mapeamento para um banco de dados - LKD ou PSD.

Figura 6.7 Relacionamento entre os Módulos do Monitor de Consistência no Processo de Geração de Tabela de Mapeamento.



Na Figura 6.7 suponha sempre a existência de um retorno por parte do gerente de Acesso aos Bancos de Dados, o qual não foi ilustrado a título de simplificação do desenho. A seguir, temos a descrição do fluxo das informações que atravessam os módulos:

1- Pedido do usuário, oriundo da GUI, para gerar um novo mapeamento.

2- Envio de uma consulta SQL ao servidor de BD para selecionar os identificadores de motor das fontes de dados remotas que serão passadas para o parser como informação, além de todos os demais pontos de entrada[1]. Como resposta, o Gerenciador de Acesso aos Bancos de Dados retorna os pontos de entrada.

3- Chamada ao parser passando os identificadores dos motores.

4- Retorno do parser, com os identificadores no formato que serão armazenados na tabela de mapeamento.

5- Pedido de informação ao Gerenciador de Tabelas de Mapeamento de qual tabela do banco de dados que está sendo gerado um novo mapeamento é a ativa.

6- Acesso ao servidor de BD através de uma consulta SQL que seleciona na tabela de gerenciamento o nome da tabela de mapeamento ativa para o banco de dados que está sendo requisitado um novo mapeamento.

7- Retorno ao Gerador de Tabelas de Mapeamento com o nome da tabela reserva, onde vai ser gerado o novo mapeamento.

8- Acessos ao servidor de BD para gravar os resultados do parser na tabela reserva. Com o retorno do Gerente de Acesso aos Bancos de Dados, o Gerador de Tabelas de Mapeamento deve informar ao Gerenciador de Tabelas de Mapeamento o nome da nova tabela em que foi gerado o mapeamento.

9- Passagem do nome da nova tabela de mapeamento bem como o novo timestamp que serão gravados na tabela de gerenciamento pelo Gerenciador das Tabelas de Mapeamento.

10- Acesso ao banco de dados para atualizar a tabela de gerenciamento com o nome e o timestamp da nova tabela de mapeamento.

11- Após atualizar os dados da tabela de gerenciamento e, portanto, tornar visível a todos os usuários o novo mapeamento, é então feito um novo acesso ao banco de dados para deletar os dados da tabela antiga, que passa a ser então a tabela reserva. Esse processo é feito completamente em *background*, liberando o usuário de aguardar o final da operação.

6.6.7 Atualizador de Consultas

Este módulo do monitor de consistência visa atualizar as consultas SQL com o nome da tabela de mapeamento ativa. Como a geração de novos mapeamentos é um processo dinâmico que pode acontecer em qualquer momento -dependendo

1. No caso do PSD, são necessárias sete consultas ao servidor de banco de dados para recuperar todas as informações: Três para acessar as fontes remotas, três para salvar os dados dessas subconsultas em tabelas intermediárias, e finalmente, uma última que integra os dados das tabelas intermediárias. Parte do resultado desta última consulta (o identificador do motor) é enviado ao Parser.

apenas que o usuário deseje fazê-lo-, conseqüentemente as consultas SQL que utilizam as tabelas de mapeamento devem verificar antes de acessar o servidor de bancos de dados qual é a tabela ativa, garantindo então a consistência das consultas SQL.

Na maioria das vezes, a tabela de mapeamento é transparente para o usuário. O usuário só tem uma forma de gerar consultas SQL onde as tabelas de mapeamento tornam-se não transparentes. Isto ocorre quando o usuário navega por entre os bancos de dados sem especificar explicitamente qualquer ponto de entrada na cláusula de condição da sua consulta. Uma vez escolhendo qualquer dos três atributos que identificam o motor, é inserido na consulta SQL apresentada ao usuário a tabela de mapeamento referente ao banco de dados.

No momento da montagem de qualquer consulta que envolva as tabelas de mapeamento, é dado um nome genérico para essas tabelas, como *PSDMapping* e *LKDMapping*, por exemplo. Antes de enviar a consulta para o servidor de banco de dados é feita uma busca nas consultas e se for encontrado qualquer destas duas seqüências atualizamos pelo nome da tabela de mapeamento corrente que é informada ao atualizador de consultas pelo gerente de tabelas de mapeamento. Considere a seguinte consulta montada pelo usuário:

SELECT PSD_Motor.Baustufe
FROM PSDMapping, PSD_Motor, LKDMapping, LKD_Motor
WHERE PSDMapping.Typ = LKDMapping.Typ **AND**
PSDMapping.Ursprung = PSD_Motor.typ **AND**
LKDMapping.Ursprung = LKD_Motor.Typ

Antes desta consulta ser enviada para o processamento no servidor de banco de dados, o atualizador de consultas faz a troca dos nomes genéricos utilizados para a tabela de mapeamento para o nome da tabela ativa. Supondo que a tabela de mapeamento ativa para o PSD seja Mapping_Motor_PSD2 e a tabela ativa para o LKD seja Mapping_Motor_LKD1 temos a seguinte consulta:

SELECT PSD_Motor.Baustufe
FROM Mapping_Motor_PSD2, PSD_Motor, Mapping_Motor_LKD1, LKD_Motor
WHERE Mapping_Motor_PSD2.Typ = Mapping_Motor_LKD1.Typ **AND**
Mapping_Motor_PSD2.Ursprung = PSD_Motor.typ **AND**
Mapping_Motor_LKD1.Ursprung = LKD_Motor.Typ

A consulta resultante da saída do atualizador de consultas pode ser finalmente enviada para o processamento. Isto não significa que o usuário precisa mudar o comportamento quando essas tabelas aparecerem diretamente na sua consulta apresentada pela GUI. Muito pelo contrário, apesar de não escondermos completamente do usuário a existência de tais tabelas, temos o cuidado de gerenciá-las completamente nos casos apresentados acima. Uma vez que essas tabelas não são restritas apenas ao uso interno do monitor de consistência, faz-se necessário que o

conector de interface também faça acesso ao atualizador de consultas antes da chamada ao servidor de banco de dados para a execução da consulta final do usuário.

6.6.8 Navegador

Este módulo representa a interface do monitor de consistência com os outros componentes da arquitetura apresentada no capítulo 3, sempre que faz-se presente uma navegação. Apenas este módulo e o gerador de tabelas de mapeamento recebem chamadas de componentes externos diretamente.

Uma navegação que envolve o identificador do motor pode ser realizada com ou sem verificação de consistência. Aqui não será abordado o caso em que existe a verificação de consistência, sendo este tópico a nossa próxima seção. Também, não é tarefa deste módulo a montagem das consultas quando a navegação ocorre sem a definição explícita de um valor na cláusula de condição da consulta, assunto que será abordado na seção 6.7.3. Resumindo, o Navegador possui as seguintes tarefas:

- Enviar a consulta formulada pelo usuário ao banco de dados corrente ao servidor de banco de dados e, dependendo da resposta, habilitar ou não o processo da navegação;
- Montar as consultas intermediárias durante o processo de navegação quando necessário;
- Montar a consulta final que será apresentada ao usuário após o processo da navegação;
- Convocar o Gerente de Consistência sempre que necessário para complementar as consultas intermediárias durante a navegação;

6.6.8.1 Vetar a Navegação

Sempre que o usuário deseja fazer uma navegação, o primeiro teste a ser feito é se a consulta SQL formulada por ele retorna resultados. Esta verificação é feita tanto se o usuário especificar valores para os pontos de entrada na cláusula de condição ou não. Na verdade, apenas em uma única situação não ocorre a verificação da consulta atual, no caso do usuário não especificar nenhum valor na cláusula de condição da sua consulta, pois consultas deste tipo sempre vão retornar valores. Se a consulta retornar pelo menos um resultado, a navegação prossegue sem maiores problemas. Caso contrário, o Navegador não inicializa o processo de navegação e retorna um código à GUI e esta o reporta para o usuário com um aviso que a consulta corrente não retorna nenhum resultado.

Esse procedimento evita trabalho desnecessário para o usuário. Se a consulta corrente não retorna resultados, a consulta posterior, após a navegação, também não

retornará. Paralisando o processo antes de ir para o outro banco de dados, damos a possibilidade para o usuário de reformular a consulta, se for o caso.

6.6.8.2 Montagem das Consultas Intermediárias

Quando um usuário especifica um identificador do motor na consulta SQL ao banco de dados corrente e deseja navegar para um outro banco de dados, alguns passos intermediários são necessários (Figura 6.8). Em primeiro lugar deve ser formatado o valor de entrada para o identificador de motor fornecido pelo usuário. Quando este encontra-se num formato comum aos identificadores armazenados nos bancos de dados é visto se o usuário deseja uma navegação seguindo os requisitos de consistência (seção 6.6.9) ou se a navegação ocorrerá de forma normal. A partir de então, são montadas as consultas intermediárias, para que sejam recuperados os motores no banco de dados para onde está seguindo a navegação que correspondem ao especificado pelo usuário na consulta corrente.

Existem dois tipos de consultas intermediárias: a primeira delas (Operação normal) é construída para recuperar no banco de dados posterior todos os motores que possuem a mesma especificação do usuário. Ou seja, são utilizados todos os resultados válidos[1] do parser na construção desta consulta. Como foi visto, os bancos de dados possuem seqüências diferentes para representar o mesmo motor em cada banco de dados. Portanto, na maioria das vezes essas consultas montadas com todas as especificações não retornam nenhum valor. Para resolver este problema, criamos o segundo tipo de consulta intermediária, chamada por nós de *other matches*. Estas consultas são um refinamento da primeira consulta. É montada uma consulta intermediária para cada resultado válido retornado pelo parser.

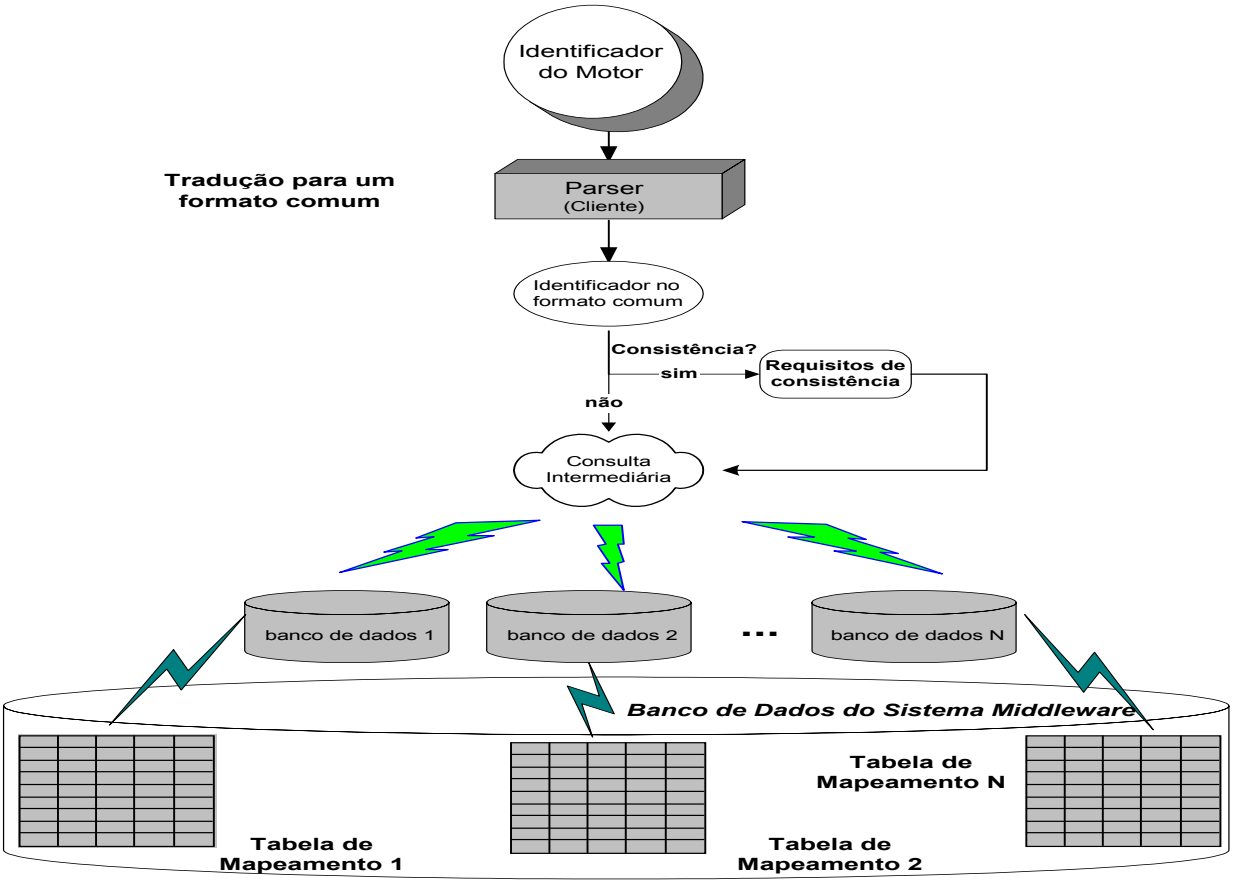
Operação Normal

Se a navegação não for vetada pelo passo anterior damos prosseguimento à navegação. Em primeiro lugar, é preciso que o motor especificado seja passado para o formato padrão de comparação quando o banco de dados corrente se tratar do LKD ou PSD. Como discutido anteriormente, o banco de dados Mentas já está dentro do padrão estabelecido.

Depois de passar o motor (ou os motores) especificado pelo usuário na cláusula de condição para o formato estabelecido, é montada a primeira consulta intermediária do processo da navegação, mostrada através da Consulta Intermediária 1.

1. O campo desconhecido retornado pelo parser não é considerado um resultado válido para a montagem das consultas.

Figura 6.8 Passos da Navegação com a Presença do Identificador do Motor.



Consulta Intermediária 1:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster
FROM EntidadeBDPosterior
WHERE EntidadeBDPosterior.Typ <comparação1> 'Tipo' **AND**
EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação' **AND**
EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção'

Onde:

EntidadeBDPosterior é o nome da entidade onde encontra-se o identificador do motor no banco de dados posterior. Se o banco de dados posterior fizer uso das tabelas de mapeamento (PSD e LKD), essa entidade será a tabela de mapeamento ativa no momento da execução da consulta. Quando se trata do banco de dados Mentas, é a própria entidade que contém o identificador do motor; Ainda se tratando do PSD e LKD os atributos *typ*, *spezifikation*, e *baumuster* são substituídos por *ursprung* que é o atributo ponteiro das tabelas de mapeamento para as fontes de dados originais.

<*comparaçãox*>, sendo x = 1, 2, 3, é o operador de comparação utilizado pelo usuário no momento da montagem da sua consulta ao banco de dados corrente, como =, !=, <, >, etc. Dessa forma garantimos que a busca ao banco de dados posterior é feita sob as mesmas características requeridas do banco de dados corrente. Lembramos que para o LKD e o PSD estes três campos são preenchidos pelo operador de comparação atribuído ao atributo Typ no banco de dados local, ou seja, <comparação1> = <comparação2> = <comparação3>, e que no Mentas os comparadores podem ser distintos, uma vez que o usuário faz a especificação separadamente para cada componente (tipo, especificação e modelo de construção).

Tipo pode ser originado de duas formas, dependendo de qual banco de dados é o corrente. No caso do Mentas este valor é o próprio valor fornecido pelo usuário no momento da formulação da cláusula de condição da consulta, ou seja, pode ou não existir na consulta intermediária, dependendo do que foi especificado pelo usuário. Em se tratando do LKD e PSD esse valor na maioria dos casos se faz presente nas consultas intermediárias, sendo originado através do Parser. O único caso em que o Tipo não está presente nas consultas intermediárias para esses dois bancos de dados é se o usuário especificar um motor que não se encaixe na gramática adotada pelo Parser, e este retorne apenas o campo desconhecido preenchido. Esse problema é resolvido da seguinte maneira:

Consulta Intermediária 2:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster
FROM EntidadeBDPosterior
WHERE EntidadeBDPosterior.Typ <comparação> 'Desconhecido'

Consulta Intermediária 3:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster
FROM EntidadeBDPosterior
WHERE EntidadeBDPosterior.Unerkannt <comparação> 'Desconhecido'

Como podemos perceber, pode ocorrer dois casos distintos. Se o banco de dados posterior for o Mentas a comparação é feita com o atributo que armazena o *tipo* já que este banco de dados não apresenta atributo correspondente para a seqüência desconhecido gerada pelo parser. Pensando em termos práticos, só existe uma possibilidade para isto acontecer, caso o usuário especifique um motor que não preencha sequer o atributo tipo, como 'M11' ou 'M11E23' por exemplo. Já se o banco de dados for o LKD ou o PSD, a comparação é feita com o próprio atributo das tabelas de mapeamento que guarda a informação de seqüências desconhecidas (*Unerkannt*). Na maioria das vezes, tipos de consultas como esta só vão retornar valores dependendo do comparador utilizado. Por exemplo, se o usuário especificar o igual "=" serão raras (para não dizer que nunca) as consultas que retornarão resultados, principalmente se envolver o banco de dados Mentas, não tendo nem a

possibilidade de *other matches* porque não terá o que procurar em separado. Ao contrário, os operadores diferente, menor e maior (!= , < , >) terão uma possibilidade muito maior de retorno de resultados.

Como podemos notar, a montagem das consultas intermediárias é um processo completamente dinâmico, que varia de acordo com os bancos de dados envolvidos na navegação e o valor fornecido pelo usuário para o identificador do motor. Caso não seja apenas o valor e sim os valores, a consulta é feita da mesma maneira, sendo estes formatados pelo Parser em caso necessário (PSD ou LKD) e anexados à **Consulta Intermediária 1** como mostrado abaixo[1]:

Consulta Intermediária 4:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster
FROM EntidadeBDPosterior
WHERE EntidadeBDPosterior.Typ <comparação1> 'Tipo(1)' **AND**
EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação(1)' **AND**
EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção(1)' **AND**
EntidadeBDPosterior.Typ <comparação4> 'Tipo(2)' ***AND***
EntidadeBDPosterior.Spezifikation <comparação5> 'Especificação(2)' ***AND***
EntidadeBDPosterior.Baumuster <comparação6> 'modelo de construção(2)' ***AND***
... ***AND***
EntidadeBDPosterior.Typ <comparação(x)> 'Tipo(n)' ***AND***
EntidadeBDPosterior.Spezifikation <comparação(x+1)> 'Especificação(n)' ***AND***
EntidadeBDPosterior.Baumuster <comparação(x+2)> 'modelo de construção(n)'

As consultas intermediárias apresentadas até o momento são responsáveis por procurar no banco de dados posterior por todos os motores que preenchem as características do informado ao banco de dados corrente. Existem dois fins previsíveis para isto: o primeiro é que seja encontrado o motor no outro banco de dados com as mesmas características informadas pelo usuário ao banco de dados corrente, e o segundo ocorre quando nenhum motor é selecionado no banco de dados posterior, sendo esse mais freqüente devido às diferenças de representações entre o identificador do motor nos bancos de dados. Por essa razão, é necessário oferecer ao usuário uma forma mais refinada na busca pelos motores correspondentes, e isto é feito através da opção *other matches* apresentada ao usuário logo após a falha da consulta anterior.

Operação de *Other Matches*

Este é o segundo tipo de consulta intermediária montada pelo navegador. O objetivo destas consultas é oferecer ao usuário uma consulta mais refinada ao banco de dados posterior levando em consideração os motores especificados por este na

1. Os valores acrescentados estão escritos em Itálico.

consulta ao banco de dados corrente. Uma nova consulta é originada para cada resultado válido do parser. De maneira geral, a consulta possui o seguinte formato quando se trata do LKD e PSD como banco de dados posterior:

Consulta Intermediária 5: `Tipo`

SELECT TabMapCorrente.Ursprung
FROM TabMapCorrente
WHERE TabMapCorrente.Typ <comparação> 'Tipo'

Consulta Intermediária 6: `Especificação`

SELECT TabMapCorrente.Ursprung
FROM TabMapCorrente
WHERE TabMapCorrente.Spezifikation <comparação> 'Especificação'

Consulta Intermediária 7: `Modelo de Construção`

SELECT TabMapCorrente.Ursprung
FROM TabMapCorrente
WHERE TabMapCorrente.Baumuster <comparação> 'modelo de construção'

E as próximas, quando se tratar do Mentas:

Consulta Intermediária 8: `Tipo`

SELECT Entidade.Typ, Entidade.Spezifikation, Entidade.Baumuster
FROM Entidade
WHERE Entidade.Typ <comparação> 'Tipo'

Consulta Intermediária 9: `Especificação`

SELECT Entidade.Typ, Entidade.Spezifikation, Entidade.Baumuster
FROM Entidade
WHERE TabMapCorrente.Spezifikation <comparação> 'Especificação'

Consulta Intermediária 10:[1] `Modelo de Construção`

SELECT Entidade.Typ, Entidade.Spezifikation, Entidade.Baumuster
FROM Entidade
WHERE TabMapCorrente.Baumuster <comparação> 'modelo de construção'

Como podemos perceber, existe diferença na montagem das consultas de acordo com o banco de dados posterior, com o tipo de operador de comparação utilizado e com o valor especificado para o banco de dados corrente. Através das consultas,

1. Nas consultas intermediárias de 1 a 4 existe a mesma diferença apresentada nas consultas intermediárias de 5 a 9. Ou seja, no Mentas deve ser projetado os atributos *Typ, Spezifikation e Baumuster* como escrito. No LKD e PSD deve ser projetado o atributo *ursprung* das tabelas de mapeamento.

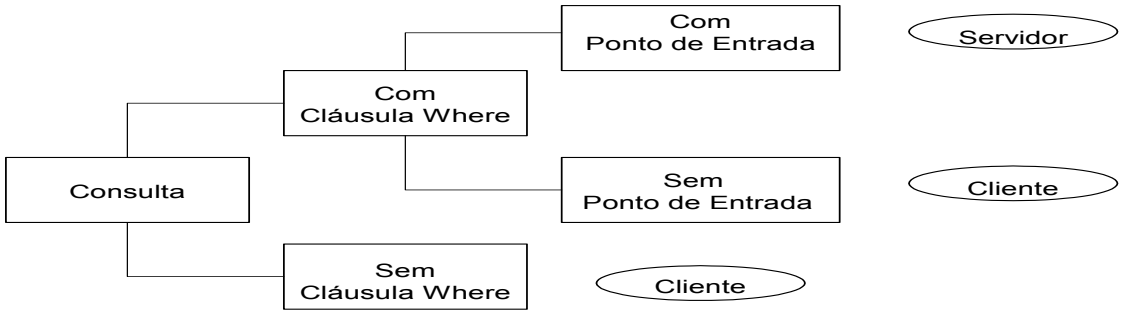
vemos que a procura agora é feita separadamente a cada parte que compõe o identificador do motor, de forma que para cada identificador especificado na cláusula de condição podem ser geradas até três consultas intermediárias, dependendo mais uma vez do resultado do Parser (LKD e PSD) ou dos atributos especificados na consulta corrente (Mentas). Os resultados de cada consulta são armazenados em um vetor e retornado a GUI que os apresenta ao usuário.

Informação dos Novos Componentes da Consulta a GUI

Outra tarefa do Navegador é definir as regras para guiar a GUI no processo de montagem das consultas após uma navegação. São regras simples, mas que devem ser observadas no momento da adição dos novos parâmetros à consulta do usuário.

Consultas do usuário que possuem cláusula de condição sempre são verificadas antes de uma navegação. Dependendo do conteúdo desta cláusula, a parte complementar da consulta originada pela navegação é montada no cliente ou no servidor (Figura 6.9). No caso de não existir cláusula de condição na consulta sempre será montada a complementação no cliente, já que não é necessário testar a consulta corrente e conseqüentemente, não é necessário uma comunicação entre cliente e servidor.

Figura 6.9 Montagem das Consultas.



Dessa forma, existe a seguinte regra para a criação da complementação da consulta do usuário no final de uma navegação. Para a montagem dessa consulta, recebemos da GUI as informações necessárias sobre a navegação, como os bancos de dados envolvidos e os pontos de entrada definidos.

Consulta de Navegacao 1:

Cláusula de condição:

EntidadePEntradaBDPosterior.PEntradaBDPosterior <comparação> valor(EntidadePEntradaBDCorrente.PEntradaBDCorrente)

Tabelas:

EntidadePEntradaBDPosterior

Onde:

EntidadePEntradaBDPosterior.PEntradaBDPosterior é inferido através da informação do ponto de entrada especificado na consulta e as informações da Tabela 6.3;

<*comparação*>, corresponde ao operador escolhido pelo usuário para o banco de dados corrente e;

valor(EntidadePEntradaBDCorrente.PentradaBDCorrente) corresponde ao valor fornecido pelo usuário ao ponto de entrada do banco de dados corrente, transformado para o formato do banco de dados para onde está ocorrendo a navegação. Caso haja mais de um ponto de entrada especificado numa mesma consulta, ou ainda, o mesmo ponto de entrada com mais de um valor, essa regra deve ser empregada para cada ocorrência.

As informações sobre o banco de dados corrente são recebidas através da GUI e os correspondentes no banco de dados posterior (*EntidadePEntradaBDPosterior.PEntradaBDPosterior*) são inferidos utilizando-se as informações contidas na Tabela 6.3, para cada par enviado pela GUI sobre o banco de dados corrente. Lembramos aqui que o usuário pode escolher mais de um ponto de ligação entre os bancos de dados, caso a entidade escolhida para efetuar a navegação os contenha, sendo o tratamento uniforme para todos os casos. Além da parte da consulta que será acrescentada na cláusula de condição do usuário, são devolvidos também para a GUI os nomes das tabelas envolvidas.

Para ilustrar como é utilizada a Consulta de Navegação 1, considere o seguinte exemplo: Um usuário monta uma consulta ao banco de dados PSD utilizando o ponto de entrada *Durchmesser* igual ao valor 35.8. Considere ainda que a navegação é feita através da entidade que possui o ponto de entrada *Durchmesser* (*PSD_Zylinder*). Para fechar o ambiente de navegação, suponha que o usuário escolheu como banco de dados posterior o LKD. Assim, tendo como consulta inicial a seguinte,

SELECT PSD_Zylinder.Atributo1, PSD_Zylinder. Atributo2, ..., PSD_Zylinder.AtributoN
FROM PSD_Zylinder
WHERE PSD_Zylinder.Durchmesser = 35.8

Com a informação que o usuário escolheu o ponto de entrada da entidade para efetuar a navegação (*PSD_Zylinder.Durchmesser*) passadas ao monitor de consistência pela GUI, este monta a seguinte consulta de navegação:

Cláusula de condição:

LKD_Motor.Bohrung = ’35,8’

Tabelas:

LKD_Motor

Esta parte da consulta é retornada à GUI que adiciona as informações à consulta corrente do usuário, transformando-a na seguinte consulta (as modificações estão em itálico):

SELECT PSD_Zylinder.Atributo1, PSD_Zylinder. Atributo2, ..., PSD_Zylinder.AtributoN
FROM PSD_Zylinder, *LKD_Motor*
WHERE PSD_Zylinder.Durchmesser = 35.8 ***AND***
LKD_Motor.Bohrung = ’35,8’

Se houver identificador do motor definido na condição da consulta, apesar da regra ser a mesma definida acima, a diferença encontra-se em como é conseguido o *valor(EntidadePEntradaBDCorrente.PEntradaBDCorrente)*. Como sabemos, o mapeamento dos valores do identificador do motor não é feito automaticamente, e é através da escolha dos motores nos passos intermediários (através das consultas intermediárias que foram apresentadas anteriormente - tanto no caso normal, quanto de *other matches*), que este campo é preenchido. Quando o usuário escolhe mais de um valor de mapeamento para um único valor fornecido, deve ser seguida a seguinte regra para atualizar a consulta corrente após a navegação:

Identificador do Motor - Mapeamento um para muitos valores

Consulta de Navegacao 2:

Cláusula de condição:

(EntidadeCorrespondente.AtributoCorrespondente <comparação> valor1(EntidadeCorrente.AtributoCorrente) OR
EntidadeCorrespondente.AtributoCorrespondente <comparação> valor2(EntidadeCorrente.AtributoCorrente) OR
... OR
EntidadeCorrespondente.AtributoCorrespondente <comparação> valorN(EntidadeCorrente.AtributoCorrente))

Tabelas:

EntidadeCorrespondente

Mais uma vez, utilizamos o exemplo de uma navegação para ilustrar como é utilizada a Consulta de Navegação 2. Suponha a navegação entre o PSD e o LKD utilizando desta vez o identificador de motor como ponto de entrada.

SELECT PSD_Motor.Atributo1, PSD_Motor. Atributo2, ..., PSD_Motor.AtributoN
FROM PSD_Motor
WHERE PSD_Motor.Typ = ’M111E18MMX’

Suponha que o usuário escolheu três motores dos resultados apresentados pelas consultas intermediárias, sendo eles: M111.920, M111.921 e M111.940. Dessa forma, a consulta final será:

SELECT PSD_Motor.Atributo1, PSD_Motor. Atributo2, ..., PSD_Motor.AtributoN, LKD_Motor.Typ
FROM PSD_Motor, LKD_Motor
WHERE PSD_Motor.Typ = 'M111E18MMX' **AND**
(LKD_Motor. Typ = 'M111.920' **OR** LKD_Motor. Typ = 'M111.921' **OR** LKD_Motor. Typ = 'M111.940')

Convocar o Gerente de Consistência

Uma parte da responsabilidade do mapeamento entre os bancos de dados, quando o ponto de entrada escolhido para realizar uma navegação é o identificador do motor, é delegada ao usuário. No entanto, o usuário pode optar por restringir o número de resultados intermediários. Isso é possível com a escolha de realizar a navegação com a verificação de consistência. É tarefa do Navegador convocar o Gerente de Consistência para que seja acrescentada à consulta intermediária todo o processo referente a checagem de consistência (Figura 6.8).

6.6.9 Gerente de Consistência

A tarefa deste módulo é montar e acrescentar parte da consulta intermediária, no caso do usuário ter habilitado a opção de realizar a navegação com a verificação de consistência (Figura 6.8).

Devido à heterogeneidade semântica e estrutural nos identificadores de motor, nos bancos de dados integrados, não é feito o mapeamento direto dos valores fornecidos pelo usuário de um banco de dados para outro, como acontece com os demais pontos de entrada do nosso sistema. Por exemplo, o motor M111E18MMX armazenado no banco de dados PSD corresponde ao motor M111.921 no banco de dados LKD e ao motor com o tipo igual a M111, especificação igual a E18 e o modelo de construção igual a 921 no banco de dados Mentas.

Uma parte da responsabilidade de mapear o motor correspondente no outro banco de dados é delegada ao usuário. As consultas intermediárias foram projetadas de forma a resgatar do banco de dados posterior apenas os motores que se enquadrem nas especificações fornecidas pelo usuário ao identificador do motor do banco de dados corrente. Quando a operação normal não consegue recuperar nenhum motor (que acontece com mais normalidade), disparamos a montagem das consultas intermediárias que procuram o motor não por todas as características informadas, e sim por cada uma separadamente (consultas intermediárias para operação de *other matches*). Os resultados recuperados por essas consultas, apesar de apresentar fa-

talmente o motor correspondente ao fornecido pelo usuário, vai requerer um maior conhecimento deste.

Por exemplo, suponha que o usuário especificou o motor M111E18MMX no banco de dados PSD e queira informações referentes a este motor no banco de dados LKD. A consulta intermediária (operação normal) não conseguirá êxito, uma vez que não existe no banco de dados LKD nenhum motor que preencha os requisitos desta consulta (com tipo igual a M111 e Especificação igual a E18). Será necessário então a criação das consultas intermediárias para operação de *other matches*. A execução dessas consultas por sua vez recuperam 26 motores no banco de dados LKD que possuem o tipo igual a M111 e nenhum motor que possue a Especificação igual a E18. Dentre os valores recuperados está o motor M111.921, que realmente corresponde no LKD ao motor que foi fornecido no PSD. Entretanto, o usuário deve ter conhecimento dessa informação, ou então, escolher todos os motores apresentados, que nem sempre é uma boa opção.

A verificação de consistência é feita de forma a restringir o número de motores especificados através da imposição de algumas restrições. Um motor é dito correspondente a outro, de acordo com a nossa verificação de consistência, quando este apresenta as mesmas especificações, e ainda, possui os valores iguais aos atributos dos pontos de entrada. Mas, usuários diferentes possuem também necessidades diferentes quando buscam informações de dois bancos de dados distintos. Assim, um atributo que é importantíssimo para um usuário, pode ser completamente indiferente ou mesmo indesejável para outro.

Percebemos que não seria adequado impor ao usuário que para um motor de um banco de dados corresponder a outro em um segundo banco de dados, todos os pontos de entradas relativos a este motor devam ser iguais. Em vez disso, devemos oferecer ao usuário a possibilidade de especificar por quais pontos de entrada ele deseja que seja feita a verificação de consistência.

Foi pensando nessa possibilidade que implementamos a verificação de consistência do MENTAS. O usuário, além de especificar se deseja que sua navegação ocorra dentro dos padrões de consistência, pode também especificar quais pontos de entrada devem ser analisados. Ainda, oferecemos a possibilidade de escolha dos pontos de entrada em relação aos pares de bancos de dados por onde pode ocorrer a navegação. Dessa forma, se o usuário sabe que um determinado ponto de entrada não merece confiança em um banco de dados ou simplesmente não lhe interessa, este pode ser ignorado no momento da comparação por este usuário.

Os pontos de entrada que são analisados entre os bancos de dados durante a navegação para a verificação de consistência só precisam ser definidos uma única vez pelo usuário, os quais são salvos no banco de dados do sistema *middleware*, conjuntamente com outras informações do usuário. No momento da inicialização do sistema, essas informações são buscadas no banco de dados. De acordo com essas

opções, é que montamos a parte da consulta intermediária referente ao processo de consistência.

A idéia básica para a verificação de consistência é comparar os pontos de entrada referentes ao motor fornecido para o banco de dados corrente com os pontos de entrada referentes ao motor correspondente no banco de dados posterior. Sempre levando em consideração as opções escolhidas pelo usuário para quais pontos de entrada devem ser considerados de acordo com o par de banco de dados no qual está acontecendo a navegação.

Portanto, a partir do motor fornecido pelo usuário para o banco de dados corrente, devemos recuperar os pontos de entrada desse motor, e a cada ocorrência de um motor que possua mesma identificação (tipo, especificação e modelo de construção) no banco de dados posterior deve ser feita também a comparação dos pontos de entrada.

IdentificadorMotorBDCorrente = IdentificadorMotorBDPosterior ***e***
PEntradasIdentMotorBDCorrente = PEntradasIdentMotorBDPosterior

Montagem das Consultas Intermediárias

Para uma melhor visualização de como a consulta é alterada por este módulo, apresentamos aqui a consulta sem conferência da consistência (produzida pelo navegador) e, logo a seguir, a nova consulta produzida pelo Gerenciador de consistência. Para diferenciar as partes acrescentadas na consulta modificada pelo gerente de consistência, elas são escritas em *itálico*.

Operação Normal

- Consulta gerada pelo Navegador:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster
FROM EntidadeBDPosterior
WHERE EntidadeBDPosterior.Typ <comparação1> 'Tipo' AND EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação' AND EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção'

- Consulta modificada pelo Gerente de Consistência:

Consulta Intermediária 11:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster

FROM EntidadeBDPosterior, *EntidadeBDCorrente, EntidadePEntradaBDC1,..., EntidadePEntradaBDCn, EntidadePEntradaBDP1,..., EntidadePEntradaBDPn*
WHERE EntidadeBDPosterior.Typ <comparação1> 'Tipo' *AND EntidadeBDCorrente.Typ <comparação1> 'Tipo'* AND
EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação' *AND EntidadeBDCorrente.Spezifikation <comparação2> 'Especificação'* AND
EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção' *AND EntidadeBDCorrente.Baumuster <comparação3> 'modelo de construção' AND*
EntidadePEntradaBDC1.PEntradaBDC1 = EntidadePEntradaBDP1.PEntradaBDP1 AND... AND
EntidadePEntradaBDCn.PEntradaBDPn = EntidadePEntradaBDPn.PEntradaBDSn

Onde:

EntidadeBdCorrente é o nome da entidade onde encontra-se o identificador do motor no banco de dados corrente. Se o banco de dados posterior fizer uso das tabelas de mapeamento (PSD e LKD), essa entidade será a tabela de mapeamento ativa no momento da execução da consulta. Quando se trata do banco de dados Mentas, é a própria entidade que contém o identificador do motor.

EntidadePEntradaBDC1 é o nome da entidade onde encontra-se o primeiro ponto de entrada para o banco de dados corrente (BDC) que será comparado entre os bancos de dados na hora da recuperação do motor. Na hora da montagem dessa consulta é necessário saber as opções de consistência definidas pelo usuário para o dado par de banco de dados envolvido na navegação. Assim como a EntidadeBdCorrente, se o banco de dados posterior fizer uso das tabelas de mapeamento (PSD e LKD), essa entidade é a tabela de mapeamento ativa no momento da execução da consulta. Quando se trata do banco de dados Mentas, é a entidade que contém o primeiro ponto de entrada definido para este banco de dados de acordo com as opções do usuário.

Assim como *EntidadePEntradaBDC1*, são adicionadas à consulta todas as demais entidades que possuem pontos de entrada que devem ser analisados no momento da navegação. Ou seja, *EntidadePEntradaBDC2* é a entidade onde encontra-se o segundo ponto de entrada marcado pelo usuário para ser usado no processo de comparação entre os bancos de dados, e assim sucessivamente, até o último ponto de entrada definido pelo usuário para o par de banco de dados onde está ocorrendo a navegação. O mesmo ocorre com *EntidadePEntradaBDP1* até *EntidadePEntradaBDPn,* sendo que neste caso trata-se da entidade do banco de dados posterior.

A presença na cláusula da condição dos atributos do motor para o banco de dados corrente (*EntidadeBDCorrente.Typ <comparação1> 'Tipo'*, *EntidadeBDCorrente.Spezifikation <comparação2> 'Especificação'*, *EntidadeBDCorrente.Baumuster <comparação3> 'modelo de construção'*) é necessária, pois devem ser pegos os pontos de entrada correspondentes a estes motores. Com relação ao PSD e LKD, todos os pontos de entrada estão localizados na tabela de mapeamento, e dessa forma (acrescentando a comparação ao identificador do motor no banco de dados corrente), conseguimos fazer a comparação com os pontos de entrada do

motor fornecido. O mesmo acontece no caso do Mentas, que também possui pontos de entrada localizados na mesma entidade (e conseqüentemente na mesma relação) que o identificador do motor. Portanto, a presença destas comparações, está diretamente associada à recuperação dos pontos de entrada para o motor fornecido ao banco de dados corrente.

Para cada identificador de motor presente na consulta do usuário, é necessário recuperar o identificador do banco de dados corrente. Assim, uma consulta que possua vários valores para o identificador do motor deve ser montada como mostrado abaixo:

- Consulta gerada pelo Navegador:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster
FROM EntidadeBDPosterior
WHERE EntidadeBDPosterior.Typ <comparação1> 'Tipo(1)' **AND**
EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação(1)' **AND**
EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção(1)' **AND**
EntidadeBDPosterior.Typ <comparação1> 'Tipo(2)' **AND**
EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação(2)' **AND**
EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção(2)' **AND**
... **AND**
EntidadeBDPosterior.Typ <comparação1> 'Tipo(n)' **AND**
EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação(n)' **AND**
EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção(n)'

- Consulta modificada pelo Gerente de Consistência:

Consulta Intermediária 12:

SELECT EntidadeBDPosterior.Typ, EntidadeBDPosterior.Spezifikation, EntidadeBDPosterior. Baumuster
FROM EntidadeBDPosterior, *EntidadeBDCorrente, EntidadePEntradaBDC1,..., EntidadePEntradaBDCn, EntidadePEntradaBDP1,..., EntidadePEntradaBDPn*
WHERE EntidadeBDPosterior.Typ <comparação1> 'Tipo(1)' *AND*
EntidadeBDCorrente.Typ <comparação1> 'Tipo(1)' AND
EntidadeBDPosterior.Spezifikation <comparação2> 'Especificação(1)' *AND*
EntidadeBDCorrente.Spezifikation <comparação2> 'Especificação(1)' AND
EntidadeBDPosterior.Baumuster <comparação3> 'modelo de construção(1)' *AND*
EntidadeBDCorrente.Baumuster <comparação3> 'modelo de construção(1)' AND
EntidadeBDPosterior.Typ <comparação4> 'Tipo(2)' AND
EntidadeBDCorrente.Typ <comparação4> 'Tipo(2)' AND
EntidadeBDPosterior.Spezifikation <comparação5> 'Especificação(2)' *AND*
EntidadeBDCorrente.Spezifikation <comparação5> 'Especificação(2)' AND
EntidadeBDPosterior.Baumuster <comparação6> 'modelo de construção(2)' *AND*
EntidadeBDCorrente.Baumuster <comparação6> 'modelo de construção(6)'
AND... AND
EntidadeBDPosterior.Typ <comparaçãox> 'Tipo(n)' *AND*
EntidadeBDCorrente.Typ <comparaçãox> 'Tipo(n)' AND
EntidadeBDPosterior.Spezifikation <comparação(x+1)> 'Especificação(n)' *AND*
EntidadeBDCorrente.Spezifikation <comparação(x+1)> 'Especificação(n)' AND
EntidadeBDPosterior.Baumuster <comparação(x+2)> 'modelo de construção(n)' *AND*
EntidadeBDCorrente.Baumuster <comparação(x+2)> 'modelo de construção(n)' AND
EntidadePEntradaBDC1.PEntradaBDC1 = EntidadePEntradaBDP1.PEntradaBDP1

AND... AND
EntidadePEntradaBDCn.PEntradaBDPn = EntidadePEntradaBDPn.PEntradaBDSn

Operação de *Other Matches*

Se as consultas anteriores não retornarem resultado algum, damos ao usuário a opção de prosseguir a navegação através da opção de *other matches*, sendo que da mesma maneira, faremos a comparação dos pontos de entrada definidos entre os bancos de dados envolvidos na navegação. Apresentaremos aqui, as consultas geradas pelo navegador, e a seguir, a consulta alterada pelo gerente de consistência. De uma forma geral, as consultas são modeladas da seguinte maneira:

Consulta Intermediária 13: `Tipo`

SELECT EntidadeBDPosterior.IdentificadorMotor
FROM EntidadeBDPosterior, *EntidadeBDCorrente, EntidadePEntradaBDC1,..., EntidadePEntradaBDCn, EntidadePEntradaBDS1,..., EntidadePEntradaBDSn*
WHERE EntidadeBDPosterior.Typ <comparação> 'Tipo' *AND*
EntidadeBDCorrente.Typ <comparação> 'Tipo' AND
EntidadePEntradaBDC1.PEntradaBDC1 = EntidadePEntradaBDS1.PEntradaBDS1 AND
... AND
EntidadePEntradaBDCn.PEntradaBDSn = EntidadePEntradaBDSn.PEntradaBDSn

Consulta Intermediária 14: `Especificação`

SELECT EntidadeBDPosterior.IdentificadorMotor
FROM EntidadeBDPosterior, *EntidadeBDCorrente, EntidadePEntradaBDC1,..., EntidadePEntradaBDCn, EntidadePEntradaBDS1,..., EntidadePEntradaBDSn*
WHERE EntidadeBDPosterior.Spezifikation <comparação> 'Especificação' **AND**
EntidadeBDCorrente.Spezifikation <comparação> 'Especificação' **AND**
EntidadePEntradaBDC1.PEntradaBDC1 = EntidadePEntradaBDS1.PEntradaBDS1 **AND**
... **AND**
EntidadePEntradaBDCn.PEntradaBDSn = EntidadePEntradaBDSn.PEntradaBDSn

Consulta Intermediária 15: `Modelo de Construção`

SELECT EntidadeBDPosterior.IdentificadorMotor
FROM EntidadeBDPosterior, *EntidadeBDCorrente, EntidadePEntradaBDC1,..., EntidadePEntradaBDCn, EntidadePEntradaBDS1,..., EntidadePEntradaBDSn*
WHERE EntidadeBDPosterior.Baumuster <comparação> 'modelo de construção' **AND**
EntidadeBDCorrente.Baumuster <comparação> 'modelo de construção' **AND**
EntidadePEntradaBDC1.PEntradaBDC1 = EntidadePEntradaBDS1.PEntradaBDS1 **AND**
... **AND**
EntidadePEntradaBDCn.PEntradaBDSn = EntidadePEntradaBDSn.PEntradaBDSn

Onde:

EntidadeBDPosterior.IdentificadorMotor, quando se tratar do LKD e PSD, será representada pela tabela de mapeamento corrente para estes bancos de

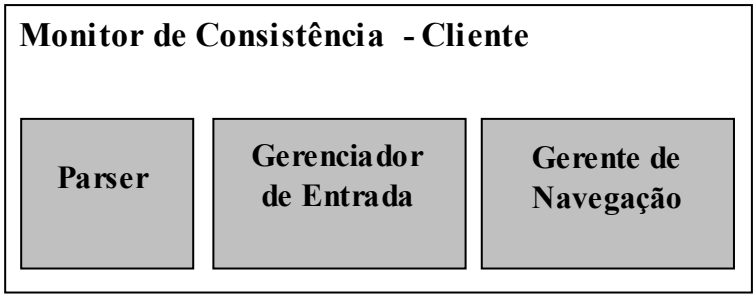
dados, seguida do atributo que representa o identificador do motor (ursprung). Para o Mentas, é a própria entidade que possui o identificador do motor, e são recuperados os três atributos que em conjunto representam o identificador (tipo, especificação e modelo de construção).

Da mesma forma que para as consultas normais, é preciso inserir na cláusula de condição a recuperação do valores para o banco de dados corrente (*Entidade-BDCorrente.Typ <comparação> 'Tipo'*), a fim de que sejam recuperados os pontos de entrada relativos a essa condição.

6.7 Monitor de Consistência do Cliente - MCC

Uma das tarefas do MCC é formatar os dados fornecidos pelo usuário através da GUI. Dessa forma, em um trabalho conjunto, o Monitor de Consistência do Cliente formata os dados de entrada, enquanto o Monitor de Consistência do Servidor formata os dados das fontes remotas (seção 6.6). Abaixo podemos ver que o MCC é composto de três módulos: Parser, Gerenciador de Entrada e Gerente de Navegação (Figura 6.10).

Figura 6.10 Arquitetura do Monitor de Consistência do Cliente.



6.7.1 Parser

Possui a mesma funcionalidade do parser presente no servidor, sendo que o cliente não está envolvido com a criação das tabelas de mapeamento e sim com a formação dos dados de entrada dos usuários quando tratar-se do identificador do motor. Quando o usuário deseja fazer a navegação entre os bancos de dados e especifica um valor para o identificador do motor na sua consulta, a primeira atuação do monitor de consistência, caso ele esteja no PSD ou LKD, é decompor o valor de entrada. Caso o usuário esteja no BD Mentas e for navegar para o PSD ou LKD, não

será necessário a chamada ao parser uma vez que os valores de entrada já estão no formato adequado para a realização da navegação.

Optamos por duplicar essa função no cliente para poupar uma comunicação entre o cliente e o servidor. O parser do cliente não precisa fazer acesso aos bancos de dados já que sua única função é formatar para o modelo padrão os dados de entrada do usuário.

Se a GUI detectar o uso do identificador do motor como ponto de entrada, ela aciona o Monitor de Consistência do Cliente passando um ou mais valores deste atributo que foram especificados na cláusula de condição. Com os resultados do MCC, ou seja, com os motores especificados pelo usuário obedecendo o formato tipo, especificação e modelo de construção, é montada uma consulta intermediária para acessar as tabelas de mapeamento, ou para acessar o Mentas, dependendo de como está ocorrendo a navegação.

6.7.2 Gerenciador de Entradas

Outra tarefa importante realizada pelo MCC é a formatação dos valores de entrada informados pelos usuários ao sistema, de forma que seja totalmente transparente para o usuário as diferenças de representações nos valores armazenados nos bancos de dados. Essa tarefa é realizada pelo módulo Gerenciador de Entrada (Figura 6.10). No cliente são feitas as modificações necessárias à adequação dos valores da consulta para o banco de dados corrente, independente de haver navegação ou não. Como veremos posteriormente, ao contrário, o MCS é responsável pela formatação dos valores de acordo com o banco de dados para o qual está ocorrendo a navegação, de modo a torná-la possível. Dessa forma, os trabalhos são complementados.

Para ilustrar a atuação do gerenciador de entradas, podemos imaginar que um usuário do LKD, acostumado à notação da vírgula como caracter separador de casas decimais, resolve consultar o banco de dados PSD, o qual possui outra notação. Não devemos exigir do usuário que este tenha conhecimento de todas as diferenças de notações existentes entre os bancos de dados. Por isso, cuidamos para que essas diferenças sejam totalmente escondidas no momento da montagem da consulta no MENTAS. Portanto, no nosso exemplo, caso o usuário especifique um valor numa consulta ao banco de dados PSD utilizando a notação do LKD, este valor será automaticamente mapeado para o usuário.

Para realizar esta operação, o monitor de consistência recebe todas as informações necessárias tanto da GUI (nome do atributo e valor), como do Conector de Interface que fornece informações sobre os meta-dados dos bancos de dados integrados.

6.7.3 Gerente de Navegação

A terceira e última funcionalidade apresentada pelo MCC é a montagem das consultas no caso da navegação sem a especificação explícita de valores na cláusula da condição. Consultas que não possuem especificação na cláusula de condição não são verificadas, já que neste caso, sempre haverá retorno de valores[1]. Dessa forma, evitamos uma consulta do cliente ao servidor, poupando a comunicação.

Como já foi apresentado, nesse caso, o usuário é obrigado a especificar pelo menos um ponto de entrada através do qual deseja realizar a navegação (através da janela apresentada pela Figura 5.14). Daí, a montagem das consultas é feita de forma dinâmica, e para isso, a GUI passa todas as informações necessárias. Dentre estas informações, encontram-se o banco de dados corrente, a entidade pela qual o usuário está realizando a consulta, e os pontos de entrada definidos como ligação (escolhidos através da janela apresentada na Figura 5.16).

De uma forma geral, levando-se em consideração o banco de dados de onde está partindo a navegação (corrente) e o banco de dados posterior, podemos considerar que as consultas são montadas da seguinte forma[2]:

Consulta de Navegacao 3:

Cláusula de condição:

EntidadePEntradaBDCorrente.PEntradaBDCorrente = EntidadePEntradaBDCorrespondente.PEntradaBDCorrespondente

Tabelas:

EntidadePEntradaBDCorrente

EntidadePEntradaBDCorrespondente

Onde:

EntidadePEntradaCorrente é o nome da entidade a qual pertence o ponto de entrada escolhido para que ocorra a navegação.

PEntradaBDCorrente é o nome do ponto de entrada especificado pelo usuário.

1. Consultas que possuem apenas cláusula de projeção sempre retornam valores se a(s) tabela(s) envolvida(s) contiver(em) valores. No MENTAS, como todas as tabelas já estão populadas temos a certeza que algum resultado vai ser retornado por essas consultas.
2. Apresentamos somente a parte da consulta gerada pelo gerente de navegação, a qual é acrescida à cláusula *de condição* da consulta do usuário, originando a consulta final após a navegação.

EntidadePEntradaCorrespondente é o nome da entidade a qual pertence o ponto de entrada correspondente ao escolhido pelo usuário no banco de dados corrente.

PEntradaBDCorrespondente é o nome do ponto de entrada correspondente no banco de dados posterior ao especificado pelo usuário no banco de dados corrente.

As informações sobre o banco de dados corrente (*EntidadePEntradaCorrente*, *PEntradaBDCorrente*) são informadas ao MCC através da GUI e os correspondentes no banco de dados posterior (*EntidadePEntradaCorrespondente* e *PEntradaBDCorrespondente*) são inferidos utilizando-se as informações contidas na Tabela 6.3. Para cada par enviado pela GUI sobre o banco de dados corrente (*EntidadePEntradaCorrente*, *PEntradaBDCorrente*) é criada (e retornada à GUI) a parte da consulta como mostrado pela Consulta de Navegação 3. Lembramos aqui que o usuário pode escolher mais de um ponto de ligação entre os bancos de dados, caso a entidade escolhida para efetuar a navegação os contenha, sendo o tratamento uniforme para todos os casos. Além da parte da consulta que será acrescentada na cláusula de condição do usuário, são devolvidos também para a GUI os nomes das tabelas envolvidas.

Para ilustrar como é utilizada a Consulta de Navegação 3, considere o seguinte exemplo: Um usuário monta uma consulta ao banco de dados PSD apenas com cláusula de projeção, não tendo portanto definido nenhum ponto de entrada para efetuar a navegação. Considere ainda que este usuário definiu a projeção sobre a entidade *PSD_Zylinder* a qual apresenta dois pontos de entrada: *Durchmesser* e *Hubvolumen.* Considere ainda que a navegação é feita através desta mesma entidade e que o usuário deseja que ocorra através dos dois pontos. Para fechar o ambiente de navegação, suponha que o usuário escolheu como banco de dados posterior o banco de dados LKD. Assim, tendo como consulta inicial a seguinte,

SELECT PSD_Zylinder.Atributo1, PSD_Zylinder. Atributo2, ..., PSD_Zylinder.AtributoN
FROM PSD_Zylinder

Com a informação que o usuário escolheu um ponto de entrada da entidade para efetuar a navegação passadas ao monitor de consistência pela GUI, este monta a seguinte consulta de navegação:

Cláusula de condição:

PSD_Zylinder.Durchmesser = LKD_Motor.Bohrung

Tabelas:

PSD_Zylinder
LKD_Motor

Esta parte da consulta é retornada à GUI que adiciona as informações à consulta corrente do usuário, transformando-a na seguinte consulta (as modificações estão em itálico):

SELECT PSD_Zylinder.Atributo1, PSD_Zylinder. Atributo2, ..., PSD_Zylinder.AtributoN
FROM PSD_Zylinder, *LKD_Motor*
WHERE PSD_Zylinder.Durchmesser = LKD_Motor.Bohrung

No caso do identificador do motor estar envolvido na navegação, ainda é acrescido à consulta apresentada anteriormente o apontador para as fontes de dados originais. O atributo que serve de apontador entre as tabelas de mapeamento e as fontes de dados originais deve fazer parte da consulta final após a navegação para que possam ser recuperadas outras informações da tupla do motor especificado pelo usuário. Ainda, devido os vários tipos de representações dos bancos de dados para o identificador do motor, o tratamento dado à construção das consultas de navegação pelo monitor de consistência é diferenciado, como vemos a seguir:

Consulta de Navegacao 4:

PSD ou LKD => Mentas / Mentas => PSD ou LKD

Cláusula de condição:

TabelaMapeamentoCorrente.AtributoCorrente = EntidadeCorrespondente.AtributoCorrespondente
AND
TabelaMapeamanetoCorrente.Ponteiro = EntidadeFonte.AtributoFonteCorrespondente

Tabelas:

TabelaMapeamentoCorrente
Entidade Fonte
EntidadeCorrespondente

Consulta de Navegacao 5:

LKD => PSD ou PSD => LKD

Cláusula de condição:

TabelaMapeamentoCorrente.AtributoCorrente = TabelaMapeamentoCorrenteCorresp.AtributoCorrespondente
AND
TabelaMapeamentoCorrente.Ponteiro = EntidadeFonte.AtributoFonteCorrespondente
AND
TabelaMapeamentoCorrenteCorresp.Ponteiro = EntidadeFonteCorresp.AtributoFonteCorrespondente

Tabelas:

TabelaMapeamentoCorrente
TabelaMapeamentoCorrenteCorresp
EntidadeFonte

EntidadeFonteCorresp

Para deixar mais claro, daremos um exemplo de uma navegação entre o LKD e PSD, onde a entidade escolhida para efetuar a navegação contém o identificador do motor. Suponha ainda que o usuário no momento da navegação marcou os três pontos de entrada do identificador do motor para realizar a navegação (tipo, especificação e *baumuster*). Então, partindo da consulta abaixo:

SELECT PSD_Motor.Atributo1, PSD_Motor. Atributo2, ..., PSD_Motor.AtributoN
FROM PSD_Motor

o monitor de consistência cria a seguinte consulta de navegação ao receber a informação da GUI que o usuário marcou os atributo *Typ*, *Spezifikation* e *Baumuster* para efetuar a navegação:

Cláusula de condição:

PSDMapping.Typ = LKDMapping.Typ AND
PSDMapping.Spezifikation = LKDMapping.Spezifikation AND
PSDMapping.Baumuster =LKDMapping.Baumuster AND
PSDMapping.Ursprung = PSD_Motor.Typ AND
LKDMapping.Ursprung = LKD_Motor.Typ

Tabelas:

PSDMapping
LKDMapping
PSD_Motor
LKD_Motor

Ao retornar a cláusula de condição e as tabelas envolvidas para a GUI, esta as adiciona à consulta inicial, produzindo a seguinte consulta após a navegação:

SELECT PSD_Motor.Atributo1, PSD_Motor. Atributo2, ..., PSD_Motor.AtributoN
FROM PSD_Motor, *LKD_Motor, PSDMapping, LKDMapping*
WHERE PSDMapping.typ = LKDMapping.Typ AND
PSDMapping.Spezifikation = LKDMapping.Spezifikation **AND**
PSDMapping.Baumuster =LKDMapping.Baumuster **AND**
PSDMapping.Ursprung = PSD_Motor.Typ **AND**
LKDMapping.Ursprung = LKD_Motor.Typ[1]

1. Note que quando o PSD e o LKD estão envolvidos na navegação é necessário que os ponteiros das tabelas de mapeamento sejam colocados para ambos, como mostrado pela última consulta.

6.8 Um exemplo de navegação no MENTAS

Todos os esforços do Monitor de Consistência na fase de integração dos bancos de dados do projeto MENTAS são no sentido de possibilitar que o usuário amplie o universo da sua consulta SQL de modo que esta atinja os outros bancos de dados. Dessa forma, apresentaremos aqui um exemplo do comportamento do monitor de consistência numa suposta navegação entre os três bancos de dados integrados.

De acordo com a Figura 6.9, podemos ver que existem duas formas básicas para realizar uma navegação no MENTAS. Com a presença de uma cláusula de condição, ou sem a presença desta. Ainda, na primeira alternativa, podemos subdividir de acordo com a presença de pontos de entrada ou não.

Nesta seção, faremos a simulação de uma navegação para os dois casos apresentados: Uma navegação sem cláusula de condição e uma navegação com a presença de pontos de entrada na condição da consulta. Não apresentaremos aqui, o comportamento da interface, mas sim, a interação entre os módulos que compõem o Monitor de Consistência.

Lembramos que é possível definir uma consulta com a presença de vários valores para o mesmo ponto de entrada. Aqui, no entanto, será apresentado apenas com um valor para o ponto de entrada. Ainda, optamos por um exemplo onde a condição é composta por dois pontos de entrada: o identificador do motor e outro ponto de entrada que é tratado pelo gerente de pontos de entrada. Dessa forma, apresentaremos a funcionalidade de todos os componentes do monitor de consistência.

O tratamento também é diferenciado, de acordo com os pares de bancos de dados que são escolhidos pelo usuário para realizar a navegação. Considere então, que a navegação no nosso exemplo começa no banco de dados **PSD**, seguindo para o **Mentas** e por último o **LKD**. Considere que a consulta inicial a seguir seja executada com sucesso pelo banco de dados e que retorne alguns resultados, possibilitando ao usuário seguir adiante na navegação. As transformações dos pontos de entrada são feitas na maioria das vezes de forma transparente para o usuário. A única exceção é o identificador de motor. Neste caso, parte do mapeamento é feito pelo usuário. Entretanto, para o identificador do motor é dada a opção de fazer a checagem da consistência entre os demais pontos de entrada para um determinado motor (seção 6.8.2).

Ao contrário disto, tentamos restringir ao máximo o conjunto de possíveis resultados, de forma que o usuário consiga identificar o motor correspondente de forma rápida e segura. Quando não conseguimos encontrar motores que preenchem as características do que foi informado pelo usuário, é proposto a este analisar os motores que preenchem os requisitos dos componentes do identificador do motor separadamente (operação de *other matches*). Veremos como cada um desses aspectos é tratado pelo Monitor de Consistência.

No nosso exemplo, nenhum atributo é incluído na projeção da consulta nos bancos de dados que seguem o inicial. A razão para isto é apenas simplificar a consulta, já que este tipo de inserção não traz mudanças para a atuação do monitor de consistência, não sendo portanto, útil no nosso exemplo, o qual objetiva apenas mostrar a atuação deste durante uma navegação. Com a intenção de simplificar também, supomos que o usuário não adiciona nenhum atributo na cláusula de condição antes de iniciar a segunda navegação (entre o Mentas e o LKD). Como a projeção, este tipo de inserção não traz mudanças ao monitor de consistência, já que são completamente ignorados pelo mesmo.

Suponha que o usuário começa a navegação partindo apenas de um ponto de entrada, *Typ*, no PSD, sendo que ao chegar no Mentas, um novo ponto de entrada é incluído na entidade Motor (*Bohrung)*, seguindo a navegação por dois pontos de entradas tratados por módulos distintos do Monitor de Consistência.

Utilizamos a consulta a seguir como ponto de partida para o nosso exemplo: Palavras em negrito simbolizam os comandos SQL. Os atributos são escritos seguindo a notação ″entidade.atributo″.

SELECT PSD_Motor.Typ
FROM PSD_Motor
WHERE PSD_Motor.Typ = 'M111E18MMX'

6.8.1 Sem Checagem de Consistência

Partimos do princípio que a consulta acima já foi formulada, e que o usuário escolheu o banco de dados Mentas como o banco de dados posterior a ser visitado pela navegação.

PSD - Mentas

Como existe um valor para o identificador do motor na consulta, é necessário a intervenção do monitor de consistência localizado no cliente. Dessa forma, a GUI faz uma chamada ao monitor de consistência do cliente, passando como informação o valor fornecido ao ponto de entrada na consulta acima, e este retorna o valor decomposto no formato adequado: tipo, especificação e modelo de construção. É montado pelo navegador uma consulta intermediária que procura no banco de dados Mentas o motor correspondente ao fornecido pelo usuário no PSD, utilizando o resultado do parser, que mostramos a seguir:

Tabela 6.11 Atuação do Parser.

Parser				
Motor	*Tipo*	*Especificação*	*Arte de Construção*	*Desconhecido*
M111E18MMX	M111	E18		MMX

Utilizando a definição da Consulta Intermediária 1, obtemos a seguinte consulta intermediária:

SELECT Motor.Typ, Motor.Spezifikation, Motor. Baumuster
FROM Motor
WHERE Motor.Typ = 'M111' **AND** Motor.Spezifikation = 'E18'

Como podemos notar, a seqüência final do motor especificado foi totalmente ignorada, já que foi retornado no campo "desconhecido" pelo parser. O campo desconhecido só é utilizado pelo monitor de consistência para montar as consultas intermediárias, no caso de não ser retornado nada em nenhum dos outros campos. Por exemplo, se o usuário informasse o motor "M11", o parser do cliente retornaria que essa seqüência só encaixa-se no campo desconhecido.

A seguir é feita uma chamada ao Monitor de Consistência - servidor, mais precisamente ao gerente de acesso ao banco de dados - passando então a consulta intermediária, a qual seleciona no banco de dados Mentas o motor com o tipo igual a M111, a especificação igual a E18 e o baumuster igual a 921. Este é o único motor no banco de dados Mentas que satisfaz à cláusula de condição da consulta intermediária. Note porém que o motor possui a seqüência 921 que corresponde a modelo de construção, apesar do motor especificado ao banco de dados PSD não ter essa seqüência. Retornamos então o motor encontrado ao cliente e a GUI o apresenta ao usuário, através de uma janela de escolha. Neste exemplo, apenas um motor foi selecionado, mas em muitos casos, a lista de candidatos chega a ter dezenas de motores diferentes. Como é sabido, os bancos de dados possuem representações diferentes, e o motor M111E18921 é o que mais assemelha-se ao motor procurado pelo usuário, M111E18MMX. Caso concorde que os motores são realmente idênticos, selecionando o motor na janela de escolha na qual é apresentado, a navegação para o Mentas é terminada com a alteração da consulta SQL inicial efetuada pela GUI, que agora apresenta-se da seguinte forma:

SELECT PSD_Motor.Typ
FROM PSD_Motor, Motor
WHERE PSD_Motor.Typ = 'M111E18MMX' **AND** Motor.Typ = 'M111' **AND** Motor.Spezifikation = 'E18'
AND Motor.Baumuster = '921'

O usuário portanto já se encontra no banco de dados MENTAS. Considere que um novo ponto de entrada é adicionado - através da GUI - a entidade Motor, sobre a qual irá ocorrer a navegação[1]. O resultado pode ser visto na consulta abaixo:

```
SELECT PSD_Motor.Typ
FROM PSD_Motor, Motor
WHERE (PSD_Motor.Typ = 'M111E18MMX' AND Motor.Typ = 'M111' AND Motor.Spezifikation = 'E18'
AND Motor.Baumuster = '921') AND (Motor.Bohrung > 11.5 OR Motor.Bohrung < 110.0)
```

Bem, como o Mentas já possui os dados no formato adequado, não é necessário que a GUI chame o Monitor de Consistência do Cliente. Como podemos notar, temos dois pontos de entrada tratados por dois módulos diferentes no Monitor de Consistência. A GUI separa os pontos de entrada e efetua uma chamada a cada um dos módulos passando as informações necessárias.

Mentas - LKD

Como já foram visitados o PSD e o Mentas, o único banco de dados possível para expandir os limites da consulta é o LKD. No caso do identificador do motor, será montada uma nova consulta intermediária que acessará a tabela de mapeamento do LKD para procurar os motores que preenchem os requisitos do motor especificado no Mentas, ou seja, os motores que possuam o tipo igual a M111, a especificação igual a E18 e o baumuster igual a 921. De acordo com a Consulta Intermediária 1[2], temos:

```
SELECT TabMapCorrenteLKD.Ursprung
FROM TabMapCorrenteLKD
WHERE TabMapCorrenteLKD.Typ = 'M111' AND TabMapCorrenteLKD.Spezifikation = 'E18' AND Tab-
MapCorrenteLKD.Baumuster = '921'
```

Como podemos notar, a consulta acessa as tabelas de mapeamento, e como tal, antes de executá-la no servidor de banco de dados, esta é passada pelo atualizador de consultas para que o nome da tabela de mapeamento corrente seja atualizada. Assim, antes de executar, todas as ocorrências são transformadas para a tabela de mapeamento corrente do LKD. Considere que o nome da tabela corrente para o LKD seja Mapping_Motor_LKD1, sendo a consulta alterada para:

```
SELECT Mapping_Motor_LKD1.Ursprung
FROM Mapping_Motor_LKD1
WHERE Mapping_Motor_LKD1.Typ = 'M111' AND Mapping_Motor_LKD1.Spezifikation = 'E18' AND
Mapping_Motor_LKD1.Baumuster = '921'
```

1. Note que se o ponto de entrada adicionado, pertencesse a outra entidade que não fosse Motor (a entidade que irá efetuar a navegação) não era preciso considerá-lo.
2. Note que no PSD e LKD o Typ, Spezifikation e Baumuster do Mentas correspondem à Ursprung.

Mas, não existe nenhum motor no LKD que satisfaça à consulta acima. O gerente de acesso retorna então essa informação à GUI que a reporta ao usuário. Entretanto, é dada ao usuário a possibilidade de verificar outros motores que poderão vir a ser o especificado, já que cada banco de dados possui uma representação própria. Caso o usuário deseje verificar os outros motores (*other matches*), três novas consultas intermediárias são criadas, onde iremos procurar por cada atributo que compõe o motor separadamente. De acordo com Consulta Intermediária 5, Consulta Intermediária 6 e Consulta Intermediária 7, temos as seguintes consultas respectivamente:

```
SELECT Mapping_Motor_LKD1.Ursprung
FROM Mapping_Motor_LKD1
WHERE Mapping_Motor_LKD1.Typ = 'M111'

SELECT Mapping_Motor_LKD1.Ursprung
FROM Mapping_Motor_LKD1
WHERE Mapping_Motor_LKD1.Spezifikation = 'E18'

SELECT Mapping_Motor_LKD1.Ursprung
FROM Mapping_Motor_LKD1
WHERE Mapping_Motor_LKD1.Baumuster = '921'
```

O gerente de Acesso ao Banco de dados retorna para a GUI um vetor com os resultados de cada consulta acima e esta reporta ao usuário. A GUI dá ao usuário a possibilidade de visualizar os resultados -motores- de cada uma das consultas, de modo que este escolha o motor correspondente ao especificado. Para as consultas específicas apresentadas acima, são retornados ao usuário 26 motores que possuem o tipo igual a M111, nenhum motor para a especificação igual a E18 e 4 motores que possuem a modelo de construção igual a 921. Dentro dessa lista de possibilidades, o usuário pode escolher o motor (ou os motores) correspondentes.

Os motores escolhidos são adicionados pela GUI à consulta original, e damos início à segunda etapa da navegação através do segundo ponto de entrada especificado. Outra chamada ao Monitor de Consistência é feita, dessa vez para tratar o atributo *Bohrung*. O que deve ser feito agora é o mapeamento do nome do atributo e valor especificado pelo usuário aos correspondentes para o banco de dados posterior. Assim, transformamos o valor 11.5 para '11,5' já que no LKD o atributo correspondente é um CHAR e que o separador de casas decimais é uma vírgula. Da mesma forma, transformamos o valor 110.0 para '110,0'. A seguir, mapeamos a entidade e o atributo *Motor* e *Durchmesser,* para os correspondentes no LKD, ou seja, *LKD_Motor* e *Bohrung* respectivamente.

Terminado o trabalho, o gerenciador de pontos de entrada retorna para a GUI a entidade, o atributo, e os valores já mapeados, os quais são utilizados para completar a consulta do usuário. O LKD passa a ser então o novo banco de dados corrente e acaba a navegação. Para a consulta abaixo, supomos que o usuário escolheu ape-

nas dois motores (M111.921 e M111.920) dentre os apresentados. Dessa forma, o usuário recebe a seguinte consulta:

```
SELECT PSD_Motor.Typ
FROM PSD_Motor, Motor, LKD_Motor
WHERE (PSD_Motor.Typ = 'M111E18MMX' AND Motor.Typ = 'M111' AND
Motor.Spezifikation = 'E18' AND Motor.Baumuster = '921') AND
(Motor.Bohrung > 11.5 OR Motor.Bohrung < 110.0) AND
(LKD_Motor.Typ = 'M111.921' OR LKD_Motor.Typ = 'M111.921') AND
(LKD_Motor. Bohrung > '11,5' OR LKD_Motor.Bohrung < '110,0')
```

6.8.2 Com Checagem de Consistência

Aqui apresentaremos o mesmo exemplo anterior, só que levando em consideração que o usuário deseja que ocorra a navegação dentro dos critérios definidos para a manutenção da consistência entre os valores do identificador do motor nos bancos de dados. Para não tornar as explicações repetitivas, apenas as diferenças serão levadas em consideração nesta seção.

No quadro abaixo, podemos ver como será feita a comparação entre os bancos de dados no momento da navegação pelo identificador do motor. Estes pontos de entrada são definidos pelo usuário através da janela apresentada pela Figura 5.16:

Tabela 6.12 Pontos de Entrada para Checagem de Consistência.

	Mentas ⇔ PSD	LKD ⇔ Mentas	PSD ⇔ LKD
Durchmesser	Sim	Não	Sim
Hubvolumen	Não	Sim	Sim
Ventil_DM_Aussen_1 Ein_Auslass_Kanal	Não	Não	Não
Pleuellaenge	Sim	Sim	Sim
Hub	Não	Não	Não
Max_Ventil_Hub Ein_Auslass_Kanal	Não	Não	Não

Assim, para o par Mentas - PSD, devem ser levados em consideração o *Durchmesser* e o *PleuelLaenge*. Para o par LKD - Mentas, apenas os atributos *HubVolumen* e *PleuelLaenge* são considerados. Com relação ao PSD e LKD são analisados o *Durchmesser*, o *HubVolumen* e *PleuelLaenge*.

PSD - Mentas

Do mesmo modo que a consulta sem verificação de consistência, também faz-se necessário a chamada ao parser do cliente, de forma que o valor é decomposto em tipo, especificação e modelo de construção. É montada pelo navegador uma con-

sulta intermediária que procura no banco de dados Mentas o motor correspondente ao fornecido pelo usuário no PSD, sendo que desta vez, é necessário acrescentar a consulta os pontos de entrada que são comparados entre os bancos de dados para garantia da consistência entre os motores dos dois bancos de dados envolvidos.

Utilizando a definição da Consulta Intermediária 11, obtemos a seguinte consulta intermediária:

```
SELECT Motor.Typ, Motor.Spezifikation, Motor. Baumuster
FROM Motor, psdMapping, Pleuel
WHERE Motor.Typ = 'M111' AND psdMapping.Typ = 'M111'
AND Motor.Spezifikation = 'E18' AND psdMapping.Spezifikation = 'E18'
AND Motor.Bohrung = psdMapping.Bohrung
AND Pleuel.Laenge = psdMapping.PleuelLaenge
```

Através desta consulta, são recuperados os motores que além de possuírem a mesma identificação (tipo e especificação), ainda possuam o *Bohrung* e o *PleuelLaenge* iguais. Quanto maior for o número de pontos de entrada escolhidos pelo usuário para a comparação entre os bancos de dados, maior será a segurança de que o motor apresentado corresponde realmente ao motor informado no banco de dados corrente.

Como no exemplo sem verificação de consistência, apenas o motor com tipo igual a M111, especificação igual a E18 e modelo de construção igual a 921 é recuperado do banco de dados Mentas. Isso significa que este motor possui o *Bohrung* e o *PleuelLaenge* também correspondentes. Considerando que este motor é escolhido pelo usuário, é originada a consulta no final desta navegação:

```
SELECT PSD_Motor.Typ
FROM PSD_Motor, Motor
WHERE PSD_Motor.Typ = 'M111E18MMX' AND Motor.Typ = 'M111' AND Motor.Spezifikation = 'E18'
AND Motor.Baumuster = '921'
```

Como podemos notar, nenhuma modificação na consulta apresentada ao usuário é feita. As mudanças realizadas são apenas nas consultas intermediárias.

Mentas - LKD

Agora, já é possível navegar para o banco de dados LKD. Como no exemplo anterior, adicionamos à consulta um novo ponto de entrada à entidade Motor, sobre a qual irá ocorrer a navegação. O resultado pode ser visto na consulta abaixo:

```
SELECT PSD_Motor.Typ
FROM PSD_Motor, Motor
WHERE PSD_Motor.Typ = 'M111E18MMX' AND (Motor.Typ = 'M111' AND
Motor.Spezifikation = 'E18' AND Motor.Baumuster = '921') AND
(Motor.Bohrung > 11.5 OR Motor.Bohrung < 110.0)
```

No caso do identificador do motor, é montada uma nova consulta intermediária que acessa a tabela de mapeamento do LKD para procurar os motores que podem preencher os requisitos do motor especificado no Mentas, ou seja, os motores que possuam o tipo igual a M111, a especificação igual a E18 e o baumuster igual a 921. Além disso, é preciso que possuam o *HubVolumen* e o *PleuelLaenge* iguais. De acordo com a Consulta Intermediária 11[1], temos:

```
SELECT TabMapCorrenteLKD.Ursprung
FROM TabMapCorrenteLKD,Motor, Pleuel
WHERE TabMapCorrenteLKD.Typ = 'M111' AND Motor.Typ = ’M111’ AND
TabMapCorrenteLKD.Spezifikation = 'E18' AND Motor.Spezifikation = ’E18’ AND
TabMapCorrenteLKD.Baumuster = '921' AND Motor.Baumuster = ’921’ AND
Motor.ZylinderHubvolumen =TabMapCorrenteLKD.Hubvolumen AND
Pleuel.Laenge =TabMapCorrenteLKD.PleuelLaenge
```

Antes de enviar a consulta acima para execução no servidor de banco de dados, esta é passada pelo atualizador de consultas para que o nome da tabela de mapeamento corrente seja atualizado. Assim, antes de executar, todas as ocorrências são transformadas para a tabela de mapeamento corrente do LKD. Supondo que a tabela de mapeamento ativa para o banco de dados LKD é Mapping_Motor_LKD1, temos:

```
SELECT Mapping_Motor_LKD1.Ursprung
FROM Mapping_Motor_LKD1,Motor, Pleuel
WHERE Mapping_Motor_LKD1.Typ = 'M111' AND Motor.Typ = ’M111’ AND
Mapping_Motor_LKD1.Spezifikation = 'E18' AND Motor.Spezifikation = ’E18’ AND
Mapping_Motor_LKD1.Baumuster = '921' AND Motor.Baumuster = ’921’ AND
Motor.ZylinderHubvolumen =Mapping_Motor_LKD1.Hubvolumen AND
Pleuel.Laenge =Mapping_Motor_LKD1.PleuelLaenge
```

Como no exemplo anterior, não existe nenhum motor no LKD que satisfaça à consulta acima. O gerente de acesso retorna então essa informação à GUI que a reporta ao usuário. Se o usuário optar por realizar a navegação através da opção de *other matches*, três novas consultas intermediárias são criadas, onde iremos procurar por cada atributo que compõe o motor separadamente. Mais uma vez de acordo com Consulta Intermediária 13, Consulta Intermediária 14 e Consulta Intermediária 15, temos as seguintes consultas:

```
SELECT TabMapCorrenteLKD.Ursprung
FROM TabMapCorrenteLKD, Motor, Pleuel
WHERE TabMapCorrenteLKD.Typ = 'M111' AND Motor.Typ = ’M111’AND
Motor.ZylinderHubvolumen =TabMapCorrenteLKD.Hubvolumen AND
Pleuel.Laenge =TabMapCorrenteLKD.PleuelLaenge
```

```
SELECT TabMapCorrenteLKD.Ursprung
FROM TabMapCorrenteLKD, Motor, Pleuel
WHERE TabMapCorrenteLKD.Spezifikation = 'E18' AND Motor.Spezifikation = ’E18’ AND
Motor.ZylinderHubvolumen =TabMapCorrenteLKD.Hubvolumen AND
Pleuel.Laenge =TabMapCorrenteLKD.PleuelLaenge
```

1. Mais uma vez note que o typ, spezifikation e baumuster da projeção foi trocado pela ursprung.

```
SELECT TabMapCorrenteLKD.Ursprung
FROM TabMapCorrenteLKD
WHERE TabMapCorrenteLKD.Baumuster = '921' AND Motor.Baumuster = ’921’ AND
Motor.ZylinderHubvolumen =TabMapCorrenteLKD.Hubvolumen AND
Pleuel.Laenge =TabMapCorrenteLKD.PleuelLaenge
```

Bem diferente do resultado retornando anteriormente, quando estas mesmas consultas foram montadas sem a comparação dos pontos de entrada, agora temos um conjunto reduzido de motores que são apresentados ao usuário. Desta vez, foram descartados os motores que apesar de ter o mesmo tipo (ou especificação, ou mesmo modelo de construção), não possuem os pontos de entrada com valores iguais. Dessa forma, é retirada do usuário a responsabilidade de conhecer a semântica do identificador do motor nos bancos de dados. Desta vez, apenas dois (2) motores são retornados para a consulta através do tipo, e um (1) motor é retornado para a consulta através da modelo de construção. Dentro dessa lista de possibilidades, o usuário pode escolher o motor (ou os motores) correspondente (s), com uma possibilidade muito maior de acerto.

As diferenças entre a navegação, com e sem checagem de consistência, terminam aqui. O próximo ponto de entrada é mapeado da mesma maneira que no exemplo anterior. Assim, o LKD passa a ser então o novo banco de dados corrente e acaba a navegação. Para a consulta abaixo, supomos que o usuário escolheu apenas um motor (M111.921) dentre os apresentados. A consulta final é apresentada ao usuário da seguinte forma:

```
SELECT PSD_Motor.Typ
FROM PSD_Motor, Motor, LKD_Motor
WHERE PSD_Motor.Typ = 'M111E18MMX' AND
Motor.Typ = 'M111' AND Motor.Spezifikation = 'E18' AND
Motor.Baumuster = '921' AND (Motor.Bohrung > 11.5 OR
Motor.Bohrung < 110.0) AND (LKD_Motor.Typ = 'M111.921') AND
(LKD_Motor. Bohrung > '11,5' OR LKD_Motor.Bohrung < '110,0')
```

6.9 Consistência X Performance

A propriedade de consistência é uma das mais desejadas em um sistema. Mas também é uma das mais difíceis de ser garantida. Mas, o que vem a ser consistência? O termo é tão genérico que existem várias definições de consistência de acordo com a comunidade científica interessada [Cho96]. Apesar da consistência ser umas das propriedades mais importantes em um sistema de banco de dados, ela compromete outra propriedade muito importante no sistema, a performance. Consistência e performance são inversamente proporcionais. É preciso definir até onde cada uma dessas propriedades é importante para o sistema.

No Mentas, não é diferente. A performance do sistema cai quando o usuário especifica que a navegação deve ser feita dentro das especificações de consistência. O usuário recebe informações mais seguras mas com tempo de resposta mais lento.

Mas o MENTAS busca além de garantir a consistência das informações acessadas em banco de dados heterogêneos também garantir a performance do sistema. Portanto, no momento de definir as técnicas de consistência, procuramos sempre a melhor resolução também com relação à performance. Como resultado, MENTAS é capaz de acessar os bancos de dados heterogêneos com uma performance exemplar.

Existem algumas técnicas do monitor de consistência que são a chave para que a navegação entre os bancos de dados ocorra com uma excelente performance. A primeira delas são as tabelas de mapeamento. Com as tabelas de mapeamento, resolvemos vários problemas referentes à navegação, dentre os quais podemos citar:

- a possibilidade da comparação dos atributos que identificam o motor entre os banco de dados integrados.
- aumento na performance da navegação, quando o usuário requisita que esta seja feita dentro da checagem de consistência (seção 6.9).

Além disso, o caso de uma navegação onde a verificação de consistência é requisitada mostrou-se muito ineficiente, caso todos os pontos de entrada fossem buscados para comparação nas fontes de dados originais. O tempo de espera por parte do usuário seria enorme, ultrapassando 15 minutos para a realização de uma navegação simples. Armazenando esses atributos nas tabelas de mapeamento, conseguimos diminuir o tempo consideravelmente, não chegando a mais de um minuto (no pior dos casos) para que a navegação seja concluída.

Outra vantagem das tabelas de mapeamento é que, como foi dito, elas são armazenadas no banco de dados do sistema *middleware*. Com isso não existe quase o acesso ao banco de dados remoto no momento da navegação (apenas para executar a consulta corrente do usuário e com isso, habilitar ou não a navegação), já que todas as operações podem ser feitas a nível local.

Mas, em algum momento deve ser feito o acesso aos bancos de dados remotos para obter as informações dos pontos de entrada. Isso é feito no momento em que é requisitado a geração de uma nova tabela de mapeamento. O caso mais crítico é verificado no banco de dados PSD, que possui atributos em várias entidades distintas. Devido a natureza da distribuição dos pontos de entrada neste banco de dados, a consulta utilizada para recuperar esses valores no momento da criação da tabela de mapeamento, ultrapassava facilmente o tempo de 15 minutos para realizar a execução.

Para resolver este problema, quebramos a consulta original em subconsultas que são enviadas ao servidor de banco de dados e os resultados são armazenamos temporariamente em tabelas no banco de dados do sistema *middleware*. Por último, fazemos um *join* dessas tabelas, para finalmente armazenar os valores nas tabelas de mapeamento. Com esse procedimento, conseguimos que o tempo de geração

das tabelas de mapeamento não passe dos 40 segundos. Apesar de ainda ser um tempo de espera longo para o usuário, devemos levar em consideração que esta não é uma atividade freqüente realizada pelo usuário, ao passo que a navegação é a base da integração dos bancos de dados do MENTAS, e por isso, é uma atividade freqüente no sistema.

Um segundo tópico de grande importância para a garantia da performance é a nossa técnica de verificação de consistência quando trata-se do ponto de entrada que identifica o motor. Poderíamos apenas ter definido que para um identificador de motor ser considerado semelhante a outro, todos os pontos de entrada deveriam ser iguais. Mas, mais uma vez, pensando na performance decidimos deixar a critério do usuário definir quais são os pontos de entrada importantes para serem comparados no momento da navegação. Isso ocasionou uma grande dificuldade a nível de implementação, mas o resultado é surpreendente. Parte desses resultados foram conseguidos devido à adição de todos os pontos de entrada nas tabelas de mapeamento. Mesmo que o usuário escolha que a verificação de consistência deve ocorrer sobre todos os pontos de entrada, como todos estão presentes nas tabelas de mapeamento e estas por sua vez no banco de dados do sistema *middleware*, localmente, não existe uma grande queda na performance.

6.10 Considerações

Aqui foi apresentada uma proposta para integração de esquemas de bancos de dados heterogêneos integrados através da tecnologia *middleware* de banco de dados. Como o *middleware* não trata as diferenças de esquemas e representações dos dados no momento da integração das fontes, foi necessário tratá-los separadamente. Isto é feito pelo Monitor de Consistência no projeto MENTAS. Este módulo é responsável por tratar as diferenças entre os pontos de entrada dos bancos de dados no momento da integração e cuidar de todas as operações que envolvem estes pontos como: as tabelas de mapeamento, montagem das consultas que acessam as tabelas de mapeamento, transformação da entrada dos dados referentes aos pontos de entrada, montagem das consultas ao final de uma navegação. De uma forma geral este módulo trata de todas questões referentes à navegação de um banco de dados para outro dentro do MENTAS, sempre levando em consideração a performance final do sistema. Dessa forma o usuário consegue acessar informações de vários bancos de dados, relacionando-as ou comparando-as sem a necessidade de ter o conhecimento de como os dados estão representados nos bancos de dados e o melhor, conseguindo resultados de forma rápida e segura.

7 Conclusão

7.1 Considerações Finais

Existem muitas propostas para a integração de bancos de dados heterogêneos na literatura. Mas, devido às características apresentadas pelos bancos de dados integrados pelo MENTAS, não foi possível empregar nenhuma das técnicas. As três propostas mais aceitas, esquema global, sistema de banco de dados federados e os sistemas de linguagens de *multidatabases* não são adequadas para o MENTAS. A princípio seria possível definir um esquema global para integrar as fontes de dados utilizando a tecnologia *middleware*. Mas, devido às diferenças das representações de atributos isto não foi possível. Já as linguagens de consultas de *multidatabases* não são adequadas para o MENTAS devido ao fato de colocar a responsabilidade de grande parte da integração dos bancos de dados sobre o usuário e isso fere o principal propósito do MENTAS no qual o usuário deve ser o maior beneficiado.

Encontramos na tecnologia *middleware* o suporte necessário para integrar as fontes de dados do MENTAS, garantido a autonomia das fontes locais após a integração dos bancos de dados e ainda a total transparência da localização das fontes de dados (garantida pelo DataJoiner através da definição de *nicknames*). Mas, apesar do DataJoiner providenciar a transparência de localização, transparência de linguagens de acesso aos bancos de dados e transparência da heterogeneidade dos sistemas envolvidos (*hardware* e sistemas operacionais e protocolos de comunicação) não existe nenhuma homogeneização estrutural e de representação das diferenças entre os esquemas dos múltiplos bancos de dados envolvidos. Sabendo que este é o principal gargalo da fase de integração de banco de dados heterogêneos, propusemos uma solução para integração dessas fontes de dados. A metodologia empregada possui como ponto marcante o conceito de navegação.

O projeto de integração de esquemas no MENTAS dividiu-se em quatro fases distintas:

- Descobrir os objetos que estão relacionados nos diversos bancos de dados, os quais são denominados de pontos de entrada na nossa metodologia.
- Descobrir as diferenças entre esses objetos.
- Definir regras para efetuar o tratamento correto desses pontos de entrada no momento da navegação.
- Resolução das diferenças entre os bancos de dados.

7.2 Contribuições

A principal contribuição do nosso trabalho encontra-se exatamente na última fase. Apresentamos nesta dissertação técnicas para garantir a consistência no momento da integração das várias fontes de dados e ao mesmo tempo conservando a performance do sistema. Encontramos no uso de tabelas intermediárias armazenadas no banco de dados do sistema *middleware* a solução para muitos dos nossos problemas durante a integração destas fontes de dados.

Outra contribuição relevante da nossa dissertação é a apresentação de busca por proximidades (*other matches*) quando o valor informado pelo usuário a um banco de dados não é encontrado no banco de dados para onde está ocorrendo a navegação. Nos bancos de dados do MENTAS esta foi a solução encontrada o problema na existência de várias formas que um mesmo dado é armazenado nos diversos bancos de dados heterogêneos.

Assim como a busca por proximidades, possibilitamos ainda no caso de pontos de entrada cujas representações variam muito entre as fontes integradas a possibilidade da navegação com a checagem de consistência, onde além do ponto de entrada definido para efetuar a navegação são comparados entre os bancos de dados todos os demais pontos de entrada escolhidos pelo usuário para um dado par de banco de dados. Com esta técnica, garantimos uma maior corretude nos dados apresentados ao usuário.

Outra contribuição foi a realização de um estudo detalhado sobre os sistemas de banco de dados heterogêneos. Foi exposto um panorama geral da pesquisa nesta área. Uma série de referências bibliográficas abordando temas referentes a integração de esquemas estão disponíveis, possibilitando a sua utilização em trabalhos futuros.

7.3 Trabalhos Futuros

Embora o monitor de consistência tenha atingindo o objetivo inicial que era providenciar a manutenção de consistência no momento de integração de bancos de dados heterogêneos, foram identificadas algumas sugestões para o desenvolvimento de futuras extensões. A primeira delas seria melhorar nossa busca por proximidades. É possível inferir valores de um certo atributo considerando outros atributos dos bancos de dados. Será necessário uma interação com os engenheiros mecânicos para capturar o algoritmo utilizado por eles atualmente e traduzí-lo para o MENTAS. A descoberta de outros pontos de entrada nos bancos de dados seria bastante útil uma vez que disponibilizaria aos usuários maiores opções no momento da navegação. Poderia também ser adicionado ao monitor de consistência um método automático para gerar as tabelas de mapeamento no momento que ocorrer uma atualização nos bancos de dados originais. Ainda, existem algumas

regras utilizadas pelos engenheiros para validar os dados recebidos para um determinado motor. Essas regras poderiam ser capturadas por um tipo de sistema especialista visando oferecer informações mais seguras para os usuários do MENTAS.

Outra proposta seria integrar o trabalho do monitor de consistência com o gerente de *workflow* do MENTAS. Como existe uma série de ferramentas a serem integradas será necessário o apoio do monitor de consistência para gerenciar o fluxo de informações entre as ferramentas, bem como entre as fontes de dados que cada ferramenta mantém.

Referências

[ASDK+91] Ahmed, R., Smedt, P. D., Du, W., Kent, W., Ketabchi, M. A., Litwin, W. A., Rafii, A. and Shan, M. *The Pegasus Heterogeneous Multidatabase System*. IEEE Computer, 1991.

[Bar97] Barroso, M. C. *PROJETOO: Uma Metodologia para Projeto de Banco de Dados Distribuídos Orientados a Objetos*. Dissertação de Mestrado. Universidade Federal de Pernambuco, Recife, Dezembro de 1997.

[BBE98] Bouguettaya, A., Benatallah, B., Elmagarmid, A. *Interconnecting Heterogeneous Information Systems*. Kluwer Academic Publishers, 1998.

[Bel88] Belcastro, V. *An Overview of the Distributed Query System DQS*. Proc. Int'l Conf. Extending Database Technology, Springer-Verlag, 1988, pp.170-189.

[BGL87] Bell, D. A., Grimson, J.B. and Ling, D. H. O. *EDDS - A System to Harmonize Access to Heterogeneous Databases on Distributed Micros and Mainframes*. Information and Software Technology, Vol 29, No. 7, 1987, pp. 362-370.

[BGS92] Breitbart, Y., Garcia-Molina, H., Silberschatz, A. *Overview of Multidatabase Transaction Management*, VLDB Journal,1992.

[BHP92] Bright, M. W., Hurson, A. R., Pakzad, S. H. *A Taxonomy and Current Issues in Multidatabase Systems*. In IEEE Computer, Vol. 25, N. 3, Março de 1992, pp.50-60.

[BLN86] Batini, C., Lenzerini, M. and Navathe, S. *A Comparative Analyses of Methodologies of Database Schema Integration.* ACM Computing Surveys, 18(4):323-364, 1986.

[BNPS88] Bertino E., Negri M., Pelaggati G. and Sbatella L. *The Commands Integrating System: An Object-Oriented Approach to the Interconnection of Heterogeneous Applications*. In Proceedings of the Second International Work-shop on Object-Oriented Database Systems, 213-218, September 1988.

[BNPS89] Bertino E., Negri M., Pelaggati G. and Sbatella L. *Integration of Heterogeneous Database Applications Through an Object-Oriented Interface*. Information Systems, 407-420, 1989.

[BOHGM92] Buchman A., Ozsu M. T., Hornick M., Georgakopoulos D. and Manola F. A. *A Transaction Model for Active Distributed Systems.* In Database Transaction Models for Advanced Applications, A. K. Elmagarmid, Ed. Morgan Kaufmann, San Mateo, Calif, 123-158, 1992.

[BR95] Breitbart, Y. and Reyes, T. *Overview of the ADDS System*. In Kim, W. (Ed.), Modern Database Systems - The Object Model, Interoperability, and Beyond, Addison-Wesley, USA, 1995. (Capítulo 33).

[Brz84] Brzezinski, Z. *Unibase - An Integrated Access to Databases*. Proc. 10 th Int'l Conf. Very Large Databases, Morgan Kaufmann, San Mateo, Calif, 1984, pp.388-396.

[Car87] Cardenas, A. F. *Heterogeneous Distributed Database Management: The HD-DBMS*. Proc. IEEE, Vol.75, N0. 5, May 1987, pp. 588-600.

[CGHI+94] Chavathe, S., Garcia-Molina, H., Hammer, J. Ireland, K., Papakonstantinou, Y. Ullman J. and Widom, J. *The TSIMMIS Project: Integration of Heterogeneous Information Sources*, In Proceedings of IPSJ Conference, pp. 7-18, Tokyo, Japan, October 1994.

[Che76] Chen, P. P.: *The Entity-Relationship Model: Toward a Unified View of Data*. ACM Transactions on Database Systems, Vol. 1, March 1976. pp. 9-37.

[CHKR98] Carey, M. J., Haas, M. L., Kleewein, J. and Reinwald, B. *Data Access Interoperability in the IBM Database Family*. In IEEE Computer Society Technical Committee on Data Engineering, Vol. 21, N. 3, September 1998.

[Cho96] Cholvy, L. A. *Differents Definitions of Consistency*, 1996. http://www.cert.fr.francais/deri/cholvy/partie2.

[Cod70] Cood, E.F.: *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, Vol. 13, 1970. pp. 377-387.

[Cod90] Cood, E. F.: *The Relational Model for Database Management* - Version 2. Addison-Wesley, USA, 1990.

[CR94] Chrysanthis, P. K. and Ramamritham, K. *Autonomy Requirements in Heterogeneous Distributed Database Systems*. Proceedings of the 6th International Conference on Management of Data (COMAD'94), December 1994.

[DAT87] Deen, S. M., Amin, R. R. and Taylor, M. C. *Data Integration in Distributed Databases*. In IEEE Transactions on Software Engineering, Vol. SE-13, N. 7 1987.

[Dav98] Davis, T.E.: *Build your own Object Pool in Java to Boost Application Speed*. JavaWorld, http://www.javaworld.com/javaworld/jw-06-1998/jw-06-object-pool.html, Jun.1998.

[DD97] Date, C. J., Darwen, H.: *A Guide to the SQL Standard*. Addison-Wesley, 4 Ed., USA, 1997.

[DH84] Dayal, Umeshwar and Hwang, Hai-Yann: *View Definition and Generalization for Database Integration in a Multidatabase System*. In IEEE Transactions on Software Engineering, Vol. SE-10, N° 6, November 1984.

[DMFV90] DeWitt, D. J., Maier, D., Futtersack, P., Velez, F.: *A Study of Three Alternative Workstation/Server Architectures for Object-Oriented Databases*. In Proc. of the 16 th Int. Conf. on Very Large Data Bases (VLDB'90), Brisbane, Australia, 1990. pp. 107-121.

[EN94] Elmasri, R. and Navathe, S. B. *Fundamentals of Database Systems*. Addison-Wesley, second edition, USA, 1994.

[Esc84] Esculier, C. *The Sirius-Delta Architecture: A Framework for Cooperating Database Systems*. Computer Networks, Vol. 8, No. 1, 1984, pp. 43-48.

[FHM92] Fang, D., Hammer, J., McLeod, D. *An Approach to Behavior Sharing in Federated Database Systems*. pp. 334-346, IWDOM 1992

[FHMS90] Fang, D., Hammer, J., McLeod, D. and Si, A. *Remote-Exchange: An Approach to Controlled Sharing among Autonomous, Heterogeneous Databases Systems*,1990.

[GGO90] Gagliard R., Ganeve M. and Oldano G. *An Operational Approach to the Integration of Distributed Heterogeneous Environments*. In proceedings of the PARBASE-90 Conference, 368-377, Flórida, March 1990.

[GHIP+95] Garcia-Molina, H, Hammer, J. Ireland, K. Papakonstantinou, Y., Ullman, J. and Widom, J. *Integrating and Accessing Heterogeneous Information Sources in TSIMMIS*, Department of Computer Science, Stanford University, 1995.

[GRS97] Gehani, N., Ramamritham, K. and Shmueli, O. *Accessing Extra-Database Information: Concurrency Control and Correctness*, Information Systems, 1997.

[Ham94] Hammer, J. *Resolving Semantic Heterogeneity in a Federation of Autonomous, Heterogeneous Database Systems*. Dissertation presented to the Faculty of Graduate School. University of Southern California, 1994.

[HCF97] Hamilton, G., Cattell, R., Fisher, M.: *JDBC Database Access with Java: A Tutorial and Annotated Reference*, Addison-Wesley, USA, 1997.

[HDRK97] Hamill, S., Dixon, M., Read, B. J. and Kalmus, J. R. *Interoperating Database Systems: Issues and Architectures*. Technical Report, RAL-TR-97-063, CLRC (Central Laboratory of the Research Councils), November 1997.

[Her98] Hermsen, U. *Design and Implementation of an Adaptable Cache in a Heterogeneous Client/Server Environment* (Em alemão). Diploma Thesis, University of Kaiserslautern, Kaiserslautern, Germany, 1998.

[HM85] Heimbigner, D. and McLeod, D. *A Federated Architecture for Information Management*. ACM Trans. Office Information Systems, Vol. 3, No.3, 1985, pp. 253-278.

[HM93] Hammer, J. and McLeod, D. *An Approach to Resolving Semantic Heterogeneity In a Federation of Autonomous, Heterogeneous Database Systems*. International Journal of Intelligent & Cooperative Information Systems, 2(1):51-83, March 1993.

[HMNR95] Härder, T., Mitschang, B., Nink, U., Ritter, N.: *Workstation/Server Architectures for Database/Based Engineering Applications* (Em Alemão). Informatik Forschung & Entwicklung, 1995.

[HMS93] Hammer, J., McLeod, D. and Si, A. *An Intelligent System for Identifying and Integrating Non-Local Objects in Federated Database Systems*, 1993.

[HR98a] Hermsen, U., and Rezende, F.F.: *MEntAs Database Interface Tutorial* (Em alemão). Internal Report, MEntAs Version 0.2, Daimler-Benz AG Research & Technology, Ulm, Germany, 1998.

[IBI97] Information Builders: *EDA/SQL Manuals*. Information Builders, 1997.

[IBM95] IBM Corporation: *DataJoiner: A Multidatabase Server*. White Paper, May,1995.

[IBM97] IBM Corporation: *DB2 DataJoiner Administration Guide*. IBM, 1997.

[IBM97a] IBM Corporation: *DB2 DataJoiner for AIX - Planing, Installation, and Configuration Guide*. IBM, 1997

[Jav97] *Java 1.1*, Third Edition, 1997, USA.

[KDN91] Kaul M., Drosten K., Neuhold E. J. *Viewsystem: Integrating Heterogeneous Information Bases by Object Oriented Views*. In IEEE International Conference on Data Engineering, 2-10, 1991.

[KGCS95] Kim, W., Choi, I., Gala, S., Scheevel, M.: *On Resolving Schematic Heterogeneity in Multidatabase Systems*. In Kim, W. (Ed.), Modern Database Systems - The Object Model, Interoperability, and Beyond, Addison-Wesley, USA, 1995. (Capítulo 26).

[KGKR+93] Kelley, W., Gala, S., Kim, W., Reyes, T. and Graham, B. *Schema Architecture of UniSQL/M Multidatabase System*. In Kim, W. (Ed.), Modern Database Systems - The Object Model, Interoperability, and Beyond, Addison-Wesley, USA, 1995. (Capítulo 30).

[Kim81] Kimbleton, S. R. *Applications and Protocols.* Distributed Systems - Architecture and Implementation, Spring-Verlag, 1981, pp. 308-370.

[KS91] Kim, W. and Seo, J. *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*. IEEE Computer, December, 1991.

[KS95] Kashyap, V. and Sheth, A. *Semantic and Schematic Similarities between Objects in Databases: A Context-based approach*. 1995.

[LMR90] Litwin, W., Mark, L. and Roussopoulos, N. *Interoperability of Multiple Autonomous Databases*. In ACM Computing Surveys, Vol 22, N. 3, September 1990.

[LR82] Landers, T. A. and Rosenberg, R. L. *An overview of multibase*. In Distributed Databases, H.J. Shneider, Ed. Amsterdam, The Netherlands: North-Holland, 1982.

[LZ88] Litwin, W. and Zeroual, A. *Advances in Multidatabase Systems*. Proc. european Teleinformatics Conference - Euteco 88, Research into Networks and Distributed Applications, North-Holland, Amsterdam, 1988, pp. 1, 137-1, 151.

[Mel90] Melton, J. (Ed.) *Database Language SQL 2*. American National Standards Institute, Washington, D.C., USA, 1990.

[MHGHB92] Manola F., Heiler S., Georgakopoulos D., Hornick M., and Brodie M. *Distributed Object Management*. Int. J. Intell. Cooperative Info Syst. 1, June 1992.

[MIR93] Miller, R. J., Ioannidis, Y. E. and Ramakrishnan, R..*Understanding Schemas*. Research Issues in Data Engineering: Interoperability in Multidatabase Systems, 1993.

[MPPLS93] Mitschang, B., Pirahesh, H., Pistor, P., Lindsay, B., Südkamp, S.: *SQL/XNF - Processing Composite Objects as Abstractions over Relational Data*. In: Proc. of the Int. Conf. on Data Engineering, Vienna, Austria, Apr. 1993.

[MRJ99] Missier, P., Rusinkiewicz, M. and Jin, W. *Multidatabase Languages. In Management of Heterogeneous and Autonomous Database Systems*. Edited by Ahmed Elmagarmid, Marek Rusinkiewicz and Amit Sheth, Morgan Kaufmann Publishers, Inc. San Francisco, California, 1999. (Capítulo 7).

[OHE94] Orfali, R., Harkey, D., Edwards, J.: T*he Essential Distributed Objects Survival* Guide. John Willey & Sons, USA, 1994.

[Oli99] Oliveira, G.S.: *Uma Interface Independente de Plataforma para Acesso a Banco de Dados Heterogêneos*. Dissertação de Mestrado, Universidade Federal de Pernambuco, Recife, Brasil, 1999.

[OMG92] Object Management Group. *The Common Object Request Broker Architecture and Specification (CORBA)*, OMG, Framingham, USA, 1992.

[OMG95] Object Management Group. *The Common Object Request Broker Architecture and Specification* - Rev. 2.0, Technical Report, OMG, Framingham, USA, 1995.

[OPSY98] Olson, S., Pledereder, R., Shaw, P. and Yach, D. *The Sybase Architecture for Extensible Data Management*. In IEEE Computer Society Technical Committee on Data Engineering, Vol. 21, N. 3, September 1998.

[ORA97] ORACLE Corporation: *Transparent Gateway Manuals*. ORACLE, 1997.

[OV91] Ozsu, T. and Valduriez, P. *Principles of Distributed Database Systems*, Prentice-Hall, 1991.

[Pire97] Pires, Paulo de Figueiredo: *HIMPAR, Uma Arquitetura para Interoperabilidade de Objtetos Distribuídos*. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, 1997.

[Rez97] Rezende, F.F. *Transaction Services for Knowledge Base Management Systems* - Modeling Aspects, Architectural Issues, and Realiyation Techniques. Infix Verlag, Germany, 1997. (DISDBIS 35).

[Rez98] Rezende, F. F.: *LKD-Datenbank, Sichtdefinitionen*. Daimler-Benz-AG - Forschung &Technologie, Ulm, Germany, 1998.

[Rez98a] Rezende, F. F.: *PSD-Datenbank, Sichtdefinitionen*. Daimler-Benz-AG - Forschung &Technologie, Ulm, Germany, 1998.

[Rez98b] Rezende, F. F.: *MENTAS-Datenbank,Entity-Relationship-Datenmodell und Data Dictionary*. Daimler-Benz-AG - Forschung &Technologie, Ulm, Germany, 1998.

[RH96] Rezende, F. F., Härder, T.: *An Approach to Multi-User KBMS in Workstation/Server Environments*. In: 11 Simpósio Brasileiro de Banco de Dados (SBBD'96), São Carlos, Brasil, Outubro de 1996. pp.58-72.

[RH98] Rezende, F. F.; Hergula, K.: *The Heterogeneity Problem and Middleware Technology: Experiences with and Performance of Database Gateways*. In: Proceedings of the 24th VLDB Conference, New York, USA, 1998.

[RHO+98] Rezende, F. F, Hermsen, U., Oliveira, G. S., Pereira, R. C. G., Rütschlin, J. and Schneider, P. *The Database Access Interface in MEnTAs: Architecture und Functionality.* Technischer Bericht FT3/E-1998-003, Daimler-Benz AG, Forschung und Technologie 3, Prozeßkette Produktentwicklung, Ulm, Germany, 1998.

[RHS98] Rezende, F. F., Hergula, K., Schneider, P.: *A Comparative Analysis and Performance of Database Gateways*. Technischer Bericht FT3/E-1998-001, Daimler-Benz AG, Forschung und Technologie 3, Prozeßkette Produktentwicklung, Ulm, Germany, 1998.

[Roc96] Rocha, Helder. *Diferenças entre Java e C/C++*; Lan times Brasil (artigo submetido em 16 de maio de 1996, ainda não publicado). http://www.dsc.ufpb.br/~helder/java/javavsg.html

[RR99] Ram,S. and Rameshq, V. *Schema Integration: Past, Present, and Future. Management of Heterogeneous and Autonomous Database Systems*. Edited by Ahmed Elmagarmid, Marek Rusinkiewicz and Amit Sheth, Morgan Kaufmann Publishers, Inc. San Francisco, California, 1999. (Capítulo 5).

[Rüt99] Rütschlin, J.: *Security Management in a Heterogeneous Database Environment* (Em Alemão). Diploma Thesis, University of Ulm, Ulm, Germany, 1999.

[Rym96] Rymer, J. R. *The Muddle in the Middle*. Byte Magazine, April 1996.

[SADD+93] Shan, M., Ahmed R., Davis, J., Du, W. and Kent W. *Pegasus: A Heterogeneous Information Management System*. In Kim, W. (Ed.), Modern Database Systems - The Object Model, Interoperability, and Beyond, Addison-Wesley, USA, 1995. (Capítulo 32).

[SBD+81] Smith, J. M., Bernstein, P. A., Dayal, U., Goodman N., Landers, T. Lin, K. W. T., and Wong, E., *Multibase - Integrating Heterogeneous Distributed Database Systems* in Proc. AFIPS NCC, 1981, pp. 487-499.

[SK93] Sheth, Amit and Kashyap, Vipul. *So Far(Schematically) Yet So Near (Semantically)*, DS - 5 1992: 283 - 312.

[SL90] Sheth, A. P., and Larson, J. A. *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*. In ACM Computing Surveys, Vol. 22, N. 3 September 1990.

[Spa82] Spaccapietra, S. *Scoop - A System for Cooperation Between Existing Heterogeneous Distributed Databases and Programs*. Database Eng. Vol. 5, No. 4, 1982, pp. 288-293.

[SRK91] Sheth, A., Rusinkiewicz, M. and Karabatis, G. *Using Polytransactions to Manage Interdependent Data*, 1991.

[Sta85] Staniszkis, W. *Architecture of the Network Data Management System*. Proc. Third Int'l Seminar on Distributed Data Sharing Systems, North-Holland, Amsterdam, 1985, pp.57-75.

[Str84] Strocker, P. M. *Proteus: A Heterogeneous Distributed Database Project*. Cambridge Univ. Press, 1984, pp. 125-150.

[SUN98] Sun Microsystems Inc. World wide web JDBC page at http://java.sun.com/products/jdbc/overview.html, 1998.

[SUN98a] Sun Microsystems Inc. World wide web RMI page at http://java.sun.com/products/jdk/rmi/index.html, 1998.

[Tak83] Takizawa, M. *Heterogeneous Distributed Database System: JDDBS*. Data Eng., Vol. 6, No.1, 1983, pp. 58-62.

[TTCB+90] Thomas, G., Thompson, G. R., Chung, C., Barkmeyer, E., Carter, F., Templeton, M., Fox, S. and Hartman, B. *Heterogeneous Distributed Database Systems for Production Use*. In ACM Computing Surveys, Vol 22, N. 3, September 1990.