Alvaro Cesar Pereira Barbosa

Middleware
para Integração de Dados Heterogêneos
Baseado em
Composição de Frameworks

TESE DE DOUTORADO

Departamento de Informática

Rio de Janeiro, 03 de Maio de 2001

Alvaro Cesar Pereira Barbosa

Middleware
para Integração de Dados Heterogêneos
Baseado em
Composição de Frameworks

Tese apresentada ao Departamento de Informática da PUC-Rio como parte dos requisitos para obtenção do título de Doutor em Informática: Ciência da Computação.

Orientador: Rubens Nascimento Melo
Co-Orientador: Carlos José P de Lucena

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 03 de Maio de 2001.

A minha esposa Joelma, pelo seu carinho,
pelas minhas ausências, pelo incentivo constante
e pela participação ativa na realização deste trabalho.

A meus pais

e a meu avô (*in memorian*),

pela educação recebida e pelo exemplo de vida.

AGRADECIMENTOS

A Deus, pela vida, pelas conquistas e por me dar forças para superar os momentos difíceis;

Ao prof. Rubens Nascimento Melo pela oportunidade da realização deste curso, pelos ensinamentos e pelas valiosas contribuições no fechamento deste trabalho;

Ao prof. Carlos José Pereira de Lucena pela confiança depositada, pelas críticas e sugestões, e por sempre ter uma resposta para as minhas dúvidas;

Ao prof. Sérgio Carvalho (*in memorian*) pelos sábios conselhos, que me permitiram um melhor discernimento na definição inicial desta pesquisa;

Aos profs. Antônio Furtado, Sidney Dias e Luiz Tucherman, pela participação na banca e pelas sugestões incorporadas na versão final deste trabalho;

Aos demais professores do DI/PUC-Rio pelo conhecimento recebido;

Ao amigo de luta Fábio Porto pelas inúmeras horas de discussões que foram fundamentais para a conclusão deste trabalho;

Aos demais amigos da PUC-Rio que contribuíram direta ou indiretamente na realização deste trabalho, em especial a: Carolina, Fausto, Georgia, José Antônio (Tonho), Sean, Viviane Torres;

Aos meus familiares e demais amigos extra-PUC pelo incentivo e pela torcida constante;

Aos funcionários dos diversos setores da PUC-Rio, em especial a Carmen, Isabela e Ruth (DI), Vera (FPLF), Luiz (CEAD);

Ao Departamento de Informática da Universidade Federal do Espírito Santo (UFES) pelo afastamento durante estes quatro anos, bem como à UFES e à CAPES pelo apoio financeiro.

RESUMO

O desenvolvimento de sistemas para integração de dados heterogêneos e distribuídos na Internet é um problema atual e complexo. Isso, devido à existência de muitos usuários com diferentes perfis e interesses, necessitando acessar diferentes tipos e modelos de dados e requerendo diferentes funcionalidades para os sistemas de integração.

Na literatura são encontrados diversos sistemas voltados para integração. Porém tais sistemas ou são demasiadamente específicos, atendendo a uma única aplicação, ou extremamente genéricos para atender às diversas situações de integração. As duas situações levam a problemas quando da necessidade de adaptá-las a uma nova aplicação.

Para conciliar de modo eficaz a diversidade de interesses dos usuários no acesso à grande variedade de dados existentes, torna-se necessário pesquisar novos sistemas de integração de dados, que possam ser moldados a um contexto específico. Este é o objetivo deste trabalho, através do desenvolvimento do CoDIMS *(Configurable Data Integration Middleware System*): um ambiente flexível e configurável que possibilita gerar sistemas configurados para a integração de dados heterogêneos e distribuídos.

A essência do ambiente CoDIMS é um componente de Controle que disponibiliza mecanismos para configuração física e lógica. A configuração física é realizada através da seleção e integração de um conjunto adequado de componentes, que implementam serviços *middleware* de integração de dados, denominados DIMS (*Data Integration Middleware Services*). Componentes DIMS são baseados em serviços típicos de gerência de dados, tais como os existentes nos SGBDs e são desenvolvidos através da técnica de *frameworks* para prover maior flexibilidade e facilidade de adaptação dos mesmos a uma aplicação específica. A configuração lógica é resolvida utilizando-se uma abordagem original onde a lógica de execução dos serviços DIMS é abstraída em uma representação baseada no conceito de *workflow*.

ABSTRACT

Developing heterogeneous data integration systems in the Internet is a current and complex problem. There are many users with different skills and interests, needing to access different data types and data models, and requiring different functionality from the integration systems.

In literature, there are a large number of integration systems reported. However, such systems are either specifically developed according to one application, or extremely generic to cope with different situations. Both cases bring problems when it is necessary to customise them to a new application.

In order to conciliate, in an efficient way, the interests of different users in accessing a great variety of existing data, it is necessary to research new models for data integration systems, which could be tailored to a specific context. This is the goal of the present work: the development of the CoDIMS (Configurable Data Integration Middleware System), a flexible and configurable environment for generating configured systems for the integration of distributed and heterogeneous data.

The essence of the CoDIMS environment is the Control component, which implements mechanisms for physical and logical configuration. The physical configuration is obtained through the selection and integration of an adequate set of components that implement Data Integration Middleware Services - DIMS. DIMS Components are based on typical data management services, like the existing ones in the DBMSs. In order to provide more flexibility and facility in the adaptation to a specific application, DIMS Components are modelled using the framework technique. The logical configuration is obtained using an original approach, where DIMS services execution schedule is abstracted in a representation based on the workflow concept.

SUMÁRIO

1. INTRODUÇÃO……...1

1.1 – Contexto……...1

1.2 – Tópicos de Pesquisa……..3

1.2.1 – Sistemas Configuráveis e Extensíveis……3

1.2.2 – *Frameworks*……4

1.2.3 – Banco de Dados Flexíveis……..……5

1.2.4 – Componentização……..……...6

1.2.5 – Aplicações *Web*……..……...7

1.3 – Objetivo da Tese……...8

1.4 – Motivação……..……..9

1.4.1 - Sistema HEROS……..……10

1.4.2 - Projeto ECOHOOD……..……11

1.4.3 - Projeto ECOBASE……..……12

1.5 – Visão Geral da Solução Proposta……..……...13

1.6 – Contribuições Esperadas……...……17

1.7 – Organização da Tese……..……18

2. TRABALHOS RELACIONADOS……..……19

2.1 – Introdução……..……...19

2.2 – Strudel……..……20

2.3 – Araneus……..……..21

2.4 – Le Select……..……23

2.5 – Disco……..……..24

2.6 – TSIMMIS……..……26

2.7 – MIRO-Web……..……27

2.8 – Garlic……..……..28

2.9 – MOCHA……..……..29

2.10 – HEROS……..……31

2.11 – Quadro-Resumo dos Sistemas……..……33

2.12 – Síntese do Capítulo……..……...34

3. CONCEITOS……35

3.1 – Introdução……..……..35

3.1.1 – Padrões de Interoperabilidade……...37

3.1.2 – *Middleware* para Integração de Dados……....38

3.2 – SGBDHs…….……..…40

3.2.1 – Integração de Esquemas em SGBDHs……..42

3.3 – Informações na *Web*…….……...46

3.3.1 –Tecnologias Utilizadas na *Web*……..…47

3.3.2 – Bancos de Dados e *Web*…….……...49

3.3.3 – Integração de Informações na *Web*……..…52

3.4 – Desenvolvimento de Sistemas de *Software*…….……..…54

3.4.1 – Arquitetura de *Software*……..56

3.4.2 – Componentes……...58

3.5 – *Frameworks*……60

3.5.1 – Classificação dos *Frameworks*……..63

3.5.2 – Evolução dos *Frameworks*……65

3.5.3 – Integração de *Frameworks*……..…66

3.6 – *Patterns*……...67

3.6.1 – Classificação dos *Patterns*……....68

3.6.2 – Alguns *Patterns* para Integração……...69

3.7 – Síntese do Capítulo……...73

4. O AMBIENTE CoDIMS……...74

4.1 – Introdução……74

4.1.1 – Exemplos de Configuração……..…75

4.2 – Visão Geral do Ambiente……77

4.2.1 – Configuração: Visão Conceitual……..78

4.2.2 – Configuração: Visão Operacional……78

4.3 – O Componente Controle……..……..79

4.4 – O Componente Gerência de Metadados……...80
4.4.1 – Metadados das Fontes de Dados…….80
4.4.2 – Metadados de Exportação…….81
4.4.4 – Metadados Global…….82
4.5 – O Componente Processamento de Consulta…….82
4.5.1 – Analisador…….84
4.5.2 – Re-escritor…….84
4.5.3 – Otimizador…….85
4.5.4 – Processamento……..85
4.6 – O Componente Acesso aos Dados…….86
4.7 – Os Componentes Gerência de Transação, Controle de Concorrência e Gerência de Regras……86
4.7.1 – Gerência de Transação……..87
4.7.2 – Controle de Concorrência…….87
4.7.3 – Gerência de Regas…….88
4.8 – O Processo de Configuração……..88
4.9 – Síntese do Capítulo…….90

5. ESPECIFICAÇÃO DO CoDIMS……...91
5.1 – Introdução……..91
5.2 – Os Componentes do CoDIMS…….93
5.3 – O Componente Controle…….95
5.4 – O Componente Acesso aos Dados…..98
5.5 - O Componente Metadados……..99
5.6 – O Componente Processamento de Consulta…..101
5.7 – Os Componentes Gerência de Transação, Controle de Concorrência e Gerência de Regras…….102
5.8 – Diagramas de Casos de Uso……..102
5.9 - Diagramas de Seqüência…….105
5.10 – Síntese do Capítulo…….. …..107

6 – ESTUDO DE CASO.....108
6.1 – Introdução.....108
6.2 – Descrição da Aplicação.....108
6.3 – Projeto do Sistema.....110
6.4 – Configuração do Sistema.....113
6.5 – Projeto da Aplicação.....115
6.6 – Utilização do Sistema.....119
6.7 – Incluindo Novas Facilidades no Sistema Configurado.....122
6.8 – Implementação do Sistema.....124
6.9 – Síntese do Capítulo.....125

7 – CONCLUSÕES, CONTRIBUIÇÕES E TRABALHOS FUTUROS.....126
7.1 – Conclusões.....126
7.2 – Contribuições.....129
7.3 – Trabalhos Futuros.....130

REFERÊNCIAS BIBLIOGRÁFICAS.....132
ANEXO A.....147

ILUSTRAÇÕES

Figura 1.1 – Arquitetura de três camadas e o middleware……..……14
Figura 1.2 – CoDIMS: Um ambiente configurável baseado em componentes..15
Figura 1.3 – O CoDIMS e seus componentes……..15
Figura 2.1 – Arquitetura do Strudel……..……21
Figura 2.2 – Arquitetura do Araneus..……..……22
Figura 2.3 – Arquitetura do Le Select……..……23
Figura 2.4 – Arquitetura do DISCO……..……25
Figura 2.5 – Arquitetura do TSIMMIS……..……26
Figura 2.6 – Arquitetura do MIRO-Web……..……27
Figura 2.7 – Arquitetura do Garlic…..……..……29
Figura 2.8 – Arquitetura do MOCHA..……..……30
Figura 2.9 – Arquitetura do HEROS……..……..……31
Figura 3.1 – Integração de esquemas……..……..……43
Figura 3.2 – Arquitetura da Web……..……48
Figura 3.3 – Arquitetura de três camadas……..……50
Figura 4.1 – Exemplo de Configuração……..……76
Figura 4.2 – Exemplo de Configuração……..……76
Figura 4.3 – CoDIMS: Um ambiente configurável baseado em componentes..77
Figura 4.4 – O CoDIMS e seus componentes.……..……79
Figura 4.5 – Arquitetura de Metadados em 4 níveis…..……..……81
Figura 4.6 – Processamento de Consulta……...83
Figura 5.1 – Diagrama de Componentes do CoDIMS……..……93
Figura 5.2 – As Fachadas dos Componentes do CoDIMS……..……94
Figura 5.3 – Diagrama de classes do componente Controle……...95
Figura 5.4 – O componente Acesso aos Dados.……..……98
Figura 5.5 – Diagrama de classes do componente Metadados ……..……100
Figura 5.6 – Componente Processamento de Consulta..……101
Figura 5.7 – Caso de Uso: Projeto do Sistema…….……...103
Figura 5.8 – Caso de Uso: Configuração……..103
Figura 5.9 – Caso de Uso: Projeto da Aplicação ……104

Figura 5.10 – Caso de Uso: Utilização……..……..104
Figura 5.11 – Diagrama de Seqüência: Configuração…….……105
Figura 5.12 – Diagrama de Seqüência: Definir *Workflow*.……105
Figura 5.13 – Diagrama de Seqüência: Definir Metadados……106
Figura 5.14 – Diagrama de Seqüência: Executar Consulta……106
Figura 5.15 – Diagrama de Seqüência: Enviar Mensagem……107
Figura 6.1 – Diagrama de Componentes……..……..…..…….110
Figura 6.2 – Diagrama de Fachadas……..110
Figura 6.3 – O componente Controle……..……...111
Figura 6.4 – O componente Processamento de Consulta.……112
Figura 6.5 – O componente Metadados ……..…..……112
Figura 6.6 – O Componente Acesso aos Dados...…….……113
Figura 6.7 – Grafo de Consulta……..……120
Figura 6.8 – Árvore de operação…… ……..…..……120
Figura 6.9 – Árvore de execução……....……121

Capítulo 1 - Introdução

Este trabalho propõe a especificação do CoDIMS (Configurable Data Integration Middleware System): um ambiente flexível e configurável para integração de dados heterogêneos e distribuídos. Neste capítulo são descritos o contexto, a motivação, os objetivos, os tópicos de pesquisa relacionados, uma visão geral da solução proposta e as contribuições esperadas. No final do capítulo é apresentada a organização deste trabalho.

1.1 – Contexto

O ambiente computacional atual é composto por sistemas de informações autônomos, distribuídos e heterogêneos. Os sistemas são autônomos porque sua construção, manutenção e operação ocorrem de forma independente e sem a preocupação com integração com outros sistemas. Esta autonomia traz como conseqüência um ambiente altamente heterogêneo, pois favorece a utilização de diferentes tecnologias de *hardware* e de *software* e de diversos fornecedores no desenvolvimento de sistemas. Se por um lado a *internet*, a disponibilidade das redes de alta velocidade e as recentes tecnologias de comunicação proporcionam uma maior facilidade e diminuição das distâncias no que se refere à distribuição, localização e acesso às informações, por outro lado estimulam uma crescente divulgação de novas informações e em diversas mídias, realimentando o problema da heterogeneidade e distribuição.

Estas ilhas de informações, autônomas e heterogêneas, demandam a construção de pontes que possam integrá-las. Muitas aplicações atualmente existentes, necessitam incorporar informações de diversas fontes de dados onde os dados relacionados podem diferir na representação ou na tecnologia utilizada. Muito embora sempre exista a opção de se refazer tais sistemas para facilitar o processo de integração de informações, os riscos, os custos e o

tempo envolvidos podem ser proibitivos. As dificuldades de adaptá-los para se comunicarem e compartilharem informações com outros sistemas mais novos e avançados tecnologicamente são grandes e complexas. Muitos sistemas atualmente existentes foram construídos e bem customizados para atender às necessidades específicas das empresas e, apesar de poderem estar desatualizados tecnologicamente, a importância dessas aplicações para a empresa continua nova. Assim, torna-se importante o desenvolvimento de sistemas que permitam o acesso e a integração de dados existentes em diferentes fontes.

A abordagem utilizada para a construção de sistemas de integração de dados é semelhante a utilizada nos ambientes cliente-servidor de três camadas: uma camada das fontes de dados a serem integradas, uma da aplicação cliente e uma camada intermediária com a função de integração dos dados denominada *middleware*. A utilização de sistemas *middleware* libera os desenvolvedores de aplicações de ter que se envolver com os detalhes de cada fonte de dados e das complexidades dos ambientes computacionais.

Por sua vez, para o desenvolvimento e a manutenção de sistemas de *software*, como no caso dos sistemas integradores de dados, devem ser utilizadas técnicas que facilitem o projeto e que possibilitem a redução do tempo de implementação e de manutenção. Uma alternativa importante é a possibilidade de reuso dos projetos e dos códigos já existentes, em vez de se recriar os mesmos processos repetidamente. Este procedimento se torna necessário devido às rápidas mudanças tecnológicas e às constantes alterações nos requisitos das aplicações, o que implica uma revisão periódica dos projetos, na tentativa de não permitir que os sistemas se tornem obsoletos em pouco tempo. Um outro fator é uma maior prudência financeira em novos investimentos por parte das empresas, onde os custos de desenvolvimento de novas aplicações e de manutenção devem ser reduzidos.

Este quadro produz uma demanda por:

- Desenvolver novos tipos de sistemas para o acesso e integração de informações armazenadas em diferentes fontes de dados, de maneira

transparente à localização e a forma de armazenamento dos dados, e que forneçam uma visão uniforme e integrada de suas informações;
- Utilizar novas tecnologias de desenvolvimento de *software* que permitam produzir aplicações de forma rápida, flexível, confiável e de menor custo.

Para o atendimento destas demandas, torna-se fundamental a pesquisa e a proposta de novas soluções que levem em conta as seguintes questões:
- Que abordagem deve ser utilizada para o desenvolvimento de sistemas de integração de dados heterogêneos e distribuídos?
- Que tipo de estrutura deve ser utilizada para permitir que estes sistemas possam ser construídos mais facilmente e adaptados para atender aos novos requisitos das aplicações?
- Que tecnologia de desenvolvimento de *software* utilizar para construir sistemas em um tempo reduzido, de baixo custo e confiáveis?

A busca de respostas e opções para atender a estas questões, nos levaram a procurar alguns tópicos de pesquisas relacionados, dentro das áreas envolvidas, que são apresentados a seguir.

1.2 – Tópicos de Pesquisa

Temos observado e acompanhado os recentes esforços e resultados de pesquisas em outras áreas que, se utilizados de forma apropriada em um mesmo ambiente, passam a oferecer uma nova abordagem para o desenvolvimento de sistemas de integração de dados. São eles:

1.2.1 - Sistemas Configuráveis e Extensíveis

A necessidade dos sistemas passarem a atender uma grande variedade de aplicações com requisitos diferentes, incentivou a pesquisa para o desenvolvimento de sistemas configuráveis e extensíveis. Os grandes sistemas comercialmente disponíveis têm sido assim desenvolvidos. Uma forma de se construir sistemas configuráveis é disponibilizar todas as suas funcionalidades que podem ser customizadas via parametrização para o atendimento de uma

aplicação específica. No caso de sistemas extensíveis, o objetivo é facilitar a incorporação de novas funcionalidades.

Uma abordagem atualmente utilizada no desenvolvimento de sistemas propõe que os mesmos sejam construídos a partir de uma arquitetura comum, que possa ser adaptada para atender aos requisitos de diferentes aplicações sem que se tenha que implementar sistemas chamados *heavy weight* [BT95]. Sistemas *heavy weight*, ou de objetivo geral, são projetados para atender às várias situações diferentes fazendo com que, para uma certa aplicação, parte do código implementado se torne desnecessário, o que provoca aumento do custo de desenvolvimento e de manutenção, maiores recursos de *software* e de *hardware* e também problemas de performance.

Já os sistemas *light weight*, omitem uma ou mais características dos sistemas *heavy weight* e especializam o desenvolvimento de suas próprias funcionalidades para maximizar performance [BT95]. Tais sistemas são projetados para que somente o código necessário a uma aplicação específica seja disponibilizado, sendo, então, denominados sistemas enxutos ou de aplicação específica. Assim, torna-se relevante projetar e implementar sistemas *light weight* que sejam flexíveis e que possam ser configurados para poder atender aos diferentes requisitos das aplicações. Uma solução é construir tais sistemas a partir de uma base comum, utilizando um mecanismo de integração de componentes, possibilitando disponibilizar uma família de sistemas com um menor número possível destes componentes para cada aplicação específica e, principalmente, permitir o reuso de componentes já existentes. Para o atendimento destes objetivos, uma abordagem é a utilização da técnica de *framework* [FS97] [John97], por oferecer uma infra-estrutura adequada para a solução destes problemas.

1.2.2 - *Frameworks*

Framework é uma arquitetura semi-completa que pode ser instanciada para produzir aplicações customizadas, permitindo o reuso de análise, de projeto e de código [FSJ99a]. Novas técnicas vêm sendo estudadas e propostas nas áreas de Engenharia de *Software* e de Orientação a Objetos, visando atingir

uma forma de reuso melhor e mais eficiente, além da disponibilidade de sistemas mais flexíveis. Embora seja prática comum a construção de sistemas a partir de outros já existentes, não é levado em consideração, durante o projeto, a possibilidade de sua reutilização no futuro, nem a possibilidade de integração com outros sistemas. Reuso de *software* é uma abordagem que visa a redução do esforço de desenvolvimento e, quando aplicado eficientemente, pode evitar que sejam recriadas e revalidadas soluções comuns para os requisitos de aplicações recorrentes.

A utilização de *frameworks* têm se tornado cada vez mais freqüente no processo de desenvolvimento dos sistemas atuais, sendo uma proposta atraente e interessante não só por permitir reuso, mas também para a construção de sistemas configuráveis. O uso de *frameworks* no projeto de sistemas de integração de dados heterogêneos permite construir sistemas com uma maior flexibilidade e que podem ser adaptados com maior rapidez e facilidade para atender aos requisitos de novas aplicações.

Em grandes sistemas, é prática comum que o mesmo seja dividido em subsistemas e, por analogia, também em *frameworks* existe a proposta de dividi-los em *frameworks* menores chamados *framelets* [PK99] e desenvolver aplicações através da integração de *framelets*. Mas, composição de *frameworks* é ainda um assunto novo e aberto na comunidade científica, podendo levar a diversos problemas [MB97] [MB98]. Um destes problemas é que *frameworks* são geralmente desenvolvidos para reuso por extensão com a escrita de novos códigos específicos para a aplicação e não para composição com outros componentes de *software* [FS97]. Diversas propostas para composição de *frameworks* têm sido apresentadas na literatura [MB97] [Eske99a] [Matt00], mas não existem soluções bem definidas: elas dependem do domínio da aplicação, do problema a ser resolvido e dos *frameworks* envolvidos.

1.2.3 - Banco de Dados Flexíveis

Novas técnicas têm sido pesquisadas e aplicadas de forma a baixar os custos e o tempo para se implementar e manter sistemas, além de permitir que os

mesmos sejam flexíveis o suficiente para poderem permitir o armazenamento e o acesso às informações de forma eficiente em diferentes domínios de aplicações. Inicialmente foram propostos a criação de Sistemas de Gerência de Banco de Dados (SGBD) extensíveis. O EXODUS [CDG+90] e o GENESIS [BBR+90] foram os primeiros projetos a seguir esta linha, propondo um modelo mais geral para a construção de SGBDs específicos, através de geradores de sistemas [Catt94]. Outros projetos como Open OODB [WBT92], P2 [BT95] e Shore [Shor97] continuaram a pesquisa nesta área. P2 e Shore são os sucessores do GENESIS e do EXODUS respectivamente.

Para a integração de SGBDs foram propostos os Sistemas de Gerência de Bancos de Dados Heterogêneos (SGBDHs) [SL90], que apresentam os serviços típicos de um SGBD e que são estendidos para este fim. A aplicação destes serviços em sistemas integradores permite remover o código das aplicações e o delegar ao sistema que passa a incorporar as dificuldades e o código especializado associado à aplicação. Com a atual necessidade de se integrar não somente SGBDs, mas também dados armazenados em outras formas, como no caso da *Web*, surgiu a necessidade de se pesquisar e desenvolver novos tipos de sistemas.

Como apresentado em [BBC+98], [SSU96] e em [SZ97], são necessárias novas soluções e avanços na tecnologia de bancos de dados para a integração de informações heterogêneas, devendo a pesquisa em banco de dados contribuir mais nesta área. De acordo com [SZ97], extensibilidade e componentização são tópicos de pesquisa que devem ser atacados. É necessário procurar abrir as arquiteturas de banco de dados onde novos serviços possam ser incorporados e permitir configurar a funcionalidade de maneira mais flexível, de acordo com as necessidades da aplicação.

1.2.4 - Componentização

O desenvolvimento de sistemas baseados em componentes tem como objetivo fornecer aos usuários e desenvolvedores de *software* os mesmos níveis de interoperabilidade tipo "*plug-and-play*" encontrados na confecção de circuitos integrados [OHE96]. Componentes são entidades de *software* caracterizados

pela dissociação entre especificação e implementação, que interagem com outros componentes através de interfaces e que são projetados visando composição [ND95]. A comunicação entre os componentes é mediada por um elemento que recebe as chamadas e as repassa para o componente destino. Os modelos de interoperabilidade de componentes propostos, como *CORBA* [OMG00], *DCOM* [OHE96] e *Enterprise JavaBeans* [Java01] permitem a comunicação entre componentes de forma transparente à localização e heterogeneidades das redes, protocolos, aplicações, linguagens, ferramentas e sistemas operacionais.

O desenvolvimento de *software* baseado em componentes vem tendo destaque atualmente. Componentes de *software* podem ser desenvolvidos separadamente, testados, compilados e posteriormente integrados em uma aplicação. Os desenvolvedores de aplicações podem se abstrair dos detalhes de implementação dos componentes e se concentrarem nos requisitos fundamentais do sistema. No caso de manutenção, é mais fácil e rápido trocar um componente, que fazer o mesmo procedimento em sistemas monolíticos. Com os padrões de interoperabilidade e a facilidade da *Internet*, componentes podem ser mais facilmente compartilhados e reusados, evitando-se refazer soluções já existentes e disponíveis.

1.2.5 – Aplicações *Web*

Com a disponibilidade da *Internet*, das recentes tecnologias de comunicação e das redes de alta velocidade, houve uma maior facilidade e diminuição das distâncias no que se refere à distribuição, localização e acesso às informações. Mas, por outro lado, estimulou-se uma crescente divulgação de novas informações e em diversas mídias, provocando por sua vez um aumento do número das fontes de dados e da heterogeneidade entre as mesmas.

Recentemente, tem-se aumentado a demanda por aplicações voltadas para a *Web* e diversos sistemas estão sendo propostos e desenvolvidos, seja para disponibilizar, acessar, procurar ou integrar informações [Frat99]. Integração de informações disponíveis na *Web* é um assunto atual e aberto na comunidade científica. A *Web* é uma tecnologia nova, caracterizada por um

grande recurso de informação globalmente distribuída e multi-plataforma, ainda de forma não muito organizada. Tem sido muito pesquisado como integrá-la à tecnologias já existentes e bem definidas, como também, à novas tecnologias. Padrões têm sido pesquisados e propostos para facilitar a troca de informações entre sistemas, como o uso de *XML* [XML01]. Existem, ainda, os padrões de interoperabilidade em *GIS* [OGC98] e de documentos para a área financeira e de saúde. Na área de integração de dados, diversas propostas de diferentes abordagens têm sido apresentadas, podendo ser baseadas em: linguagens de consulta, sistemas de recuperação de informação, mediadores e em serviços de gerência de banco de dados.

1.3 – Objetivo da Tese

O objetivo da tese é apresentar uma nova abordagem para a construção de sistemas de integração de dados, através da proposta, especificação e desenvolvimento do CoDIMS *(Configurable Data Integration Middleware System*): um ambiente flexível e configurável que possibilita gerar sistemas configurados para a integração de dados heterogêneos e distribuídos.

A configuração é realizada através da seleção e integração de um conjunto adequado de componentes, que implementam serviços *middleware* de integração de dados, denominados DIMS (*Data Integration Middleware Services*). Componentes DIMS são baseados em serviços típicos de gerência de dados, tais como os existentes nos SGBDs e são desenvolvidos através da técnica de *frameworks* para prover maior flexibilidade.

A configuração permite a cada sistema configurado resultante utilizar o menor número de componentes possível, de acordo com a especificidade da aplicação, minimizando o código a ser utilizado e, por sua vez, o tamanho do sistema, com o intuito de fornecer melhor performance e facilidade de manutenção.

Portanto, a abordagem do CoDIMS pode ser resumida na seguinte frase: "*what you need is only what you get" (WYNWYG)*. Isto possibilita a construção de sistemas *middleware* para a integração de dados da forma *light weight*.

A essência do ambiente é um componente de Controle, que permite definir e validar a configuração do sistema configurado e gerenciar a invocação dos serviços durante a execução do referido sistema. Como os serviços oferecidos diferem de uma configuração para outra, torna-se necessário definir quais serviços e em que ordem devem ser executados. Para o escalonamento dos serviços é utilizado um mecanismo baseado no conceito de *workflow* [JB96] [Koib97] [Lawr97]. Na realidade, um mesmo sistema configurado pode ter diferentes escalonamentos, para atender às diferentes requisições da mesma aplicação.

A abordagem proposta traz inovações através dos mecanismos de configuração física e lógica, permitindo construir sistemas configurados apenas com os serviços necessários e adequados à aplicação. O mecanismo de configuração física determina quais são os serviços oferecidos e quais os que serão requisitados por cada componente. O mecanismo de configuração lógica, através do *workflow*, determina a ordem de execução destes serviços.

Embora existam na literatura diversas propostas para o problema de integração de dados, como será visto a seguir, não foram encontrados sistemas flexíveis e configuráveis através do uso de componentes e de *frameworks*, tal como proposto nesta nova abordagem.

1.4 – Motivação

Na literatura são encontrados diversos sistemas para a integração de dados heterogêneos e distribuídos: Araneus [MAM+98], DISCO [TAB+97] [TRV98], Garlic [CHS+97] [HMN+99], Infomaster [GKD97], InfoSleuth [BBB+97], Jedi [HFAN99], Le Select [LeSe99], MIRO-Web [FGL+98], MOCHA [MR00], Strudel [FFK+97] [FFL+00], TSIMMIS [MHI+97]. Alguns deles, os mais citados na literatura, serão apresentados e discutidos no capítulo 2 deste trabalho. O aparente vasto número de soluções pode, erroneamente, indicar uma área de pesquisa já resolvida. Pelo contrário, é uma área cada vez mais importante e onde não existe uma solução geral que seja adequada ou que se ajuste aos diversos problemas de integração, o que se constata pelo surgimento de novas propostas.

Estes sistemas geralmente são projetados e construídos para atender a um domínio de aplicação específico e para a sua utilização em outras aplicações são necessárias alterações em quase todo o sistema ou, até mesmo, a construção de um novo. Os sistemas mais abrangentes que procuram atender a várias situações diferentes são grandes e pesados, pois se propõem a disponibilizar todas as funcionalidades do sistema, necessitando da presença de todos os seus módulos, que nem sempre serão utilizados. O seu desenvolvimento, certamente, se dá com um maior custo e tempo, refazendo, muitas vezes, soluções já existentes.

Existe uma variedade muito grande de problemas, soluções e usuários nesta área, havendo a necessidade de se pesquisar novas soluções, flexíveis e configuráveis, que possam ser moldadas para um contexto específico.

Nos últimos anos, nosso grupo de pesquisa em integração de dados tem acompanhado e realizado experiências com as tecnologias emergentes descritas anteriormente, que podem contribuir com novas propostas de solução para o problema de integração de dados. Os primeiros trabalhos foram em integração de sistemas de bancos de dados heterogêneos e distribuídos e o desenvolvimento do sistema HEROS, um SGBDH orientado a objetos [DPSM93], [Silv94], [Ucho94], [BM99], [BT99]. Mais recentemente, foram pesquisados e utilizados padrões de interoperabilidade e *Web* [Oliv97] [Picc98], sistemas configurados [MPLB98], *frameworks* [MPLB98], [Ucho99], [BL00] e desenvolvidos os projetos ECOHOOD e ECOBASE.

1.4.1 - Sistema HEROS

O Sistema HEROS (HEteRogeneous Object System), um SGBDH orientado a objetos [DPSM93], foi desenvolvido nos anos 90 no DI/PUC-Rio. Na última versão, a arquitetura do HEROS foi definida usando a técnica de *frameworks*, com o objetivo de proporcionar extensibilidade para a criação e utilização de novos SGBDHs [Ucho99]. O *framework* HEROS ($HEROS^{fw}$) possibilita a instanciação de SGBDHs para permitir que um conjunto de sistemas de bancos de dados heterogêneos sejam integrados em uma federação, para que

consultas e atualizações sejam realizadas de forma transparente à localização dos dados, aos caminhos de acesso e a qualquer heterogeneidade existente.

A partir do HEROS^{fw} foram instanciadas outras federações, no caso SGBDHs, como as aplicações desenvolvidas em [Silv99] e [Siqu99]. Muito embora a estrutura do HEROS^{fw} atenda a sua finalidade de integrar sistemas de banco de dados, percebemos que em algumas aplicações de integração nem sempre existe a necessidade de se ter todos as suas funcionalidades presentes, isto é, em algumas aplicações não foi necessário utilizar o mecanismo de regras, a gerência de transação ou controle de concorrência. Além disso, temos observado que a grande demanda por sistemas integradores se refere apenas a consulta, sem se ter a necessidade de efetuar alterações nas fontes de dados integradas. A característica do HEROS^{fw} de ter todos as suas funcionalidades presentes em qualquer instanciação e o grande relacionamento entre os seus *frameworks*, aumenta a sua complexidade e a dificuldade de customização, integração e manutenção. Também é primordial, atualmente, a necessidade de se permitir a integração de dados de outras fontes de dados não baseadas em banco de dados, principalmente da *Web*. Na seção 2.10 é apresentado com mais detalhes o sistema HEROS.

1.4.2 - O Projeto ECOHOOD

A partir das experiências adquiridas com o desenvolvimento do projeto HEROS, passamos a pesquisar novas formas para descrever arquiteturas para SGBDs com ênfase em sistemas configuráveis e extensíveis e no uso de *frameworks* e de *Design Patterns* [GHJV95], através do projeto ECOHOOD. O projeto ECOHOOD (*Environment of Configurable Heterogeneous Object-Oriented Databases*) [MPLB98] propõe uma abordagem nova e mais geral para se construir SGBDs. Seu objetivo foi a especificação de um ambiente para permitir a configuração de SGBDs específicos para diferentes domínios de aplicação.

Nesse cenário, um SGBD configurado é um conjunto de serviços interrelacionados requeridos para a gerência de dados em um domínio específico. A adaptação seria possível para todos os serviços básicos, tais

como: processador de consulta, gerência de transação, gerência de regras e gerência de armazenamento, permitindo também a introdução de novos tipos de serviços. Para obter essa flexibilidade de configuração, a abordagem proposta usa os conceitos de sistemas configuráveis e extensíveis, de geradores de sistemas de bancos de dados, de *frameworks* e de *Design Patterns*.

1.4.3 - O Projeto ECOBASE

O projeto ECOBASE (Tecnologias de Bancos de Dados e *Web* para Sistemas de Informações Ambientais) [Ecob99] é um projeto de pesquisa multi-institucional, envolvendo a comunidade de pesquisa de banco de dados do Rio de Janeiro (Brasil) e o INRIA (França), dentro do acordo de cooperação CNPq (Brasil) e INRIA. As instituições envolvidas são a PUC-Rio, a UNI-Rio (Universidade do Rio de Janeiro), a UFRJ (Universidade Federal do Rio de Janeiro) e o INRIA Rocquencourt – Groupe Rodin. O objetivo do projeto é compartilhar os resultados de pesquisa e os componentes derivados, bem como realizar experiências com várias aplicações, no que se refere ao acesso e integração de informações ambientais. A arquitetura comum a ser adotada é de três camadas, com uma camada cliente, uma camada *middleware* e uma camada de fonte de dados. A arquitetura deve ser baseada em componentes, onde cada grupo pode desenvolver seus próprios componentes, utilizar componentes já existentes e compartilhar componentes e integrá-los de várias maneiras para experimentação.

A camada cliente deve prover uma interface baseada em *Web*, integrada com navegadores padrões para acessar fontes de dados com uma interface transparente e uniforme. A camada *middleware* gerencia metadados e é capaz de traduzir consultas sobre fontes de dados distribuídas em consultas sobre fontes locais. A camada de fontes de dados provê *wrappers* para uniformizar fontes heterogêneas e permitir que dados sejam facilmente extraídos ou consultados via *Internet*. *Wrappers* podem ser construídos para homogeneizar a aparência de qualquer tipo de fonte de dados, inclusive sistemas legados, podendo variar em complexidade dependendo das capacidades das fontes de dados. É também interesse, a pesquisa de geração

de *middleware* como SGBDs configurados, especializados no domínio de Sistemas de Informações Ambientais e usando o conceito de *framework* da engenharia de *software*, principalmente para a definição do catálogo e do processador de consultas.

Conforme visto, integração de dados heterogêneos e distribuídos é um problema bem atual, sendo necessários a pesquisa, o desenvolvimento e a proposta de novas soluções. A disponibilidade de novas tecnologias nas áreas afins, citadas anteriormente, e as experiências adquiridas nos trabalhos desenvolvidos pelo nosso grupo de pesquisa nestes últimos anos, impulsionaram o avanço da pesquisa e permitiram a realização do presente trabalho. Nossa proposta é baseada em linhas de pesquisa atuais e em problemas ainda em aberto na comunidade científica.

O trabalho aqui apresentado é, ainda, uma contribuição para o projeto ECOBASE, no que se refere a proposta, especificação e definição de uma arquitetura comum para o projeto, de acordo com os seus propósitos iniciais. Um artigo sobre o CoDIMS [BP01] foi submetido, aceito e apresentado no *WIIW2001* (*International Workshop on Information Integration on the Web - Technologies and Applications*), patrocinado pelo projeto ECOBASE.

1.5 – Visão Geral da Solução Proposta

Este trabalho propõe a especificação de um ambiente flexível e configurável, denominado CoDIMS, que possibilita gerar sistemas *middleware* configurados para a integração de dados heterogêneos e distribuídos. O modelo de configuração proposto pretende viabilizar a construção de sistemas integradores “enxutos”, de forma rápida, mais flexível e de menor custo. Para esse fim é utilizada a técnica de componentes e de *frameworks*.

Diversas são as definições de *middleware* encontradas na literatura. A mais abrangente o define como “um *software* que permite aplicações interoperarem através da rede, apesar de diferenças nos protocolos de comunicação, arquiteturas de sistemas, sistemas operacionais, bancos de dados e outros serviços disponíveis” [Ryme96]. Em analogia a uma arquitetura

cliente-servidor de três camadas [MST97], *middleware* é a camada intermediária entre a camada de dados e a de apresentação (figura 1.1).

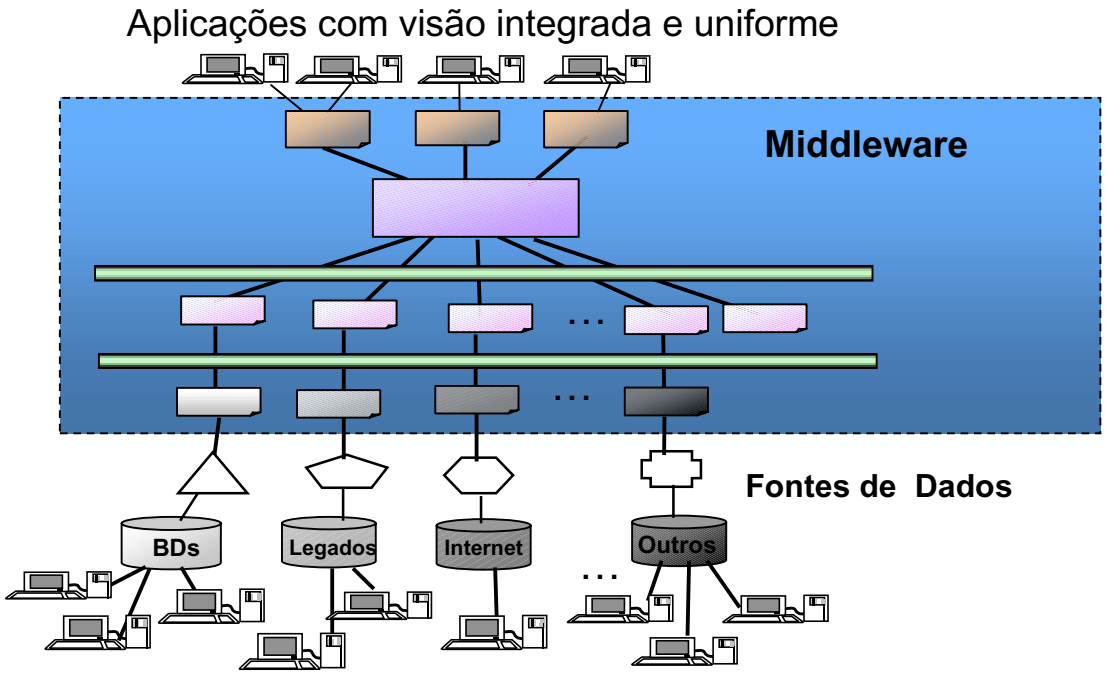

Figura 1.1 – Arquitetura de três camadas e o *middleware*

A utilização de *middleware* não significa ligar fisicamente clientes e servidores, o que é o papel dos protocolos de conectividade como *TCP/IP* [Stev94], mas sim, ligar os elementos lógicos das aplicações de modo que elas possam interoperar, protegendo-as das diversidades do ambiente. Em resumo, um sistema *middleware* de integração visa fornecer uma visão uniforme e integrada dos recursos (dados e ambientes) distribuídos e heterogêneos.

As características fundamentais do CoDIMS são flexibilidade e configuração. Flexibilidade, no sentido que permite: integrar diferentes fontes de dados; utilizar diferentes técnicas de comunicação tanto entre os componentes internos quanto com as fontes de dados; permitir a utilização de diferentes modelos de dados internamente. Configuração, de forma que o *middleware* seja configurado apenas com os componentes necessários para uma aplicação específica e que seja instanciado de acordo com os requisitos da aplicação e das características das fontes de dados que se deseja integrar. Nossa proposta utiliza tecnologias abertas, bem conhecidas e propõe

integração sem replicação dos dados. As figuras 1.2 e 1.3 apresentam um esquema geral do CoDIMS e seus componentes.

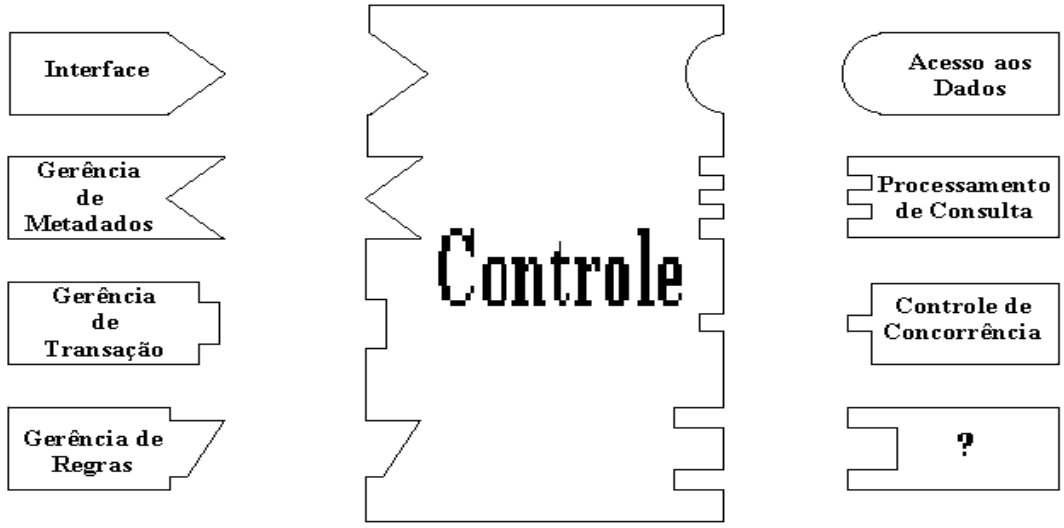

Figura 1.2 – CoDIMS: um ambiente configurável baseado em componentes

Cada componente do CoDIMS, denominado DIMS, é baseado nas funcionalidades dos serviços típicos de gerência de dados e são definidos como *frameworks*.

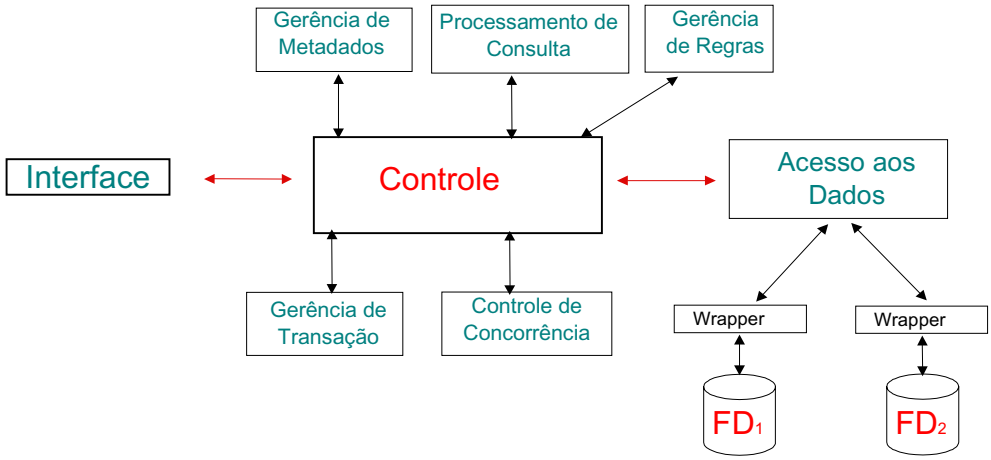

Figura 1.3 – O CoDIMS e seus componentes

Cada componente se comunica com os outros componentes sempre através do Controle, o qual se assemelha a um *BUS*, como utilizado em

arquiteturas de *hardware*, ou a um *ORB* (*Object Request Broker*) [OMG00], como em *CORBA*. Essa técnica possibilita não se utilizar o relacionamento direto entre os componentes, permitindo uma maior facilidade e flexibilidade na escolha, na substituição, na modificação e na integração dos componentes do sistema configurado.

Enquanto em *CORBA* a função do *ORB* é simplesmente repassar as mensagens recebidas ao componente destino, o Controle, além dessa função, é responsável pela coordenação do sistema, pela consistência da configuração, pelo escalonamento e pela gerência dos serviços a serem executados, de acordo com a requisição recebida. A invocação dos serviços para execução é expressa através de um *workflow* pré-definido. A utilização de um *workflow* proporciona flexibilidade para a adaptação de novas linguagens de consulta e, principalmente, o desacoplamento dos serviços. O Controle apresenta ainda uma *API* (*Application Programming Interface*) [Lima97] para a comunicação com a aplicação cliente.

Além do Controle, os componentes Gerência de Metadados, Processamento de Consulta e Acesso aos Dados têm de estar presentes em toda e qualquer configuração:

- **Acesso aos Dados**: responsável pela comunicação com as fontes de dados a serem integradas.
- **Processamento de Consulta**: responsável por analisar a consulta global recebida, derivar as sub-consultas que devem ser submetidas a cada fonte de dados, produzir um plano global otimizado para a execução da consulta e proceder sua execução.
- **Gerência de Metadados**: Responsável pela gerência dos metadados do sistema, incluindo os das fontes de dados necessários para o processo de integração.

Os serviços/componentes restantes devem ser utilizados de acordo com a necessidade da aplicação. Por exemplo, no caso do *middleware* a ser configurado executar funções de um SGBDH, será necessário apresentar os componentes de Gerência de Transação e Controle de Concorrência.

- **Gerência de Transação**: Responsável pela gerência e garantia das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) quando da atualização dos dados [ÖV99] [SKS99].
- **Controle de Concorrência**: Provê um mecanismo de controle quando da ocorrência de pelo menos um pedido de atualização sobre um mesmo dado entre transações diferentes, neste caso, concorrentes.

A incorporação do componente **Gerência de Regras** enriquece o sistema com um comportamento ativo, isto é, capacidade de reagir automaticamente à ocorrência de determinados eventos, tal como proposto nos sistemas de bancos de dados ativos [WC96].

A utilização de *frameworks* na especificação e desenvolvimento dos componentes permite aumentar a flexibilidade, devido a possibilidade customização, de acordo com os requisitos da aplicação. *Frameworks* possibilitam reuso de análise, de projeto e de código, aumenta a qualidade do *software*, reduz o esforço e o tempo de desenvolvimento de novas aplicações e o de manutenção.

1.6 – Contribuições Esperadas

As contribuições esperadas com este trabalho são:

- A especificação de um ambiente configurável e flexível para integração de dados heterogêneos e distribuídos;
- Um mecanismo de configuração que permita a seleção e a integração de um conjunto variável e adequado de serviços e de componentes;
- O uso de componentes baseados nos serviços tradicionais de sistemas de gerência de banco de dados, adaptados para o processo de integração de dados;
- Utilização da técnica de *frameworks* para especificação dos componentes, possibilitando maior flexibilidade, facilidade de customização para uma aplicação específica e reuso;

- Para nosso grupo de pesquisa, a especificação de um ambiente que permita o desenvolvimento em paralelo dos serviços/componentes, possibilitando sua composição futura através das interfaces definidas;
- Para o projeto ECOBASE, uma proposta de arquitetura e a possibilidade do desenvolvimento, implementação e experimentação de componentes, que podem ser compartilhados com outras instituições, bem como a possibilidade de uso de outros componentes já disponíveis.

1.7 – Organização da Tese

O restante deste trabalho está assim organizado:

No capítulo 2 são apresentados e discutidos os trabalhos mais importantes existentes na literatura sobre integração de dados heterogêneos e distribuídos, relacionados com a presente pesquisa.

No capítulo 3 são discutidos os problemas e propostas para o desenvolvimento de sistemas para integração de dados heterogêneos. Adicionalmente, são apresentados os conceitos de *frameworks*, componentes e de outros tópicos relacionados como: arquitetura de *software*, reuso, *design patterns* e sistemas configuráveis.

No capítulo 4 é apresentado o ambiente CoDIMS, descrevendo seus componentes e os mecanismos de configuração e de escalonamento das tarefas a serem executadas.

No capítulo 5 é apresentada a especificação do ambiente CoDIMS, através de uma extensão da *UML* para a descrição de *frameworks*.

No capítulo 6 é apresentado um exemplo de configuração e customização do ambiente CoDIMS, apresentando um estudo de caso e sua implementação.

Finalmente, o capítulo 7 apresenta uma síntese do trabalho realizado, suas principais contribuições e a indicação de alguns trabalhos futuros para a continuidade desta pesquisa.

Em seguida são listadas as referências bibliográficas utilizadas.

Capítulo 2 – Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos mais importantes existentes na literatura sobre integração de dados heterogêneos e distribuídos, com o propósito de investigar os serviços básicos existentes em cada uma das arquiteturas e verificar as características comuns entre elas, para permitir um maior embasamento para a definição dos serviços e componentes do ambiente CoDIMS.

2.1 – Introdução

Na literatura são encontradas diversas propostas e sistemas voltados para o problema de integração de dados heterogêneos e distribuídos. A grande maioria destes sistemas oferece apenas a possibilidade de consulta (leitura), sem se preocupar com a necessidade de também se efetuar alterações nas fontes de dados integradas. São utilizadas diversas abordagens nas propostas existentes para integração de informações:

- Linguagens de consultas;
- Sistemas de recuperação de informação;
- Sistemas de mediadores;
- Sistemas baseados nos serviços de gerência de dados existentes nos SGBDs.

Segundo [ÖV99], a pesquisa em integração de informações, principalmente na *Web*, é muito desafiadora e apenas está em seu início. Os projetos propostos têm aumentado o entendimento de integração de informações de fontes de dados distribuídas e heterogêneas, mas ainda é necessário facilitar o desenvolvimento de sistemas *middleware* e de *wrappers* em vários domínios de aplicação.

A seguir são apresentadas as arquiteturas de alguns sistemas: os mais referenciados na literatura e mais diretamente relacionados com a nossa proposta. Além destes, outros foram analisados, mas não serão aqui apresentados devido às semelhanças com outros sistemas ou por utilizarem outras abordagens. [BBH+00] e [ÖV99] apresentam um resumo de outros sistemas de integração de dados. Nosso objetivo foi investigar os serviços básicos existentes em cada uma das arquiteturas e verificar as características comuns entre elas, com o propósito de nos permitir um embasamento maior para a definição dos serviços e componentes do CoDIMS.

2.2 – Strudel

Strudel [BBH+00] [FFK+97] [FFL+00] [ÖV99] é um projeto de integração de informação da *AT&T Labs Research*, com o objetivo de construir e gerenciar *Web sites*. Strudel separa o conteúdo das fontes de dados da visão lógica das informações integradas disponíveis no *Web site*, e estas, da apresentação das informações em páginas *HTML*. A construção de um *Web site* envolve as etapas de: escolher as informações que estarão disponíveis, organizar as informações em páginas individuais ou em um grafo de páginas conectadas e especificar a visão das páginas *HTML*.

A integração de informações das múltiplas fontes é representada em um modelo de grafo, para o qual todos os dados das fontes de dados são uniformemente mapeados. O mecanismo de processamento de consulta é baseado principalmente em sistemas de recuperação de informação. As fontes de dados podem ser estruturadas, semi-estruturadas ou até mesmo *Web sites* já existentes.

Strudel possui uma linguagem de consulta declarativa chamada STRUQL, para especificar o conteúdo e a estrutura do *Web site*. Um esquema de mediação permite ao usuário acessar dados integrados de fontes heterogêneas de forma transparente (independente de sua localização ou de sua apresentação) e customizar ou reorganizar o *Web site* através da definição de diferentes visões sobre os dados.

A arquitetura do Strudel (Figura 2.1) contém:

- *Wrappers* que fazem a comunicação e o mapeamento da representação da fonte de dados para o modelo de grafo;
- Um mediador que integra as informações em um grafo de dados;
- Um processador de consultas que executa as consultas com base no grafo de dados;
- Um grafo do *site*, que são visões do grafo de dados;
- Um gerador *HTML* que materializa o grafo de *site* como um grafo navegável de páginas *HTML*.

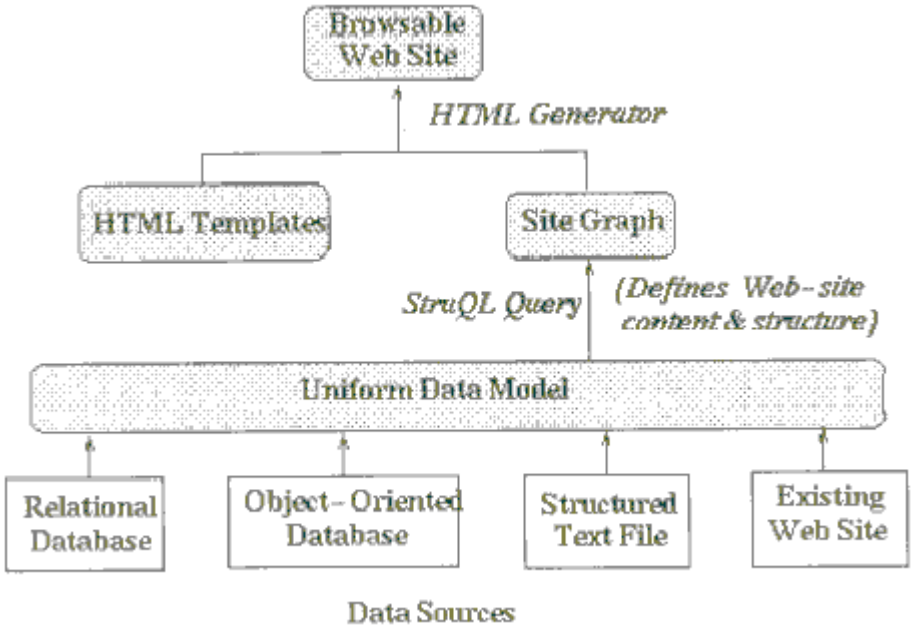

Figura 2.1 – Arquitetura do Strudel

2.3 – Araneus

Araneus [MAM+98][BBH+00] é um sistema de gerência de *Web-base* (WBMS), desenvolvido pela Universidade de Roma Tre (Itália), que propõe um novo tipo de repositório de dados projetado para gerenciar dados *Web*, no estilo de banco de dados. Um WBMS deve prover funcionalidades para gerenciar tanto bancos de dados quanto *web sites*. Uma *Web-base* é uma coleção de dados de natureza heterogênea, permitindo a criação, consulta e manutenção de *Web sites* distribuídos de forma transparente ao usuário e independente de

plataforma. Na verdade, uma *Web-base* é um *web site* com dados integrados a partir de outros *web sites*.

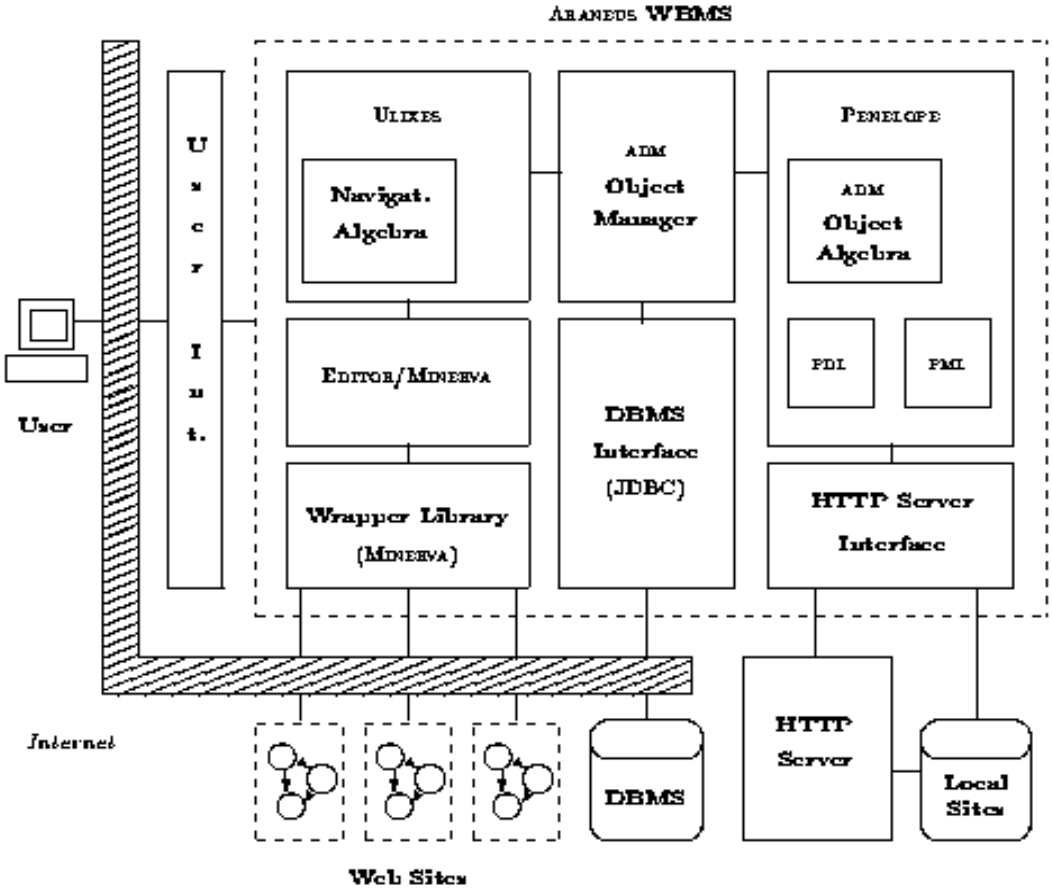

Figura 2.2 – Arquitetura do Araneus

A arquitetura do Araneus (Figura 2.2) é composta por:

- Ferramentas para escrever *wrappers* (Editor e Minerva);
- Uma biblioteca de *wrappers*;
- Um SGBD relacional para armazenar os dados;
- Um gerente de objetos que usa um modelo de dados orientado à páginas, denominado ADM. Os objetos (páginas) são armazenados no SGBD relacional sobre a forma de tabelas;
- Um módulo, denominado Penelope, para auxiliar no processo de definição e manutenção de um novo *site*;
- Um servidor de páginas *HTTP* para o gerenciamento de páginas;
- Um processador da linguagem Ulixes.

O Araneus acessa os dados existentes em SGBDs (relacional ou relacional-objeto) via *SQL*/*JDBC*. Os dados de *web sites* são acessados via uma álgebra navegacional através da linguagem Ulixes. Todos estes dados são materializados no SGBD relacional, mas existe a opção de não materialização. A Interface do usuário é escrita em *HTML*, o que permite aos usuários finais e aos administradores o acesso ao sistema de qualquer ponto da rede.

2.4 – Le Select

Le Select [LeSe99] é um *middleware* desenvolvido no INRIA (França) através da abordagem de mediadores, cujo principal objetivo é disponibilizar dados e programas, além de permitir a integração de fontes de dados heterogêneas.

O Le Select possui uma arquitetura distribuída onde os servidores se comunicam segundo o modelo *peer-to-peer* (Figura 2.3) e a integração de dados é realizada através do modelo relacional. Um *site publicador* disponibiliza os seus dados e/ou programas para qualquer cliente Le Select via *Web*.

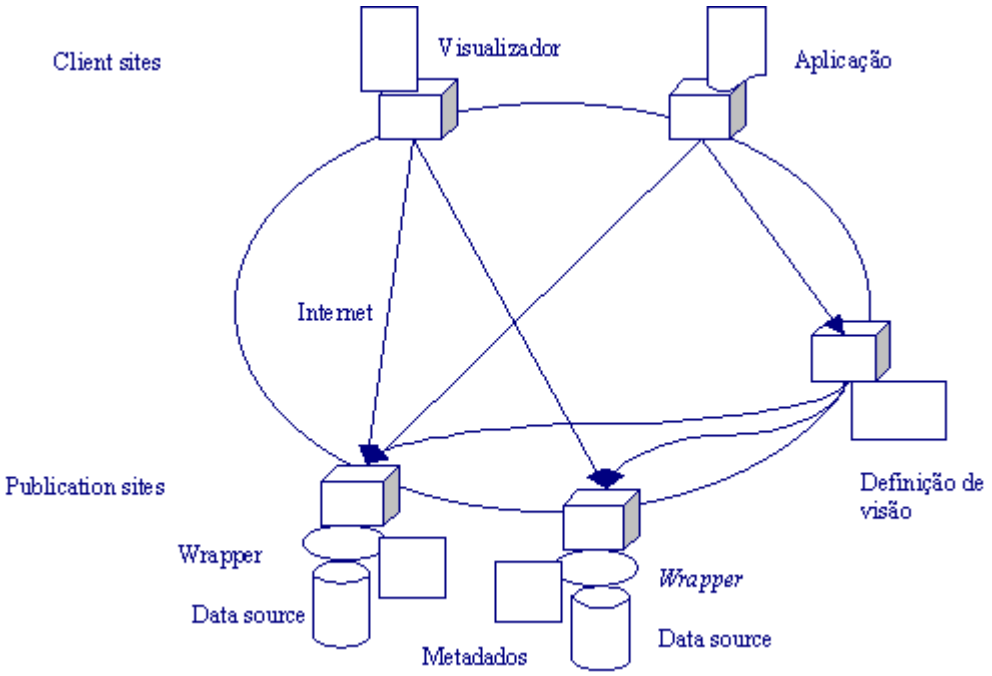

Figura 2.3 – Arquitetura do Le Select

Um *site* publicador é criado por um usuário Le Select que, além de ser o responsável pela instalação do servidor Le Select no seu *site*, deve também definir os *wrappers* (via *XML*) que, além de converter as suas fontes de dados em tabelas relacionais, também registram onde os dados estão armazenados e como devem ser acessados. O sistema já disponibiliza *wrappers* para textos, tabelas e qualquer fonte via *JDBC*. Os dados/programas publicados são convertidos em tabelas pelos *wrappers* no instante em que as consultas são geradas pelo cliente. Portanto, essas tabelas não são materializadas e não há necessidade de se usar um SGBD específico para gerenciar os dados. Os metadados são escritos em *XML*.

Os clientes podem acessar os dados através de consultas *SQL* e podem disparar a execução de programas, de forma assíncrona, nos *sites* publicadores de programas que geram sessões persistentes de execução. Existe ainda a possibilidade do acesso aos dados via *FTP* ou através do protocolo *HTTP*. Para oferecer a possibilidade de integração de dados, um mecanismo de visão é utilizado. O uso de visão permite a declaração das transformações a serem aplicadas aos dados ou a integração de dados de diferentes fontes. Uma vez definida uma visão, a mesma pode ser usada como se fosse uma tabela contendo o resultado de uma consulta. Le Select foi implementado através de classes *Java* e procura tirar proveito de padrões abertos tais como *XML, JDBC e CORBA*.

2.5 – DISCO

DISCO [TAB+97] [TRV98] [ÖV99] (*Distributed Information Search Component*) é um projeto de integração de dados do INRIA Rocquencourt (França), baseado em mediadores. As fontes de dados podem ser: bancos de dados; arquivos; servidores de dados dedicados como: um servidor de multimídia ou páginas *HTML*. Os dados podem ser estruturados, semi-estruturados ou não estruturados. O modelo de dados utilizado é orientado a objeto. As aplicações alvo do DISCO são as da *Internet* e *Intranet* que, tipicamente, requerem integração de grande número de fonte de dados.

Sua arquitetura (Figura 2.4) é baseada em arquiteturas de três camadas, estendida com diversas novas características. Os mediadores (M) e *wrappers* (W) operam independentemente: um mediador acessa um *wrapper* através da descrição *URL* (*Uniform Resource Locators*). Esta característica permite, facilmente, compartilhar *wrappers* entre múltiplos mediadores. Cada mediador exporta sua capacidade de mediação usando uma descrição das operações que o *wrapper* suporta. Os mediadores se adaptam automaticamente para as capacidades dos *wrappers* usando uma distinção entre o plano de execução inicial (que não considera a capacidade do *wrapper*) e o plano de execução final (que adiciona as capacidades do *wrapper*). Além disso, os *wrappers*, opcionalmente, exportam estatísticas e equações de custo que descrevem o tamanho dos dados das respectivas fontes (D) e o custo de acessá-las. Os mediadores usam estas informações para realizar otimização de consultas.

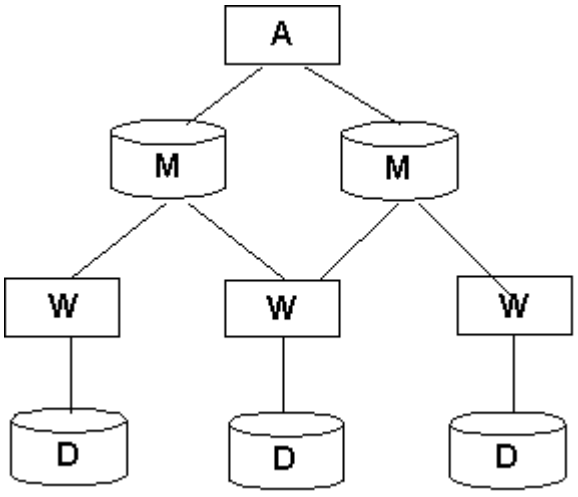

Figura 2.4 – Arquitetura do DISCO

Aplicações (A) acessam uma representação uniforme das fontes de dados através de uma linguagem *SQL-like*. As sub-consultas são expressas em uma linguagem algébrica que suporta operações relacionais. Durante a fase de processamento de consulta, o mediador transforma a consulta em um plano otimizado, contendo as sub-consultas e a composição dos resultados.

2.6 – TSIMMIS

TSIMMIS [MHI+97] [Tsim99] é um projeto da Universidade de Stanford, baseado em mediadores, cujo objetivo é desenvolver ferramentas que facilitem a rápida integração de fontes de informação heterogêneas que podem incluir dados estruturados ou não estruturados.

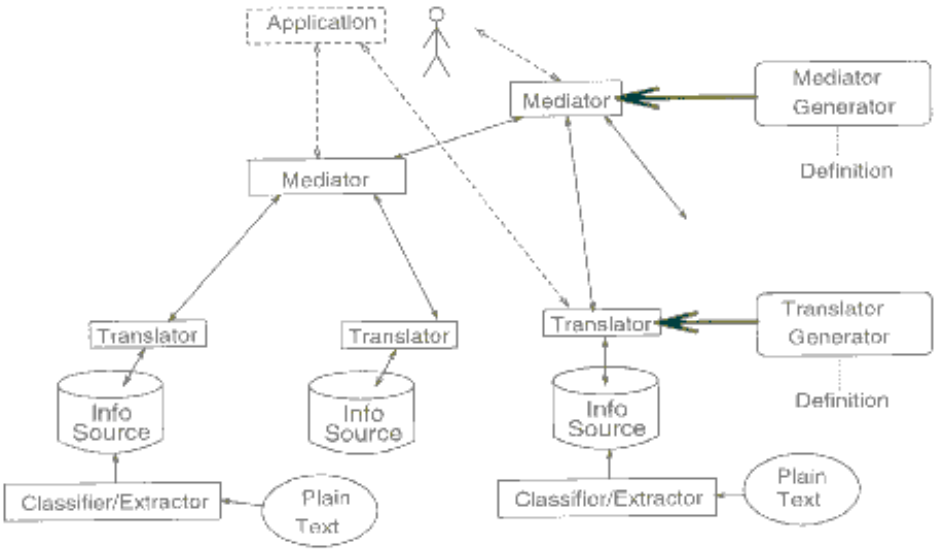

Figura 2.5 – Arquitetura do TSIMMIS

A arquitetura (figura 2.5) é composta dos seguintes componentes: tradutores (ou *wrappers*), mediadores e interface com o usuário. Os tradutores convertem os dados para um modelo comum denominado *OEM (Object Exchange Model*). Os mediadores apresentam conhecimento necessário para processar um tipo de informação específico. Considerando que a implementação de um mediador pode ser complicada e é um processo consumidor de tempo, um dos objetivos do projeto é gerar mediadores automaticamente ou semi-automaticamente.

A linguagem de consulta utilizada é a *OEM-QL*, uma adaptação da *OQL*. Não existe um esquema global. O processo de integração requer, muitas vezes, que o próprio usuário final o faça manualmente. O implementador do *wrapper* especifica, através de uma linguagem declarativa, as sub-consultas padrão aceitas pelo *wrapper*. Quando a aplicação submete uma consulta, é

verificado se existe uma sub-consulta que a atende. Quando o *wrapper* recebe o resultado, um mecanismo de filtragem é usado para produzir o resultado final.

2.7 – MIRO-Web

O MIRO-Web [FGL+98] é um projeto do *GMD (German National Research Center)* e de outras instituições colaboradoras, que tem por objetivo o desenvolvimento de componentes *middleware* para fornecer acesso integrado à várias fontes de dados, através de *browsers Web* padrão, usando o protocolo *Intranet*/*Internet*.

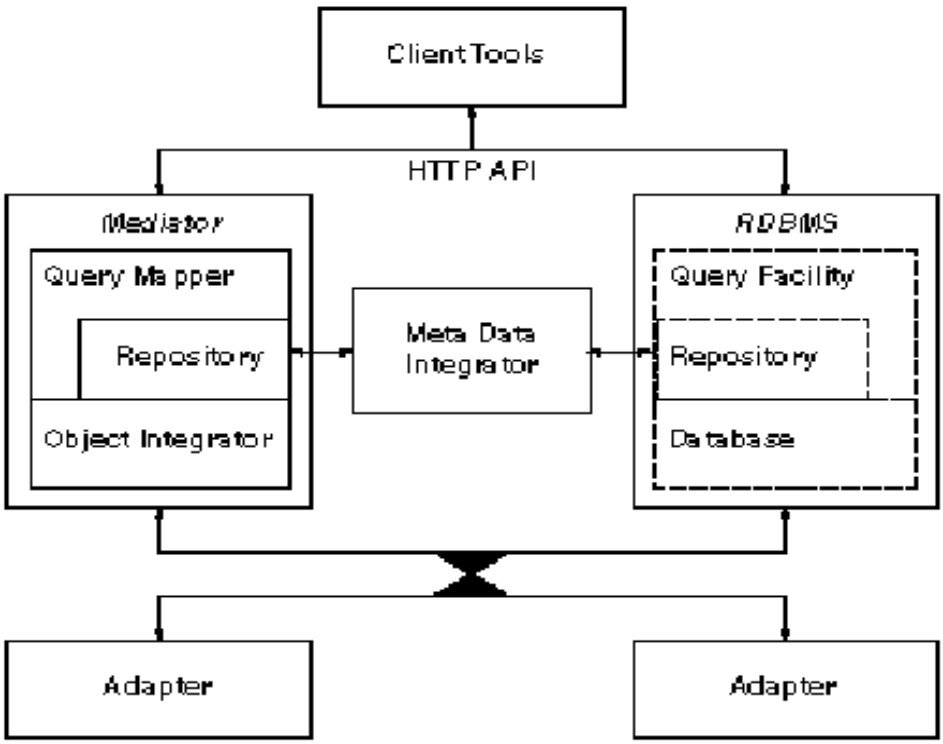

Figura 2.6 – Arquitetura do MIRO-Web

Os sistemas *Web* e os legados possuem novos requisitos, tendo um maior grau de heterogeneidade, principalmente devido a não estruturação dos dados. Para atender a estes requisitos, o sistema possui três componentes principais em sua arquitetura (Figura 2.6):

- Um *toolkit* para a construção de *wrappers*;
- Um integrador de metadados para extrair e integrar os vários esquemas;

- Um mediador para processar consultas do tipo relacional-objeto;
- Um *data warehouse*, baseado em um SGBD Relacional-Objeto (RDBMS).

O MIRO-Web aplica os conceitos e os componentes desenvolvidos no projeto anterior, denominado IRO-DB [GFF97], um SGBDH de acoplamento forte para a integração de bancos de dados relacionais e orientados a objetos, e os do sistema DISCO descrito anteriormente.

A arquitetura é organizada em três camadas: uma camada de adaptadores (*wrappers*) para permitir um acesso uniforme às fontes de dados individuais; uma camada de comunicação e outra de integração das visões das diversas fontes. A evolução teve por objetivo atender a necessidade de desenvolver aplicações no ambiente *Web*, permitindo o acesso a múltiplas fontes de dados de um modo integrado.

2.8 – Garlic

Garlic [CHS+97] [HMN+99] [RÖH99] é um *middleware* baseado em serviços de banco de dados do grupo de pesquisa da IBM – Almaden. Seu objetivo é construir um sistema de informação multimídia de larga escala, capaz de integrar dados que residem em diferentes sistemas de banco de dados, bem como em uma variedade de servidores não baseados em banco de dados.

Sua motivação se baseia no fato de que a maior parte dos dados existentes no mundo não está armazenada em bancos de dados e, por isso, existem muitos sistemas especializados emergindo para armazenar e pesquisar um tipo particular de dados. Os dados são integrados através de um esquema unificado expresso em um modelo orientado a objetos, que pode ser consultado usando um dialeto da *SQL*.

Os principais componentes de sua arquitetura (Figura 2.7) são: os *wrappers*, um integrador de esquemas, um repositório de metadados e um processador de consulta que usa técnicas de otimização de sistemas de gerência de bancos de dados.

Uma ferramenta, denominada Clio, tem como finalidade criar mapeamentos entre dois modelos de dados semi-automaticamente, facilitando a construção do esquema integrador. Além disso, existe disponível uma biblioteca de *wrappers*.

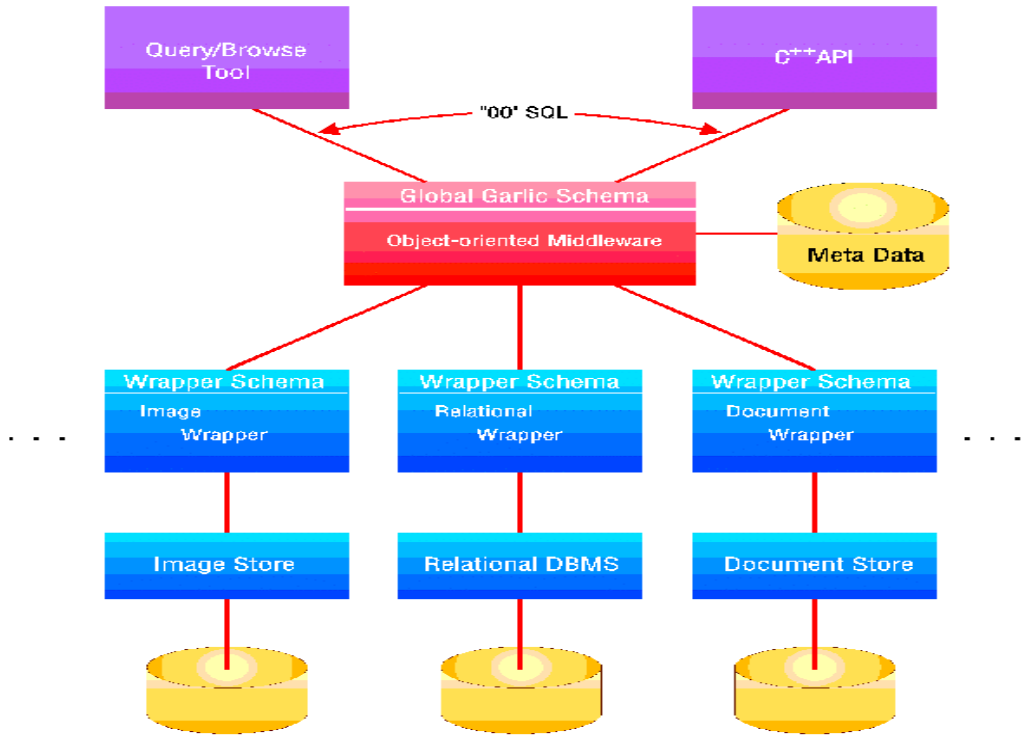

Figura 2.7 – Arquitetura do GARLIC

2.9 – MOCHA

MOCHA [MR00][Moch00] (*Middleware Based on a Code Shipping Architecture*) é um *middleware* proposto pela Universidade de Maryland, baseado em serviços de banco de dados. Foi projetado para integrar centenas de fontes de dados distribuídas sobre a *Web*, tendo sido construído com a idéia de que um *middleware* para um ambiente distribuído de larga escala deve ser auto-extensível. Esta extensibilidade permite o envio de classes *Java* para os *sites* remotos de um modo automático, de acordo com as características das fontes de dados. As classes *Java* são necessárias para executar uma sub-consulta no próprio *site*, diminuindo o tráfego de dados entre o *site* e o sistema.

Os principais componentes de sua arquitetura (Figura 2.8) são:

- Um processador de consultas (*QPC – Query Processing Coordinator*);
- *Wrappers* (*DAP – Data Access Provider*);
- Um repositório de código (classes *Java*);
- Um catálogo para o armazenamento de metadados.

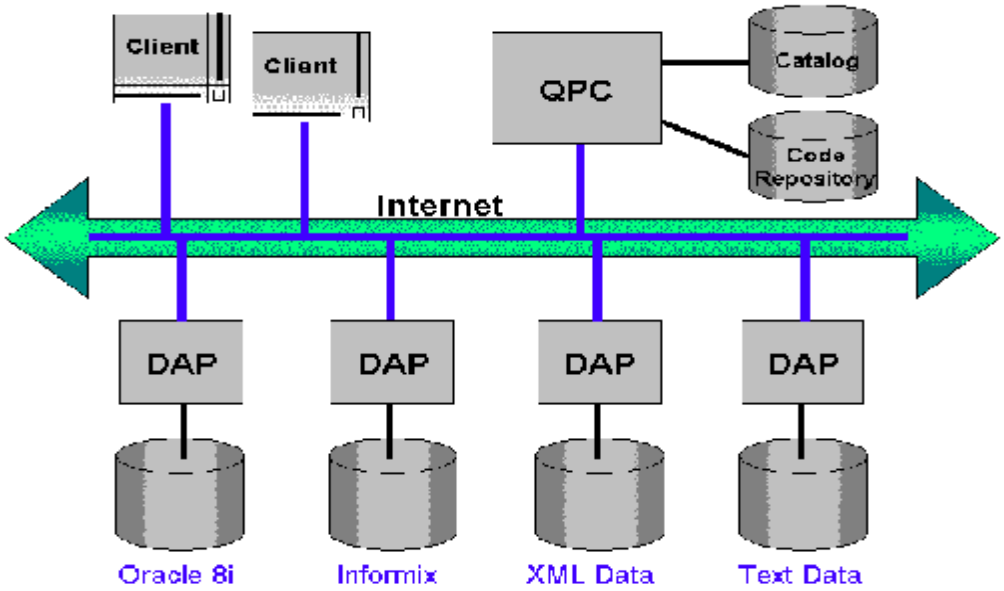

Figura 2.8 – Arquitetura do MOCHA

O projeto MOCHA é uma evolução do HERMES (*A Heterogeneous Reasoning and Mediator System*) [Moch00], que tinha como objetivo desenvolver uma linguagem para definir um mediador que expressasse a integração semântica de informação de diversas fontes de dados.

A motivação do MOCHA é que, os dados armazenados em diversos sites são baseados em tipos de dados complexos e que a *Web* tem se tornado, de fato, uma interface de usuário para aplicações em rede. Os usuários necessitam de uma solução que permita, facilmente, integrar os clientes baseados em *Web* a visualizar estes dados. Os dados podem ser acessados via *SQL*, *XML* ou através de uma interface procedural *HTTP*, *FTP* etc.

O plano gerado pelo otimizador indica as operações a serem avaliadas pelo QPC e aquelas a serem avaliadas pelos sites remotos. O plano também indica quais classes *Java* devem ser enviadas para cada *site*. Todos os planos

são codificados e enviados como documentos *XML*. Os metadados são especificados como *RDF* (*Resource Description Framework*), baseado em *XML*.

2.10 – O Sistema HEROS

O HEROS (*HEteRogeneous Object System*), um SGBDH orientado a objetos [DPSM93], foi desenvolvido nos anos 90 no DI/PUC-Rio, sendo classificado como acoplamento forte [SL90]. Na primeira versão do HEROS, o sistema de gerência de transações foi desenvolvido como um conjunto de módulos funcionais implementados como funções da linguagem de programação C [Silv94]. Para proporcionar maior flexibilidade e extensibilidade ao HEROS com relação à linguagem de programação, sistema operacional e localização dos sistemas de bancos de dados componentes, a segunda versão do HEROS passou a utilizar o padrão *CORBA* [UML96]. Na última versão, a arquitetura do HEROS foi definida usando a técnica de *frameworks*, com o objetivo de proporcionar extensibilidade para a criação e utilização de novos SGBDHs [Ucho99].

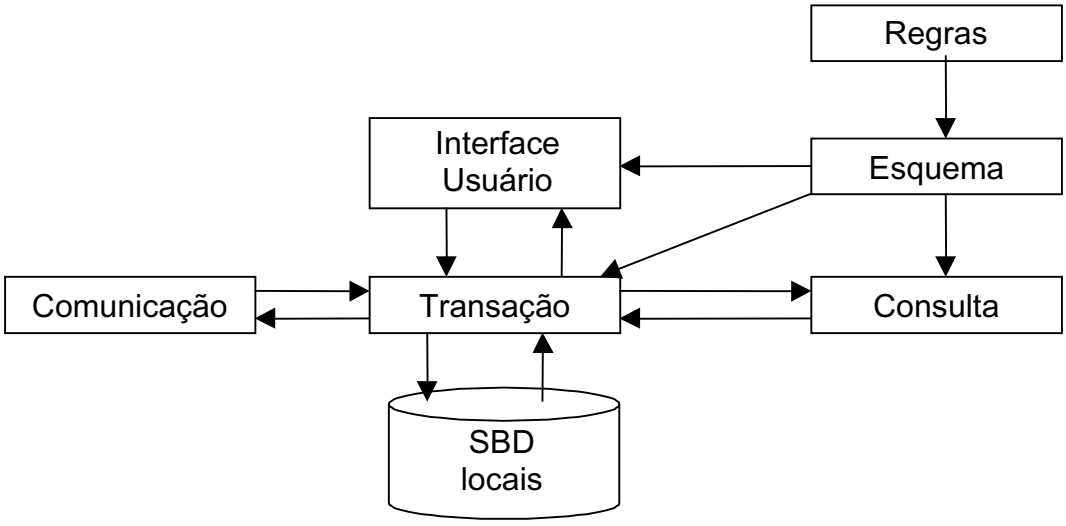

Figura 2.9 - O *framework* HEROS^{fw}

O *framework* HEROS (HEROS^{fw}) possibilita a instanciação de SGBDHs para permitir que um conjunto de sistemas de bancos de dados heterogêneos

sejam integrados em uma federação. Assim, consultas e atualizações podem ser realizadas de forma transparente à localização dos dados, aos caminhos de acesso e à qualquer heterogeneidade existente. O HEROS^{fw} (Figura 2.9) foi dividido funcionalmente em seis *frameworks*: Interface Usuário, Regras, Consulta, Esquema, Transação e Comunicação. O *framework* Interface Usuário é responsável pela comunicação do HEROS^{fw} com o usuário final, enquanto que o Esquema é responsável por gerenciar os esquemas do HEROS^{fw}. O *framework* Consulta é responsável por derivar, a partir da consulta global recebida e do esquema global, as sub-consultas que devem ser submetidas aos diversos sistemas de bancos de dados componentes, além de produzir um plano de execução alternativo otimizado. O *framework* Transação submete consultas ou solicitações de execução de procedimentos globais aos respectivos sistemas locais, implementando operações de sistemas de gerência de banco de dados necessárias para a composição dos resultados parciais e o armazenamento temporário dos resultados. O *framework* Comunicação é responsável pela comunicação física entre a federação e os sistemas componentes, detectando e reportando qualquer problema de conexão ao *framework* Transação. O *framework* Regras é responsável por prover o HEROS de um comportamento ativo, isto é, ter a capacidade de reagir automaticamente à ocorrência de determinados eventos.

Muito embora a estrutura do HEROS^{fw} atenda a sua finalidade de integrar sistemas de banco de dados, percebemos que, em algumas aplicações de integração, nem sempre existe a necessidade de todas as suas funcionalidades estarem presentes, isto é, em algumas aplicações, como em [Silv99] e [Siqu99], não foi necessário utilizar o mecanismo de regras, a gerência de transação ou controle de concorrência. A característica do HEROS^{fw} de apresentar em qualquer instanciação todas as suas funcionalidades, mesmo sendo desnecessárias, e o grande relacionamento entre os seus *frameworks*, aumenta a sua complexidade e a dificuldade de customização, integração e manutenção, como nos sistemas *heavy weight*.

Além disso, temos observado que a grande demanda por sistemas integradores se refere apenas a consulta, sem a necessidade de se efetuar

alterações nas fontes de dados integradas. Também é primordial, atualmente, a necessidade de permitir a integração de dados de outras fontes de dados não baseadas em banco de dados, principalmente da *Web*. Assim, devido a estas questões, a utilização do HEROS em vários tipos de aplicações torna-se inviável.

2.11 – Quadro-Resumo dos Sistemas

A tabela 2.1 apresenta um resumo das principais características dos sistemas descritos anteriormente, principalmente no que se refere ao modelo de dados utilizado, linguagem de consulta e a abordagem utilizada: mediador ou SGBDH (estas abordagens são descritas no capítulo 3).

Sistema	Abordagem	Modelo de Dados Integrador	Linguagem de Consulta	Processamento de Consulta
Strudel (AT & T Labs)	Mediador	Grafo de páginas HTML	STRUQL	Baseado em sistemas de recuperação de informação
Araneus (Universidade de Roma Tre)	Mediador	Relacional	Interface HTML	* Não está claro nas referências
Le Select (INRIA)	Mediador	Relacional	SQL	Tradicional
DISCO (INRIA)	Mediador	Orientado a Objeto	OQL	Tradicional
TSIMMIS (Universidade de Stanford)	Mediador	Object Exchange Model (OEM)	OEM-QL	Conjunto padrão de sub-consultas
MIRO-Web (GMD)	Mediador	Relacional-Objeto	SQL	* Não está claro nas referências
Garlic (IBM)	SGBDH	Orientado a Objeto	OQL	Tradicional
MOCHA (Universidade de Maryland)	Middleware	Relacional-Objeto	SQL	Tradicional
HEROS (PUC-Rio)	SGBDH	Orientado a Objeto	OQL	Tradicional

Tabela 2.1 – Quadro-Resumo de Alguns Sistemas de Integração

Os sistemas procuram ser flexíveis no desenvolvimento e uso de *wrappers*, para que consigam realizar o acesso a diferentes tipos e modelos de

dados. Mas a arquitetura de cada um deles se mantém a mesma, permanecendo sempre fixa para qualquer aplicação. Portanto, as características, arquiteturais e funcionais, são partes integrantes dos sistemas, não permitindo alterações para diferentes usos e aplicações. O que ocorre é que, de acordo com as fontes de dados que se queira integrar e dos requisitos de uma certa aplicação, pode ser que as características de um sistema específico não sejam as mais indicadas.

Por exemplo, no estudo de caso apresentado no capítulo 6, onde se deseja integrar informações de duas fontes de dados, uma base de dados *XML* e um banco de dados relacional, o modelo de dados integrador mais compatível seria o relacional ou mesmo *XML*. Aqueles sistemas com modelos de dados orientados a objetos produziriam um custo adicional para a conversão entre os modelos, podendo levar a problemas quando da integração semântica dos dados.

A solução adotada, atualmente, é a utilização de um destes sistemas, mesmo sendo inadequado, ou a construção de um novo sistema que apresente apenas as características específicas desejadas para a aplicação.

Estes problemas, limitações e desafios nos motivaram para a continuação da pesquisa em integração de dados, a qual resultou em nossa atual proposta de se poder construir sistemas *middleware* de integração configurados, flexíveis e enxutos.

2.12 – Síntese do Capítulo

Neste capítulo apresentamos um resumo de alguns sistemas de integração de dados mais referenciados na literatura. No final do capítulo, um quadro-resumo faz uma comparação das principais características dos sistemas abordados. O objetivo deste capítulo foi investigar as arquiteturas e as funcionalidades dos sistemas de integração apresentados como uma base para a definição dos componentes do CoDIMS.

No próximo capítulo são apresentados os conceitos e técnicas que foram utilizados para a especificação e desenvolvimento do ambiente CoDIMS.

Capítulo 3 – Conceitos: Integração de Dados Heterogêneos e *Frameworks*

Neste capítulo são discutidos os problemas e propostas para o desenvolvimento de sistemas para integração de dados heterogêneos. Adicionalmente são apresentados os conceitos de frameworks, componentes e de tópicos relacionados como: arquitetura de software, reuso, design patterns e sistemas configuráveis.

3.1 - Introdução

A globalização da economia, a internacionalização das empresas e a grande concorrência do mercado gerou a necessidade de se criar novos mecanismos para atender aos requisitos das diversas aplicações que devem ser realizadas em um tempo cada vez menor. A *internet* e a disponibilidade das redes de alta velocidade proporcionaram uma diminuição das distâncias no que se refere à localização das informações mas, para atender aos novos requisitos das aplicações, tornou-se necessário o acesso a informações armazenadas em diferentes fontes, que podem estar localizadas dentro de uma mesma empresa, em países distantes e em ambientes operacionais diferentes.

O advento do comércio eletrônico, seja ele empresa-empresa ou empresa-consumidor, ampliou as características dos ambientes computacionais atuais, que são formados por sistemas de *hardware* e *software* distribuídos, heterogêneos e autônomos. A heterogeneidade se manifesta através dos diferentes sistemas, fontes de dados, sistemas de comunicação ou *hardware* onde operam os sistemas de dados. A autonomia é um fator que aumenta a complexidade, pois os sistemas operam sobre um controle local e foram projetados independentemente uns dos outros. Distribuição, heterogeneidade e autonomia são representadas como dimensões ortogonais entre si [ÖV99] [Hass00].

Geralmente não existe tempo e justificativa para a substituição dos sistemas atualmente em operação. Novas funcionalidades devem ser integradas com outros sistemas e fontes de dados já existentes, devido às alterações nos requisitos dos negócios e avanços tecnológicos que, dada a velocidade com que ocorrem, não permitem a substituição dos sistemas existentes. Essa situação gera uma necessidade crescente por tecnologia para suportar a cooperação dos serviços fornecidos por *hardware* e *software*, os mais diversos possíveis. Os requisitos para construir sistemas que combinem recursos e serviços heterogêneos podem ser encontrados desde baixo nível de interconectividade até um alto nível de interoperabilidade. Interconectividade simplesmente suporta comunicação de sistemas enquanto interoperabilidade permite, adicionalmente, que os sistemas cooperem entre si para a execução conjunta de tarefas.

Muitas aplicações necessitam de informações de diversas fontes onde os dados relacionados podem diferir na representação ou na tecnologia utilizada. Para a integração destes dados, conforme visto no capítulo anterior, os sistemas propostos apresentam uma arquitetura semelhante às definidas nos ambientes cliente/servidor de três camadas: uma camada que representa as fontes de dados a serem integradas, uma camada das aplicações cliente e uma camada intermediária com a função de integração dos dados, surgindo então o conceito de *middleware*.

Middleware é um componente de *software* que tem como finalidade interligar processos clientes a processos servidores, disponibilizando um conjunto de serviços que visam reduzir a complexidade do processo de desenvolvimento de uma aplicação. *Midlleware* pode ser visto como uma ferramenta que “salva os desenvolvedores de se envolver em detalhes de cada fonte de dados” [Lint97]. É uma camada que existe entre a aplicação e as complexidades da rede, do sistema operacional e dos servidores de recursos, incluindo os servidores de dados. Por esta definição, o termo *middleware* é utilizado para descrever diversos níveis de interoperabilidade, desde os níveis de abstração mais baixos, como em aplicações mais próximas ao sistema

operacional e do sistema de rede, até os sistemas complexos como nos casos dos sistemas de integração de dados com maior nível de abstração.

3.1.1 – Padrões de Interoperabilidade

Uma das tentativas atuais nos produtos *middleware* de níveis mais baixos é a utilização de padrões, sejam os de mercado, oferecidos por fornecedores, ou os propostos por comitês como o *Object Management Group* (*OMG*) [Catt94] [OMG00] [ODMG00]. O *OMG* está desenvolvendo um conjunto de normas de padronização visando a integração de aplicações distribuídas através da tecnologia de objetos. A arquitetura proposta pelo *OMG* é a *OMA* (*Object Management Architecture*). No modelo *OMA* todo componente de *software* é representado como um objeto. Objetos se comunicam com outros objetos através do *ORB* (*Object Request Broker*), que é o elemento de comunicação chave. O *ORB* provê mecanismos onde objetos transparentemente fazem requisições e recebem respostas. O padrão proposto para *ORB* é o *CORBA* (*Common Object Request Broker Architecture*) que suporta uma Linguagem de Definição de Interface (*IDL*) geral, que pode ser mapeada para qualquer linguagem implementada. *CORBA* é a especificação de uma arquitetura e interface que permite que aplicações façam solicitações a objetos, de forma transparente, independente da linguagem, sistema operacional ou localização.

No passado, este tipo de *middleware* tinha característica proprietária, sem a facilidade e possibilidade de interoperabilidade. Atualmente, entretanto, está sendo comum a utilização de padrões, como *CORBA*, *DCOM* e *JavaBeans* para oferecer um alto grau de interoperabilidade entre componentes de *software*. *DCOM* serve como um padrão para ambientes *Windows* homogêneos. *CORBA* e *DCOM* procuram integrar componentes desenvolvidos em diferentes linguagens de programação, enquanto *JavaBeans* só contempla componentes desenvolvidos em *Java*. Um estudo sobre a composição destes diferentes padrões é apresentado em [Cerq00].

Cada um destes padrões define um subconjunto próprio de mecanismos para viabilizar a construção e a comunicação de seus componentes. Mesmo adotando abordagens diferentes, apresentam características semelhantes

como a dissociação entre a interface e a implementação de um componente. Portanto, para uma mesma interface podem existir diversas implementações (componentes), da mesma forma que um componente pode implementar diversas interfaces.

Um *middleware* deve possuir interfaces padronizadas (*APIs*) e protocolos de comunicação para possibilitar a distribuição, portabilidade e interoperabilidade entre as aplicações. As linguagens de definição de interface usadas por *CORBA* e *DCOM*, *OMG IDL* e *COM IDL* respectivamente, definem as interfaces de acordo com seus modelos de objetos. A descrição de uma interface é composta pelas assinaturas completas das operações (métodos) oferecidas, incluindo o nome, tipo do valor de retorno, tipos dos parâmetros e exceções [Krai00] [Cerq00]. *Java* oferece um mecanismo de suporte a chamada remota de métodos denominada de *RMI* (*Remote Method Invocation*), permitindo a passagem de objetos por valor em chamadas remotas de métodos.

3.1.2 – *Middleware* para Integração de Dados

Uma característica de um *middleware* para integração de dados é o usuário não enviar consultas diretamente para as fontes de dados. A razão disso, e seu principal objetivo, é liberar o usuário de ter que conhecer detalhes sobre todas as fontes de dados e ter que interagir com cada uma delas individualmente. Para o acesso às fontes de dados, os sistemas de integração utilizam os padrões de interoperabilidade descritos anteriormente, além de *wrappers*.

Wrapper é um tipo de *software* denominado "*glueware"* [Eske99b] [Schn99] usado para atuar junto com outros componentes de *software*. Um *wrapper* encapsula uma única fonte de dados para que ela possa ser utilizada de maneira mais conveniente, o que o distingue de outros tipos de *middleware*. Através do uso de *wrappers*, pode-se transformar uma fonte de dados em uma forma mais estruturada. *Wrappers* são assumidos como sendo simples, embora não exista uma clara demarcação entre o que seja um simples *wrapper* e um componente complexo de mais alto nível. Podem, ainda, ser

usados para apresentar uma interface simplificada, para adicionar funcionalidade ou para expor alguma das interfaces internas de uma fonte de dados.

Em [OBJS96] os *wrappers* são classificados em classes, de acordo com a funcionalidade. Dentre as classes de funcionalidade podem ser citadas:

- *Genérica*: que se refere às operações de acordo com o tipo da fonte de dados;
- *Fonte-específica*: que se refere às operações de acordo com o conteúdo da fonte de dados;
- *Estrutura de controle*: que se refere ao modo em que as requisições e as respostas são passadas;
- *Relatório de erros*: que pode ser retornado ao componente que o chamou.

Para a seleção de um *wrapper* particular devem ser analisadas quais fontes de dados serão tratadas pelo *wrapper*, se elas estão bem documentadas, quais são suas funções, como foram construídas, como é a arquitetura interna etc.

Em um nível de abstração acima dos *wrappers* estão os sistemas *middleware* para integração de dados heterogêneos e que usam *wrappers* para o acesso a cada fonte de dados. Diversos problemas devem ser tratados na construção de sistemas *middleware* para integração de dados:

- Especificação do esquema integrador;
- Linguagem de consulta a ser utilizada;
- Consistência dos dados nas fontes de dados;
- Diferentes capacidades de processamento de consultas e modelos de dados nas diversas fontes de dados;
- Mecanismos para proceder a gerência, otimização e execução de consultas globais e locais, quando for o caso.

O conjunto dos serviços disponibilizados pelo *middleware* determina a sua relevância. Historicamente, existem duas abordagens para sistemas de integração de dados heterogêneos: os mediadores e os SGBDHs.

Mediadores são um tipo de *middleware* que fornecem serviços em sistemas de informação, ligando uma ou mais fontes de dados a programas de aplicação [WG97] [Wied98]. A função de um mediador é fornecer informação integrada sem a necessidade de se integrar as fontes de dados [Wied98], ou seja, um mediador permite a interoperação de informação, somente selecionando resultados derivados das fontes de dados [Kim95] [Wied99]. Ele permite o acesso e a recuperação de dados relevantes de múltiplas fontes heterogêneas, a abstração e a transformação (homogeneização) dos dados recuperados, integração dos dados e o seu processamento para disponibilizar o resultado.

No uso de mediadores, um esforço maior é aplicado para processar os resultados da recuperação que para localizar e acessar os dados. Mediadores são construídos para domínios específicos de aplicação através da aquisição de conhecimento do domínio. A complexidade do mediador depende dos modelos de dados usados, das fontes de dados e das necessidades dos clientes.

Sendo o CoDIMS baseado em serviços de gerência de dados, neste trabalho será dada maior ênfase ao estudo de SGBDHs, que é o assunto da próxima seção.

3.2 - SGBDHs

Sistemas de Gerência de Bancos de Dados Heterogêneos (SGBDHs) [LMR90] [SL90] [TTC+90] [PBE95] [BE96] [SZ97] [CEH+97] foram propostos como uma das soluções para a integração de sistemas de bancos de dados existentes, sem a necessidade de alterá-los. Um SGBDH é uma camada de *software* com o objetivo de viabilizar o acesso a sistemas de bancos de dados autônomos, heterogêneos e distribuídos, que necessitam de flexibilidade para adaptar-se a diferentes ambientes e tipos dos sistemas componentes.

A criação de um SGBDH é uma tarefa complicada: além do fato dos sistemas de banco de dados componentes serem autônomos e terem sido projetados independentemente, também existem as heterogeneidades causadas por diferentes modelos de dados e linguagens de consulta, além de

outros fatores como gerência de transações, controle de concorrência, falhas, recuperação, segurança e integridade. De acordo com o tipo de integração existente entre os sistemas de banco de dados componentes de um SGBDH, estes podem ser classificados em sistemas com acoplamento forte ou com acoplamento fraco [SL90].

Os SGBDHs com acoplamento fraco são criados e mantidos pelo próprio usuário, não havendo nenhum controle imposto pelo sistema. O usuário, para acessar um dado, tem que compreender a semântica de cada esquema componente e resolver as heterogeneidades: todos os procedimentos estão sob sua responsabilidade.

Um SGBDH com acoplamento forte é composto por um conjunto de sistemas de bancos de dados componentes, heterogêneos, cooperativos mas autônomos, integrados de forma que as consultas e atualizações são executadas transparentemente para o usuário, independente da localização dos dados e da heterogeneidade. Neste caso, o esquema de cada sistema de banco de dados componente é traduzido para um modelo de dados comum e estes são integrados em um esquema global.

As arquiteturas de SGBDHs criam um SGBD *front-end* que realiza o mapeamento de consultas para qualquer dos diversos possíveis SGBDs *back-end*. Um SGBDH pode ser visto como um SGBD pai que coordena e acessa outros SGBDs [PBE95]. Os sistemas mais recentes usam arquiteturas de três camadas com um cliente mais enxuto que se conecta com um servidor que se conecta com outros servidores quando necessário.

Apesar de vários anos de pesquisa em SGBDHs, muitas questões ainda permanecem em aberto. Recentemente os SGBDHs têm sido estendidos e utilizados não somente para a integração de sistemas de bancos de dados, mas também para outros tipos de dados, passando então a serem denominados de uma forma mais geral como *middleware* baseados em banco de dados [MR00] [HMN+99]. Estes sistemas devem ter a facilidade de integrar dados armazenados em diversas fontes, em ambientes distintos tanto de *software* quanto de *hardware*, atender ao surgimento de novas formas de comunicação etc., sem a necessidade de reestruturar os sistemas atualmente

em operação. Faz-se, então, indispensável, projetar e implementar sistemas de integração que sejam flexíveis a ponto de poderem integrar fontes de dados heterogêneas, sem que se tenha que implementar sistemas chamados *heavy weight* [BT95].

A utilização do paradigma de banco de dados como *middleware*, removeu o código das aplicações e o delegou ao *middleware*, que passou a incorporar tanto as dificuldades da integração como o código especializado associado com a gerência de dados. A gerência de dados inclui os serviços característicos de SGBDs, como: esquema (metadados), otimização e processamento de consulta, acesso aos dados, gerência de *buffer*, linguagem de consulta de alto nível, gerência de transação, controle de concorrência, recuperação, integridade e consistência. Novas soluções e avanços na tecnologia de bancos de dados são necessários na solução do problema da integração, devendo a pesquisa em banco de dados contribuir mais nesta área [BBC+98] [SSU96] [SZ97].

Devido, principalmente, ao desenvolvimento de sistemas voltados para a *Web*, as abordagens de mediadores e de SGBDHs apresentadas anteriormente foram sendo estendidas buscando construir sistemas *middleware* mais flexíveis, o que torna difícil, atualmente, caracterizar um sistema como sendo baseado em uma ou outra abordagem. O termo que vem se tornando mais comum para as duas abordagens é *middleware*. Conforme visto, no caso de SGBDHs estendidos, estes estão sendo denominados de sistemas *middleware* para integração de dados heterogêneos baseados em banco de dados, que é o enfoque deste trabalho.

3.2.1 – Integração de Esquemas em SGBDHs

Para a integração de sistemas de bancos de dados através de um SGBDH, torna-se necessário a integração de seus esquemas. Para isso, todos os esquemas dos bancos de dados componentes devem estar descritos em um mesmo modelo de dados. O problema é que os esquemas dos diferentes bancos de dados componentes podem estar expressos em diferentes modelos.

Assim, torna-se necessário, inicialmente, transformar todos os esquemas para um mesmo modelo de dados, denominado de Modelo de Dados Comum (MDC). No caso de sistemas componentes não baseados em banco de dados, um *wrapper* deve simular a existência de um esquema (metadados) para permitir o processo de integração. O MDC é o modelo de dados usado pelo sistema integrador (SGBDH).

O esquema conceitual de cada banco de dados componente é denominado de *esquema local*. Este esquema, expresso no modelo de dados do sistema componente, deve ser transformado e descrito no MDC. O esquema derivado desta transformação é denominado de *esquema de exportação*. Um *esquema global* (ou federado), descrito no MDC, é então criado pela integração de múltiplos esquemas de exportação. Finalmente, são definidos os *esquemas externos* (visões) sobre o esquema global, para atender às necessidades de um grupo específico de usuários ou de aplicações, ou ainda por razões de controle de acesso.

A figura 3.1 apresenta uma arquitetura para integração de esquemas, mostrando os 4 níveis de esquema, utilizados no processo de integração por um SGBHD. Esta é a arquitetura utilizada pelo Sistema HEROS [USM94].

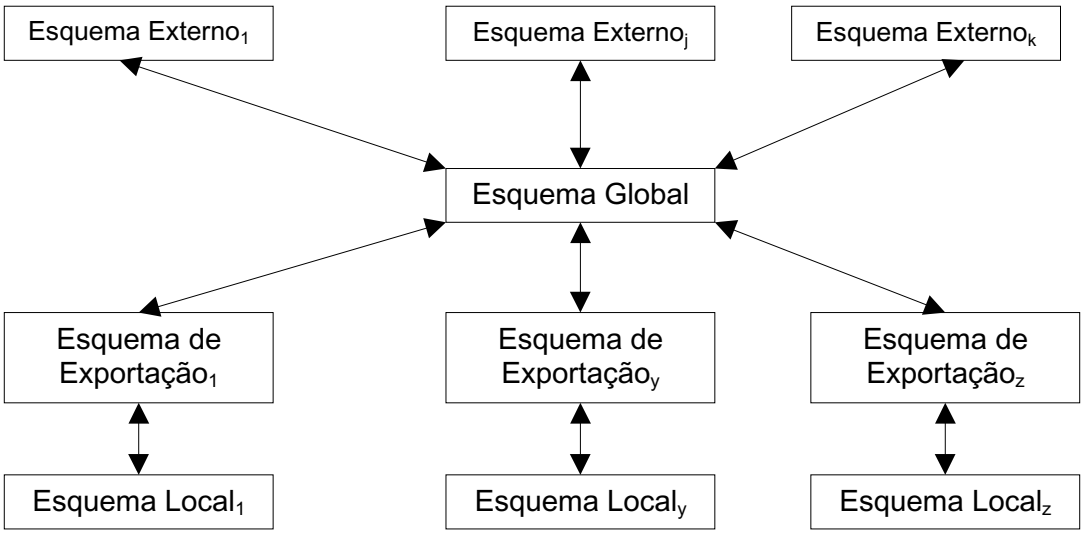

Figura 3.1 – Integração de esquemas

Diferentes tipos de SGBDHs são criados através das diferentes formas de integração dos esquemas de exportação. Um ambiente *não federado* assume a não integração explícita dos esquemas de exportação. Um exemplo de um ambiente não federado seria acessar todos os bancos de dados componentes através de uma linguagem de consulta. Um modelo *federado* [SL90] assume a integração dos esquemas de exportação para criar um esquema global. Sistemas de bancos de dados federados podem ser classificados com base na distribuição da integração. São *centralizados*, se suportam apenas um único esquema federado. No caso de *descentralizados*, cada componente constrói seu próprio esquema federado através da integração de seu esquema local com os esquemas de exportação de outros sistemas componentes.

A transformação de esquemas locais em um MDC é essencial tanto para os ambientes federados quanto para os não federados. O MDC deve ser rico o suficiente para capturar as semânticas expressas ou implícitas nos esquemas locais, e simples o suficiente para facilitar a criação de um esquema global no modelo federado e poder permitir consultas a diversos bancos de dados em um ambiente não federado. Muitos SGBDHs usam o modelo relacional como MDC [TTC+90]. Mais recentemente, foram propostos SGBDHs introduzindo o uso de modelo de dados orientado a objetos como MDC, como no caso do HEROS.

A transformação de esquema diminui os problemas que ocorrem devido ao uso de diferentes modelos de dados. Infelizmente, os mesmos conceitos podem estar representados em diferentes bancos de dados e, além disso, devido à heterogeneidade, estes conceitos podem estar representados de forma diferente. Assim, existem alguns tipos de conflitos que podem ocorrer entre os esquemas componentes e que independem do tipo de MDC que se está utilizando [ÖV99] [PBE95]:

1. ***Conflito de identidade***: ocorre quando o mesmo conceito é representado por diferentes objetos em diferentes bancos de dados componentes. Por

exemplo, quando dois exemplares de um mesmo livro têm diferentes identificadores em diferentes bibliotecas.

2. ***Conflito de esquema***: ocorre quando os esquemas componentes que representam o mesmo conceito não são idênticos:

   a) ***Conflito de nome***:
      - ***Homônimos***: ocorre quando o mesmo nome é usado por diferentes conceitos. Por exemplo, o termo “mídia” em uma biblioteca pode significar revistas e jornais e em outra apenas vídeo.

      - ***Sinônimos***: ocorre quando o mesmo conceito é descrito por diferentes nomes. Por exemplo, uma biblioteca usa o termo referências enquanto outra usa bibliografia.

   b) ***Conflito estrutural***: ocorre quando o mesmo conceito é representado por diferentes construtores do modelo de dados ou, os construtores usados têm diferentes estruturas ou diferentes comportamentos. Por exemplo, em uma biblioteca, o número de vezes que um livro foi emprestado é dado por um atributo do livro, enquanto que em outra biblioteca ele deve ser calculado por um método, quando necessário.

3. ***Conflito semântico***: ocorre quando o mesmo conceito é interpretado de forma diferente em bancos de dados distintos. Por exemplo, em uma biblioteca o termo “reserva” é usado para um livro que nunca circula, enquanto que em outra significa que alguém o está aguardando para empréstimo.

4. ***Conflito de dados***: ocorre quando os valores dos dados de mesmo conceito são diferentes em diferentes bancos de dados componentes. Por exemplo, o mesmo livro aparece tendo diferentes autores em diferentes bibliotecas.

As tarefas de transformação e integração de esquemas são fortemente influenciadas pelo modelo de dados usado para representar o esquema componente. Para executar a integração é crucial identificar, não somente o conjunto de conceitos comuns, mas também o conjunto dos conceitos diferentes nos diversos esquemas, que são mutuamente relacionados por alguma propriedade semântica. Estas propriedades são chamadas *propriedades interesquema* [PBE95].

Existem diversas propostas para integração conceitual de dados heterogêneos, que ainda é um problema em aberto e que não é o enfoque deste trabalho. Algumas destas propostas são descritas em [Srin97]. A arquitetura I3 (*Intelligent Integration of Information*) proposta pela *ARPA* (*Advanced Research Projects Agency*) especifica os serviços, ferramentas e tecnologias que são necessários para acessar informações de fontes de dados heterogêneos, incluindo os sistemas legados. Um sumário do projeto I3 está disponível em http://ic-www.arc.nasa.gov/ic/projects/xfr/misc/ARPA-I3-95.html.

3.3 – Informações na *Web*

A *WWW* (*World-Wide Web*), ou simplesmente *Web*, é hoje o meio mais utilizado para disponibilizar e acessar informações, seja através da *Internet* ou de *Intranets*. O problema é que, disponibilizar informação simplesmente como páginas estáticas escritas em *HTML* (*Hipertext Markup Language*), não é a melhor opção para grande parte dos dados que já residem em bancos de dados. Isto porque grande volume de dados tem de ser replicado, atualizações são complicadas e as facilidades de consultas são limitadas.

A *Web* é uma tecnologia nova e que, apenas recentemente, ganhou popularidade, tanto entre os pesquisadores e fornecedores como na comunidade em geral. A *Web* é caracterizada por um grande recurso de informação globalmente distribuído e multi-plataforma, ainda de forma não muito organizada. Cada vez mais se tem pesquisado como integrá-la a outras tecnologias já existentes e bem definidas como também a novas tecnologias.

Já os SGBDs foram desenvolvidos há mais tempo e tiveram sua evolução motivada para o atendimento de novas aplicações, mas sem a

preocupação inicial de integração com novas tecnologias. Mas sendo a *Web* considerada como um grande sistema de bancos de dados distribuídos, nada mais natural o crescente interesse da comunidade científica e industrial na integração destas duas tecnologias. Assim, a utilização da tecnologia de banco de dados vem se tornando cada vez mais relevante para a gerência, o acesso, a disponibilidade e, principalmente, para a integração de informações também neste ambiente.

Sistemas baseados em bancos de dados podem ser usados para auxiliar a gerência de informação na *Web*, no que se refere a modelagem, consulta, integração, extração de informação e para construir ou reorganizar sites já existentes. Diversas propostas vêm sendo desenvolvidas e divulgadas, tanto para facilitar o trabalho dos desenvolvedores quanto dos usuários *Web* para integração com a tecnologia de bancos de dados.

3.3.1 - Tecnologias Utilizadas na *Web*

A *Web* fornece acesso a informações distribuídas, semelhante a uma arquitetura cliente/servidor. O *HTTP* (*Hypertext Transfer Protocol*) é usado para comunicação entre o cliente e o servidor *Web*. O cliente é composto por *browsers* que solicitam documentos na rede e os exibem. Podem existir aplicativos para exibir algum tipo de dado (imagem, som, ..) caso o *browser* não tenha esta capacidade.

Já o servidor *Web* é responsável por atender as requisições dos clientes *Web* por documentos armazenados no sistema de arquivos ou solicitar a execução de programas de aplicação para consultas a outras fontes de dados, utilizando, por exemplo *CGI* (*Common Gateway Interface*). *CGI* é uma interface padrão, existente na maioria dos servidores *Web*, cuja principal vantagem é a portabilidade da aplicação. A *Web* foi projetada para funcionar no “topo” da *Internet*, que é composta por uma grande variedade de computadores e redes que interagem entre si. Dessa forma, a *Web* incorpora naturalmente a característica de ser um ambiente distribuído e multi-plataforma, além de encapsular vários protocolos *Internet*, como *FTP*, *Gopher* e *Telnet.* Outra

característica importante é que ela é baseada em padrões abertos que permitem transferir, descrever a formatação e localizar a informação [Lima97].

Para a transferência de informação entre o servidor e o cliente foi proposto o protocolo de comunicação *HTTP*. Apesar de eficiente, este protocolo apresenta problemas, como no caso de aplicações que exijam a identificação do usuário. Para a apresentação e formatação de documentos, o mais comum ainda é o *HTML*, uma linguagem padronizada para criar documentos hipertexto, muito embora possam ser utilizados outros padrões, como no caso de imagens que podem estar nos formatos *GIF* ou *JPEG*. Os documentos são interpretados pelos *browsers* e formatados segundo as características de cada plataforma em que são exibidos.

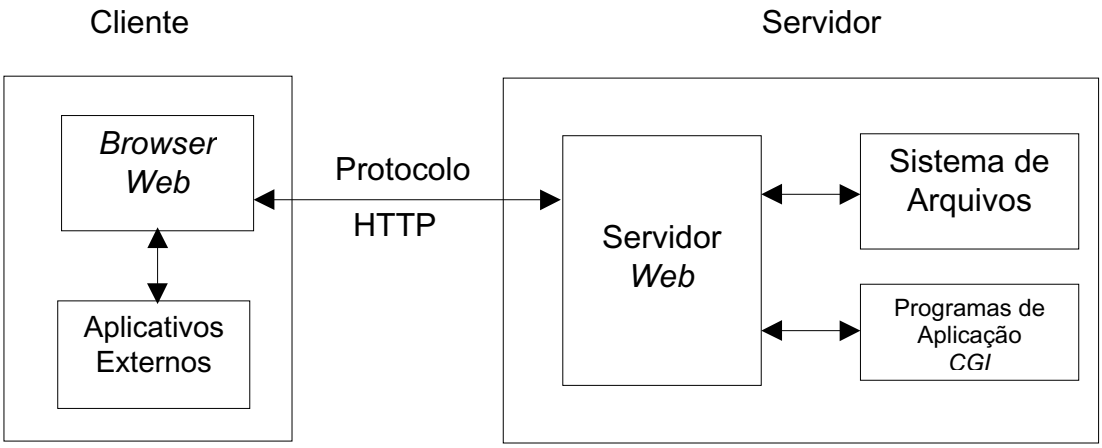

Figura 3.2 - Arquitetura da *Web*

Os documentos são acessados através de *URL* (*Uniform Resource Locators*), um formato para endereçamento de documentos *Web*. Servidores *Web* podem fornecer acesso transparente a outras fontes de dados, tais como bancos de dados, e não apenas textos *HTML*. Para isso, o servidor *Web* se comunica com programas de aplicação através de *gateways*. Este mecanismo permite aos clientes submeterem consultas que são processadas no *site* servidor, pelos servidores *Web* e de banco de dados. Além de *HTML* e *CGI,* existem outras maneiras de se criar documentos *Web.* Um exemplo são as *APIs*, utilizadas de forma semelhante no desenvolvimento de aplicações em

bancos de dados cliente/servidor. Apesar de ter vantagens sobre *CGI* como, compartilhar dados e recursos de comunicação com o servidor *Web*, a principal desvantagem é que são dependentes do servidor *Web* onde são implementadas.

Mais recentemente, tem-se utilizado de forma mais freqüente e crescente a linguagem *Java. Java* é uma linguagem de programação orientada a objetos, desenvolvida pela *Sun Microsystems* desde 1991, projetada para ser utilizada em ambientes distribuídos e heterogêneos, ou seja, tem como principal objetivo fornecer a independência de plataforma. Programas escritos em *Java*, chamados de "*applets*", são transferidos do servidor *Web* para o *browser* usando *HTTP*, sendo então processados por uma máquina virtual que é executada no *browser*.

A principal característica de *Java* é a portabilidade entre plataformas distintas, sendo assim, ideal para aplicação no ambiente *Web*. O compilador *Java* traduz o código fonte para uma forma intermediária, independente de máquina, denominada *bytecodes* que são enviados pela rede aos clientes, onde um interpretador *Java* específico para aquela plataforma interpreta e executa os *bytecodes*. Existem ainda as linguagens interpretadas *Javascript* e *Vbscript* que foram projetadas para atender ao ambiente *Web*. O código destas linguagens fica dentro de páginas *HTML*, sendo interpretado pelo *browser*. O grande problema é que são linguagens proprietárias.

3.3.2 - Banco de Dados e *Web*

Existem diversas técnicas e várias ferramentas para acessar bancos de dados pela *Web*. O mais antigo e provavelmente o mais utilizado é baseado em *CGI*. *CGI* é um padrão que permite aos servidores *Web* acessarem aplicações externas, também chamadas de *CGI scripts*, que são executadas no *site* servidor, sendo suportado por todos servidores *Web*. Um *CGI script* é inicializado pelo servidor *Web* devido a uma requisição do cliente e é executado no *site* servidor. O *CGI script* recebe os parâmetros do usuário, gera páginas *HTML* dinamicamente, as encaminha para o servidor *Web* e este as transfere para o cliente. *CGI scripts* podem ser implementados em qualquer

linguagem de programação, muito embora devam ser usadas aquelas mais largamente utilizadas para permitir portabilidade. Assim, uma opção é utilizar *CGI scripts* escritos em *Java*, que permitem uma melhor interação com os sistemas de bancos de dados, pois superam as limitações do *HTTP*. Dentre as várias possibilidades, a mais importante é o *JDBC (Java Database Conectivity*) para acesso a bancos de dados relacionais através de *Java*. *JDBC* é uma *API*, inspirada no padrão *ODBC* da *Microsoft*, que é disponibilizada por vários fornecedores [MST97].

Existem ainda problemas que não foram totalmente resolvidos, como os conceitos de transação, segurança, recuperação e falhas, performance etc., utilizados em bancos de dados e que têm de ser repensados para utilização no ambiente *Web*. Em aplicações críticas na *Web* usando SGBDs, pode ser necessário que uma autenticação entre cliente e servidor seja realizada para garantir a privacidade da transação. No caso de *Intranets*, a sua comunicação com o ambiente exterior (*Internet)* também deve ser motivo de aplicação de mecanismos de segurança. Neste caso, os servidores *Web* podem ser configurados para restringir o acesso aos dados somente ao ambiente interno da empresa, ou podem ser usados *firewalls* para controle dos acessos externos [Lima97].

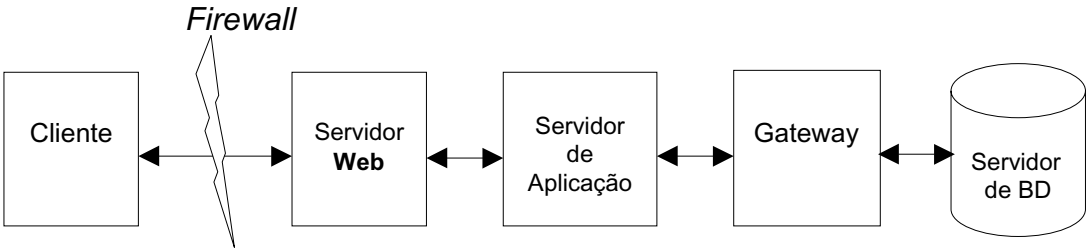

Figura 3.3 - Arquitetura de três camadas

Na arquitetura de três camadas, a primeira delas é o cliente, onde residem os programas de apresentação (interface com o usuário). Os clientes podem estar dentro ou fora do *firewall*, dependendo se a aplicação é *Internet* ou *Intranet*. A segunda camada é formada pelo servidor *Web*, pelo servidor de aplicação e pelo *gateway* de comunicação. Neste caso, todo o código da

aplicação reside em um mesmo local, facilitando a atividade de manutenção. A terceira camada corresponde ao servidor de banco de dados, que contém os dados, metadados e controle de acesso.

As propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) [EN94] [SKS99] de uma transação, implementadas nos SGBDs, devem ser estendidas para aplicabilidade no ambiente *Web* e ainda é assunto de pesquisa [Bill98]. O problema é a limitação do ambiente *Web*, já que nem os clientes, nem os servidores *Web* e nem mesmo o protocolo *HTTP*, foram projetados para tratar de controle de falhas e recuperação nos mesmos moldes que os SGBDs.

Uma outra aplicação dos sistemas de banco de dados é na construção e manutenção de *Web sites*, considerando que um *Web site*, essencialmente, fornece acesso a dados existentes em diferentes fontes. Além disso, a tarefa de fornecer uma interface *Web* para dados existentes em diferentes fontes, a partir de um único SGBD, é um problema ao se criar um *Web site*. Tarefas como, atualizar e reestruturar um *site* e garantir restrições de integridade dos dados, podem ser melhor executadas com a utilização de sistemas de bancos de dados.

Assim, em diversas maneiras, a *Web* não é similar a um SGBD. Por exemplo, não existe uma estrutura uniforme, restrições de integridade, transações, modelo de dados ou linguagem de consulta padrão. Mas, a utilização de SGBDs pode diminuir a complexidade da *Web* fornecendo novos serviços. Uma observação importante é que um *Web site* pode ser visto como sendo não um único banco de dados, mas um sistema de informação composto pela integração de diversas fontes de dados juntamente com estruturas de navegação complexas.

Diversas novas tentativas terão impacto significativo no uso da tecnologia de bancos de dados em aplicações *Web* [FLM98] [Kram97] [OBJS97] [ÖV99] [Ju97]. Como exemplo, o uso de *XML* (*Extensible Markup Language*) [XML01] [Ligh99] pode auxiliar a aplicabilidade dos conceitos de bancos de dados para a *Web*, por fornecer a estrutura necessária e com

formato vastamente aceito. Mesmo assim, o problema de integração semântica dos dados da *Web* ainda continua.

3.3.3 – Integração de Informações na *Web*

A *Web* tornou-se popular, sendo o veículo mais importante para disseminação de informação. Já os sistemas de informação estão cada vez mais heterogêneos pois as soluções de um fornecedor nem sempre podem ser facilmente integradas com as de outros. Mesmo dentro de uma mesma empresa é comum se encontrar recursos computacionais de vários fabricantes.

Um aspecto importante da *Web* é a existência de dados não estruturados ou semi-estruturados, onde não se tem um esquema fixo para os *sites*, diferenciando dos conceitos de bancos de dados até então utilizados. Para estes casos devem ser revistos tais conceitos ou se encontrar alternativas para se permitir a definição e o armazenamento de metadados da *Web*, como por exemplo *XML*.

Atualmente, *XML* está sendo considerada fundamental para interoperabilidade, pois possibilita a entrega de qualquer tipo de informação estruturada por toda a *Web* [Fing00]. Receber documento em *XML* em vez de *HTML* torna o cliente mais auto-suficiente: *HTML* foi projetada para exibição e não para intercâmbio de informação. No caso de *XML*, se o cliente quiser por exemplo, reformatar a informação, não é necessário retornar ao servidor: basta carregar uma folha de estilo diferente. O problema ainda é a não padronização dos termos para *XML*, onde cada fornecedor define seu próprio vocabulário. Assim, a tendência na *Web* é fazer com que documentos que até então eram do tipo não estruturados, possam se tornar mais estruturados com o uso de *XML*, facilitando o processo de integração de dados.

A tarefa de um sistema de integração de informações na *Web* é responder a consultas que podem requerer extrair e combinar dados de múltiplos *sites*. Além disso, sistemas de integração de dados na *Web* têm de lidar com um número grande e crescente de *Web sources* (combinação de *Web site* com seu *wrapper* associado), poucos metadados disponíveis sobre suas características, grande nível de autonomia e volatilidade.

Também tem sido agressivo o movimento para integração dos sistemas de negócio entre empresas. Muito embora a estrutura da *Web* forneça conectividade básica, ela não oferece uma plataforma de integração padrão. Os *web* sites não têm sido capazes de se intercomunicar sem extensiva codificação manual. Um mercado aberto requer suporte para conexões do tipo muitos-para-muitos e a integração via tecnologia proprietária ameaça limitar o número de conexões. Assim, *XML* na forma de uma linguagem de definição de interface (*IDL*) para a *Web*, pode proporcionar um mecanismo prático e seguro para integração de sistemas de negócios nos padrões do protocolo *Web* [Ligh99].

Uma discussão importante na construção de *middleware* de integração de dados na *Web* é que modelo deve ser usado, se o virtual ou o com materialização de dados. No primeiro, os dados continuam nas *Web sources* (não existem réplicas), e as consultas são feitas ao sistema integrador de dados, decompostas em sub-consultas e enviadas para as *Web sources* componentes necessárias. Já no modelo com materialização, os dados de múltiplas *Web sources* são consolidados e armazenados em um *data warehouse* (*DW*) [Inmo96] [Kimb96], para onde são dirigidas as consultas.

O modelo virtual é mais apropriado quando o número de *Web sources* é grande, os dados são atualizados com freqüência e existe pouco controle sobre as *Web sources*, apesar de muitos dos problemas existentes em um modelo também existirem no outro. A vantagem em usar o modelo com materialização é permitir melhor performance para a aplicação e não gerar um custo adicional de acesso às *Web sources*, muito embora tendo o custo da replicação.

A construção de sistemas *middleware* de integração tem sua demanda aumentada a cada dia, principalmente devido à *Web*. Como as empresas estão investindo atualmente em diversos sistemas, certamente terão de investir no futuro na construção de sistemas *middleware* de integração. Uma tarefa mais complexa e desafiadora é a integração de aplicações entre empresas (*EAI – Enterprise Application Integration*), onde a integração dos processos de negócios entre diferentes empresas deve ser suportada [Lint99] [Hass00]. O

fundamental é a disponibilidade de sistemas integradores que sejam flexíveis o suficiente para atender a diferentes requisitos.

Muito embora exista esta demanda crescente por sistemas para integração de dados, sempre se recai no problema de construir uma "nova" solução para cada caso. Esta constatação não é específica para o desenvolvimento de *middleware* de integração mas, de maneira geral, vale para a grande maioria dos sistemas de *software* construídos. Assim, para o desenvolvimento e a manutenção de sistemas de *software*, como no caso dos sistemas integradores de dados, devem ser utilizadas técnicas que facilitem o projeto e que possibilitem a redução do tempo de implementação e de manutenção. Uma alternativa importante é a possibilidade de reuso dos projetos e dos códigos já existentes, em vez de se recriar os mesmos processos de negócios repetidamente. Este procedimento se torna necessário devido às rápidas mudanças tecnológicas e às constantes alterações nos requisitos das aplicações, o que implica uma revisão periódica do projeto na tentativa de não permitir que os sistemas se tornem obsoletos em pouco tempo.

3.4 – Desenvolvimento de Sistemas de *Software*

Durante o processo de desenvolvimento de *software* algumas questões estão sempre presentes [Meye89]:

- Por que o desenvolvimento de *software* não é semelhante ao de *hardware*?
- Por que não existem catálogos de componentes de *software* como os catálogos de componentes de *hardware*?
- Por que não construir *software* escolhendo e combinando componentes de *software* disponíveis?
- Por que todo novo desenvolvimento de *software* tem de partir do zero?
- Por que não reusar *software*?

Reuso de *software* é o processo de criar sistemas de *software* a partir de *software* existente. A maior motivação para reusar *software* é reduzir o tempo e o esforço requeridos para construir sistemas. Muito embora reuso seja

um dos principais tópicos de pesquisa em engenharia de *software* nos últimos anos, ainda hoje reusar *software* não é uma tarefa simples. Diversas são as técnicas disponíveis para reuso [BR87] [Krue92] mas, recentemente, a técnica de *frameworks* têm sido aplicada com sucesso para facilitar o reuso seja de código, de projeto ou de implementação.

Uma forma ideal de reuso de *software* seria aquela em que, por analogia a *hardware*, componentes fossem fornecidos em catálogos e que pudessem ser facilmente conectados para construir um novo sistema, sem se precisar conhecer detalhes sobre sua implementação. Nos últimos anos tem sido significativo o esforço investido para construir repositórios de componentes de *software* para o desenvolvimento de *software* industrial [WN95]. Embora certas áreas de aplicação tenham tido algum sucesso, no geral, a técnica de reuso tem sido muito mais difícil na prática do que o esperado.

Para se integrar um componente reusável em um sistema de *software*, o desenvolvedor deve entender claramente a interface do componente, ou seja, as propriedades do componente que interage com outros componentes no contexto da arquitetura de integração. A interface de um componente é uma abstração na qual os detalhes internos são suprimidos. A técnica de reuso necessita de uma arquitetura para guiar a integração dos componentes. Um desenvolvedor de *software* usa esta arquitetura para combinar uma coleção de componentes selecionados e especializados para construir um sistema de *software* completo.

Na realidade, construir *software* de forma modular é uma idéia antiga e certamente todos os grandes sistemas têm sido assim desenvolvidos. Recentemente, o desenvolvimento de *software* baseado em componentes [ND95] [Pree97] [Szyp98], ganhou importância principalmente para a configuração de sistemas. A construção de sistemas configuráveis ainda não tem uma solução bem definida principalmente no que se refere à interdependência entre os componentes. A configurabilidade requerida por um sistema pode ser obtida através de um *framework* que suporte o desenvolvimento de *software* através de componentes. Em tal *framework* os

componentes existentes podem ser modificados e novos componentes podem ser construídos para atender à especificidade da aplicação, sendo que cada componente tem que apresentar uma interface compatível, permitindo sua composição com o *framework*. Essa necessidade se dá porque um grande número de novas áreas de aplicação requer suporte para múltiplas versões de suas entidades e, através de reuso ou da adaptação de um *framework*, novos sistemas customizados podem ser mais facilmente e rapidamente construídos.

Framework é uma arquitetura semi-completa que pode ser instanciada para produzir aplicações customizadas, permitindo o reuso de análise, de projeto e de código [FSJ99a]. Antes de apresentar os conceitos de *frameworks*, serão apresentados os de arquitetura de *software* e de componentes.

3.4.1 - Arquitetura de *Software*

Pode-se dizer que a arquitetura de *software* teve início quando programas passaram a ser divididos em módulos, sendo os programadores os responsáveis pela composição dos módulos e pela funcionalidade do sistema como um todo [SG96]. Com a necessidade de se construir sistemas cada vez mais complexos e maiores, o problema não mais se restringe a escolher algoritmos e estruturas de dados e a implementar módulos arbitrariamente, mas sim projetar e especificar a estrutura do sistema. Os problemas estruturais incluem a organização de um sistema, como: a integração de componentes, estruturas de controle global, protocolos de comunicação, acesso aos dados, sincronismo, a funcionalidade e composição dos elementos de projeto e a distribuição física.

Arquitetura de *software* envolve a descrição de elementos dos quais os sistemas são construídos, a interação entre estes elementos, os padrões para guiar a composição e as restrições destes padrões. A arquitetura de um sistema de *software* define o sistema em termos dos seus componentes computacionais e das interações entre estes componentes. O projeto de um sistema pode ser construído em níveis. Em cada nível são encontrados componentes primitivos e compostos, regras de composição para a construção de componentes não primitivos e as regras de comportamento para fornecer

as semânticas do sistema. O *software* também tem seus níveis de projeto, como: o nível de arquitetura (associação dos componentes), nível de código (algoritmos e estruturas de dados) e nível de execução (mapeamento de memória, alocação de registradores, ...) [SG96].

Segundo a definição de [BMR+96] arquitetura de *software* é uma descrição dos sub-sistemas e componentes de um sistema de *software* e dos relacionamentos entre eles. Sub-sistemas e componentes são tipicamente especificados em diferentes visões, para mostrar as propriedades funcionais e não funcionais relevantes de um sistema de *software*. Um bom projeto de arquitetura é sempre um fator determinante para o sucesso de um *software*. A questão é como escolher uma arquitetura apropriada para um problema específico. Uma base arquitetural para a integração de componentes reusáveis auxiliaria na solução de muitos dos problemas de construção de *software*. Uma das propostas mais amplamente aceita na área de orientação a objetos é a de se usar *frameworks* para domínios específicos, que poderiam ser instanciados para produzir novos produtos no domínio.

Arquitetura de *software*, gerência de configuração e sistemas configuráveis são três áreas de pesquisa que têm evoluído separadamente, mas compartilham a noção de configuração [HHW98]. Arquitetura de *software* se destina ao projeto de alto nível de um sistema. O projeto de um sistema é dividido em partes menores denominadas componentes que são então combinados para formar um sistema completo através de conexões explicitamente modeladas. Gerência de configuração suporta a fase de implementação de um sistema de *software*, fornecendo um mecanismo para escolher uma configuração conveniente para o sistema. Sistemas configuráveis se concentram na gerência de um sistema a partir do início de sua operação.

Arquitetura de *software* e sistemas configuráveis necessitam ser capazes de selecionar um subconjunto destes componentes que, quando colocados juntos formem um sistema consistente. Aqui alguns pontos se tornam importantes [HHW98]:

- ***Composição***: sistema como um conjunto de componentes interconectados;

- ***Consistência***: garantir consistência quando os componentes são combinados para formar um sistema;
- ***Construção***: suportar a construção de um sistema executável;
- ***Versão***: existência e relacionamentos entre múltiplas versões de componentes;
- ***Seleção***: seleção de componentes a partir de um conjunto de componentes disponíveis;
- ***Dinamismo***: modelar alterações de um sistema após seu desenvolvimento.

Ao nível arquitetural, os componentes são conectados ligando a funcionalidade fornecida por um componente com a funcionalidade requerida por outro. Versão e seleção são os mais complicados: controle de versão e a seleção de componentes que constituem uma configuração têm de ser feitas à mão, sem qualquer guia a partir do modelo arquitetural do sistema.

3.4.2 - Componentes

Além dos problemas estruturais vistos anteriormente, os sistemas de *software* necessitam ser cada vez mais abertos e distribuídos. Tais sistemas são abertos não apenas com relação a conexão a redes, a interoperabilidade com plataformas de *hardware* e *software* heterogêneos mas, principalmente, em termos de atender à evolução e alteração dos requisitos. Sistemas abertos devem garantir esta propriedade em três modos [ND95]:

- ***Topologia***: redes configuráveis;
- ***Plataforma***: heterogeneidade de *software* e *hardware*;
- ***Evolução***: alteração dos requisitos.

Segundo a definição de [BMR+96], um componente é uma parte encapsulada de um sistema de *software*, que tem uma interface e serve como um bloco para a construção da estrutura de um sistema. Ao nível de linguagem de programação, componentes podem ser representados como módulos, classes, objetos ou um conjunto de funções relacionadas. Um relacionamento define uma conexão entre os componentes, podendo ser estático ou dinâmico.

Relacionamentos estáticos são mostrados diretamente no código fonte e estão relacionados à localização dos componentes dentro da arquitetura. Relacionamentos dinâmicos tratam de conexões temporais e interação dinâmica entre componentes, não sendo facilmente visíveis na estrutura estática do código fonte.

Componentes são unidades de produção, aquisição e desenvolvimento independente, que interagem para formar um sistema [Szyp98]. Construir novas soluções através da combinação de componentes produzidos e/ou adquiridos de terceiros, aumenta a qualidade e suporta o rápido desenvolvimento de um sistema permitindo disponibilizá-lo mais cedo para operação. Componentes necessitam de modularidade de requisitos, de arquitetura, de projeto e de implementação. Componentes encorajam o movimento dos sistemas atuais, monolíticos e pesados, para estruturas modulares que oferecem os benefícios de permitir adaptabilidade, escalabilidade e manutenibilidade.

Em um mundo onde os requisitos dos negócios mudam rapidamente, construir *software* é, geralmente, um processo arriscado, pois o mesmo pode tornar-se obsoleto muito cedo, apesar de estar mais afinado com os procedimentos organizacionais da empresa e permitir a mesma ter um diferencial no mercado. Adquirir um pacote padrão no mercado, pode implicar em rever todos os procedimentos e forçar trocas na cultura e operação da organização, o que pode levar a resultados imprevisíveis na sua área de atuação no mercado. Assim, desenvolver *software* através da integração de componentes representa um meio termo entre o desenvolvimento integral de um sistema ou a aquisição de um pacote pronto [Szyp98]. Muito embora cada componente a ser comprado seja um produto padronizado, o processo de montar componentes permite a oportunidade para uma customização significativa. Não usar componentes disponíveis, significa reinventar soluções. Um produto que utiliza componentes se beneficia da combinação da produtividade e da inovação de todos os vendedores de componentes.

Componentes são caracterizados pela dissociação entre especificação e implementação, que interagem com outros componentes através de interfaces

e que são projetados visando composição uma [ND95]. Sempre que componentes são integrados para executar uma tarefa comum, existe um contrato implícito entre eles sobre os termos de colaboração. Para ser capaz de verificar se uma determinada configuração está correta, os contratos necessitam ser explícitos para serem comparados no caso de discrepância. Um exemplo de contrato entre componentes é o esquema de um sistema de banco de dados que especifica as condições sobre as quais um banco de dados pode ser acessado e quais condições devem ser respeitadas pelos programas que executam transações sobre ele.

Um componente tem que ser projetado para ser usado em uma combinação com outros componentes e não de forma isolada [Gogu86] [Pree97]. Componentes têm de ser projetados para reuso e ser parte de uma arquitetura (*framework*). Funções, macros, procedimentos, *templates* e módulos são exemplos de componentes. Componentes em *frameworks* podem padronizar interfaces e código genérico para vários tipos de abstrações de *software*, além de poderem, também, ser de outros tipos de entidades e não somente *software*, como: especificações, documentações, dados de teste e aplicações exemplo.

Frameworks baseados em componentes vem tendo destaque recentemente. Formam implementações parciais, especificando a natureza e o modo de estender o *framework* através de componentes do tipo "*plug-and-play*" [Lars00]. *Frameworks* permitem construir um novo sistema, usando partes pré-definidas e a necessidade de apenas um pequeno número de novos componentes [Hopk00]. Nesta abordagem, *frameworks* oferecem uma base para se construir sistemas altamente moldados e configurados para um domínio específico [FHB00].

3.5 – *Frameworks*

Dentro da comunidade de orientação a objetos, a arquitetura de *software* recebeu uma forma particular de ser representada através de *frameworks*. A técnica de *framework* oferece uma infra-estrutura adequada para permitir flexibilidade, pois sendo um sub-sistema de *software* de um domínio

específico, pode ser ajustado para aplicações individuais. Diversas são as definições para *framework* encontradas na literatura com pequenas diferenças entre elas.

Um *framework* consiste em uma grande estrutura (arquitetura) que pode ser reusada para a construção de um novo sistema. Um *framework* define uma arquitetura para uma família de sistemas fornecendo partes predefinidas para sua construção e partes que devem ser adaptadas para uma função específica. Em um ambiente orientado a objetos, um *framework* é composto de classes abstratas e concretas e a sua instanciação consiste da composição ou da herança de classes. As classes mais importantes de um *framework* são as abstratas. O ideal seria instanciar o *framework* somente a partir das classes de uma biblioteca de componentes que contém subclasses concretas das classes do *framework*. Mas, como nenhuma biblioteca por mais completa que seja possuirá todos os componentes necessários, certamente será preciso derivar novas subclasses concretas das classes abstratas do *framework*. Herança requer uma compreensão profunda da classe que se quer especializar, por isso o reuso de componentes já existentes é a melhor estratégia.

[Pree95] apresenta um *framework* como sendo formado por *frozen spots* e *hot spots*. *Frozen spots* definem a arquitetura global de um sistema, seus componentes básicos e os relacionamentos entre eles, ficando imutáveis em qualquer instanciação do *framework*. Já os *hot spots* são partes específicas para cada sistema e são projetados para serem genéricos e poderem ser adaptados de acordo com as necessidades da aplicação. Assim, um *framework* reusa código pois permite a construção de uma aplicação a partir de uma biblioteca de componentes, que podem ser facilmente integrados uns com os outros pois eles utilizam interfaces comuns.

Segundo [FS97] um *framework* é uma aplicação reusável e semi-completa que pode ser especializada para produzir aplicações customizadas. *Frameworks* garantem modularidade por encapsular detalhes de implementação voláteis junto com interfaces estáveis. As interfaces estáveis fornecidas pelos *frameworks* garantem reuso por definir componentes genéricos que podem ser reaplicados para criar novas aplicações.

[MB97] define um *framework* orientado a objetos como sendo um conjunto de classes que embute um projeto abstrato de soluções para uma família de problemas relacionados. Em outras palavras um *framework* é um projeto abstrato e uma implementação para uma aplicação em um dado domínio de problema. No processo de desenvolvimento de *software* baseado em *frameworks* são definidas as seguintes fases:

- Fase do desenvolvimento do *framework*;
- Fase do uso do *framework* (também denominada fase de instanciação do *framework* ou ainda fase do desenvolvimento da aplicação);
- Fase de manutenção e evolução do *framework*.

O processo de desenvolvimento de *frameworks* difere do desenvolvimento de aplicações padrão pois requer uma etapa extra: a etapa de instanciação, ou seja, completar os *hot-spots*. O problema é que nenhuma técnica de projeto fornece construtores adequados para descrever os *hot-spots* de *frameworks*, que precisam ser representados de forma diferente do resto do sistema. A instanciação de um *framework* é muito mais complexa do que, simplesmente, agregar componentes aos *hot-spots*, pois estes podem ter interdependências ou serem opcionais. Além disso, podem existir diferentes customizações para o *framework* oferecer a mesma funcionalidade. Os problemas mais comuns na evolução de *frameworks* são a sua complexidade estrutural, as mudanças de domínio e as novas funcionalidades [Font99].

Quando usados em conjunto com *patterns*, bibliotecas de classes e componentes, os *frameworks* podem, significativamente, aumentar a qualidade do *software* e reduzir o esforço de desenvolvimento [BMMB98]. *Frameworks*, geralmente falham se não forem reconhecidos e resolvidos os seguintes desafios: esforço de desenvolvimento, curva de aprendizado, facilidade de integração e manutenção, validação e remoção de erros, eficiência e falta de padrões [FS97]. A visão original de reuso de *software* foi baseada em componentes mas, *frameworks* não são componentes de *software* como eles foram originalmente previstos. *Frameworks* são mais customizáveis que componentes e têm interfaces mais complexas [John97].

Frameworks e componentes são tecnologias diferentes, mas que cooperam entre si [John97]. Primeiro, *framework* fornece um contexto reusável para componentes e um modo padrão para que componentes possam manusear erros, trocar dados e invocar operações um para outro. Mas qualquer tipo de *framework* fornece interfaces padrão que possibilitam o reuso dos componentes existentes. Um segundo modo no qual *frameworks* e componentes trabalham juntos é que, *framework* torna mais fácil o processo de desenvolver novos componentes.

Como visto anteriormente, a vantagem do desenvolvimento baseado em *frameworks* é o alto nível de reuso no desenvolvimento da aplicação. Embora o desenvolvimento de um *framework* requeira considerável esforço, os benefícios correspondentes durante o processo de desenvolvimento da aplicação geralmente justificam o esforço inicial.

3.5.1 - Classificação dos *Frameworks*

Segundo [FS97], os *frameworks* podem ser classificados em:

- *Frameworks de Infra-estrutura de Sistemas*: simplificam o desenvolvimento de sistemas de infra-estrutura portáveis e eficientes, tais como: sistemas operacionais, comunicação, interface com o usuário e linguagens de programação.
- *Frameworks de Integração (middleware)*: são usados para integrar aplicações distribuídas e componentes. Eles são projetados para garantir a capacidade dos desenvolvedores de *software* com relação a modularização, reuso e estender sua infra-estrutura de *software* para trabalhar em um ambiente distribuído. Tais *frameworks* representam um mercado próspero e estão, rapidamente, tornando-se produtos de primeira necessidade.
- *Frameworks de Aplicação Empresarial*: abordam grandes domínios de aplicação, tais como: telecomunicações, manufatura etc. e são a pedra fundamental de atividades de negócios da empresa. Tais *frameworks* são caros para desenvolver e/ou comprar.

Frameworks também podem ser classificados pelas técnicas usadas para estendê-los, que variam desde *frameworks* caixa branca (*white box*) até *frameworks* caixa preta (*black box*) [SM98]. Os *frameworks* caixa branca dependem fortemente dos aspectos de linguagens orientadas a objetos como herança e ligação dinâmica para prover extensibilidade. Já os *frameworks* caixa preta, suportam extensibilidade definindo interfaces para componentes que podem ser ligados a um *framework* através de composição. A funcionalidade existente é reutilizada definindo-se componentes que estão em conformidade com uma interface particular.

Framework caixa branca requer dos desenvolvedores de aplicação conhecimento íntimo de cada estrutura interna do *framework*. Já os *frameworks* caixa preta são estruturados usando composição de objetos e delegação em vez de herança. Como resultado, *frameworks* caixa preta são geralmente mais fáceis de usar e estender que os *frameworks* caixa branca. Entretanto, *frameworks* caixa preta são mais difíceis de desenvolver pois requerem que os desenvolvedores do *framework* definam interfaces que antecipem uma grande faixa de potenciais casos de uso. No início de sua vida, o *framework* é concebido como um *framework* caixa branca, podendo evoluir até se tornar um caixa preta.

Atualmente, não existem padrões vastamente aceitos para projeto, implementação, documentação e adaptação de *frameworks* [FS97] [Font99]. Muitos especialistas em *frameworks* preferem os caixa preta aos caixa branca, pois os *frameworks* caixa preta enfatizam relacionamentos dinâmicos em vez de relacionamentos de classes estáticas. A documentação é uma atividade de alto custo e que falha na captura de papéis estratégicos e colaborações entre *frameworks* componentes. Diversos padrões, certamente, surgirão para o desenvolvimento, adaptação, interoperabilidade e padrões de integração de *frameworks*.

As habilidades requeridas para produzir *frameworks* com sucesso, geralmente, ficam retidas nas cabeças dos desenvolvedores especialistas. O desenvolvimento de aplicações será, cada vez mais, baseado na integração de múltiplos *frameworks*. Entretanto, muitos *frameworks* de geração mais antiga,

foram projetados para extensão interna em vez de ser por integração com outros *frameworks* desenvolvidos externamente. Integrar *frameworks* que não foram projetados para a interoperação com outros *frameworks* é uma tarefa complexa.

Diversos trabalhos abordam projeto e uso de *frameworks*: [BGK+97], [BMA97], [BMMB98], [Bosch98], [CDSV97], [DMNS96], [DW99], [Eske99a], [FJ99], [FSJ99b], [IBM93], [IBM99], [RJ96], [Rieh00], [Schm97] e [SF97].

3.5.2 – Evolução dos *Frameworks*

Os requisitos da aplicação mudam freqüentemente, o que pode acarretar mudanças no *framework*. Com a evolução dos *frameworks*, as aplicações que os utilizam devem evoluir também. As atividades de manutenção dos *frameworks* incluem alteração e evolução. A manutenção de um *framework* pode ocorrer de diferentes formas [MB98]:

- Para reorganizar a estrutura interna em resposta a novos requisitos, mas sem afetar a funcionalidade externa;
- Para alterar, estender e remover funcionalidades.

O desenvolvimento tradicional de aplicações baseadas em *framework* assume que a aplicação é baseada em um único *framework* que é estendido com código específico da aplicação. Entretanto, podem ser identificados no desenvolvimento de aplicações a necessidade de se fazer uso de múltiplos *frameworks* que devem ser integrados para atender aos requisitos da aplicação. O principal problema, neste caso, é desenvolver uma aplicação através do reuso de *frameworks* existentes. Estes têm de ser estendidos ou compostos com outros componentes de *software* para atender aos requisitos da nova aplicação em desenvolvimento. Três diferentes componentes podem ser usados para adaptação e composição: novo código fonte para aplicação específica, biblioteca de classes ou outros *frameworks*. Toda aplicação desenvolvida baseada em *frameworks* requer o desenvolvimento de código específico da aplicação mas, neste caso, sem os problemas de integração.

Geralmente os *frameworks* têm sido desenvolvidos para reuso por extensão com a escrita de novos códigos específicos para a aplicação e não para a composição com outros componentes de *software* [FS97] [MB97]. Este enfoque sobre reuso através da extensão causa problemas quando da tentativa de integrar *frameworks*.

3.5.3 – Integração de *Frameworks*

Se uma biblioteca de classes ou um *framework* tem de ser integrado com outro *framework*, problemas de integração também podem ocorrer. Uma das causas é chamada coesão de *framework*. Ela ocorre devido ao fato que, quando um *framework* é integrado com outro módulo de *software*, esta integração deve atender ao comportamento do domínio específico da aplicação bem como estabelecer um correto comportamento de interação entre eles. A questão aqui é, garantir que um módulo desenvolvido separadamente possa atender a estes objetivos sem efeitos colaterais.

Uma outra causa é o domínio de cobertura de um *framework* pois ele não necessita cobrir todo o domínio. Quando de sua utilização, o mesmo deve ser estendido ou integrado a outro módulo para atender aos requisitos da aplicação. Assim, a integração pode gerar sobreposição de entidades, o que pode causar mais esforço na integração do que simplesmente estender o *framework* com código novo.

Uma terceira causa é a intenção de projeto, ou seja, deve ser definido explicitamente pelo projetista se o *framework* pode ser usado apenas por extensão ou o que será necessário para a sua futura integração.

Por fim uma outra causa é a dificuldade de acesso e a clareza do código fonte do *framework* para extensão ou integração. Se o código fonte não está disponível a solução é o uso do *design pattern Adapter*, que será apresentado na seção 3.6.2.

[MB97] e [Matt00] apresentam em mais detalhes os problemas mais comuns encontrados no processo de integração de *frameworks* e as possíveis soluções.

3.6 – *Patterns*

Uma área relacionada a *frameworks* é a de *patterns* [BMR+96] [GHJV95]. *Patterns* têm se popularizado na área de orientação a objetos como uma forma de reutilizar informação de projeto, descrevendo: um problema a ser resolvido, uma solução e o contexto na qual esta solução funciona. *Patterns* documentam *frameworks* [John92] e ajudam a garantir o correto uso de sua funcionalidade. Aplicações que usam *frameworks* têm que estar em conformidade com o projeto e o modelo de colaboração dos *frameworks* e assim, então, *frameworks* também geram *patterns* nas aplicações que os usam. Na verdade, a partir do momento que *frameworks* são implementados várias vezes eles passam a representar um tipo de *pattern* [Pree96].

Patterns têm se tornado um modo popular para reusar informação de projeto na comunidade de orientação a objetos [Clin96] [Pesc98]. Um *pattern* descreve um problema a ser resolvido, uma solução e o contexto no qual a solução trabalha [GHJV95]. *Patterns* são os elementos micro arquiteturais de *frameworks* [John97]. Da perspectiva de *frameworks*, *patterns* podem ser vistos como seus blocos construtores, pois um único *framework*, usualmente, contém muitos *patterns*. Da perspectiva de *patterns*, um *framework* pode ser visto como um *pattern* para um sistema de *software* completo em determinado domínio de aplicação.

Quando um especialista trabalha sobre um problema em particular, normalmente não cria uma nova solução para cada caso que seja diferente das soluções anteriores. Na verdade, sempre se recai em problemas similares que já tenham sido resolvidos, cujas soluções podem ser utilizadas para resolver novos problemas. Esta é a grande motivação do uso de *patterns*. Um *pattern* endereça um problema de projeto recorrente que está relacionado a situações específicas de projeto e apresenta uma solução para isso. *Patterns* documentam experiências de projetos existentes e bem sucedidos não sendo criados artificialmente [BMR+96].

O mesmo ocorre no desenvolvimento de um sistema de *software*: um *pattern* trata um problema específico, recorrente no projeto ou na implementação. *Patterns* podem ser usados para construir arquiteturas de

software com propriedades específicas e, por outro lado, muitos *patterns* são encontrados em arquiteturas de *software*.

São diversas as propriedades de *patterns* para arquitetura de *software* [BMR+96]:

- Aplica-se a um problema de projeto recorrente, que se manifesta em uma situação específica de projeto e apresenta uma solução;
- Documentam experiências de projeto existentes e bem sucedidas;
- Identificam e especificam abstrações que estão acima do nível de classes, instâncias ou de componentes;
- Fornecem um vocabulário e um entendimento comum para princípios de projeto;
- São uma das soluções para a documentação de arquitetura de *software*;
- Permitem a construção de *software* com propriedades definidas;
- Auxiliam a construção de arquiteturas de *software* complexas e heterogêneas;
- Auxiliam a administrar a complexidade de *software*.

3.6.1- Classificação dos *Patterns*

[BMR+96] classifica os *patterns* em três categorias: *patterns* arquiteturais, *design patterns* e idiomas.

Patterns arquiteturais são estruturas pré-definidas para a construção de arquiteturas de *software*, especificando as propriedades do sistema para uma determinada aplicação. Os *patterns* arquiteturais estão relacionados a sub-sistemas ou componentes. A seleção de um *pattern* arquitetural é uma decisão de projeto fundamental quando do desenvolvimento de um sistema de *software*.

Design patterns são unidades arquiteturais menores que fornecem um modelo para refinar os sub-sistemas ou os componentes de um sistema de *software*, bem como os relacionamentos entre eles. A maioria dos *patterns* arquiteturais permitem que problemas possam ser resolvidos com *patterns* menores.

Idiomas são *patterns* de baixo nível específicos para uma linguagem de programação.

Assim, os *patterns* arquiteturais podem ser usados no início do projeto, os *design patterns* durante o processo de refinamento do projeto e os idiomas durante a fase de implementação.

[GHJV95] classifica os *design patterns* de acordo com o objetivo, podendo ser de criação, de estrutura e de comportamento e ainda de acordo com o escopo, se são aplicados a classes e/ou a objetos. O enfoque de [GHJV95] é para programas orientados a objetos, sendo sempre referenciados se são aplicados a classes e/ou objetos. Já o enfoque de [BMR+96] é o de *patterns* no contexto de arquitetura de *software*, envolvendo *patterns* arquiteturais, *design patterns* e idiomas, sendo aplicados a componentes de uma forma geral.

3.6.2 – Alguns *Patterns* para Integração

Nesta seção serão apresentados alguns *patterns* arquiteturais *e design patterns*, a partir das referências [BMR+96] e [GHJV95] que podem auxiliar no processo de desenvolvimento de componentes e de *frameworks* para integração de dados.

*a) **Adapter***

Converte a interface de uma classe para uma outra interface, permitindo que classes trabalhem em conjunto, o que não seria possível devido a incompatibilidade de interfaces [GHJV95]. *Adapter* é também conhecido como *Wrapper*. A motivação para o uso do *Adapter* é que muitas vezes uma classe não é reusável somente porque sua interface não combina com a interface do domínio específico que a aplicação requer. Uma das questões a ser considerada quando do uso do *pattern Adapter* é a quantidade de adaptação necessária que ele deve fazer. Na classificação de [GHJV95] *Adapter* é um *design pattern* de estrutura.

O uso do *pattern Adapter* se dá quando:

- Se quer usar uma classe existente e sua interface não combina com as necessidades atuais da aplicação;
- Se quer criar uma classe reusável que coopere com classes que não tenham interfaces compatíveis;
- Se necessita usar diversas subclasses existentes, mas se torna impraticável adaptar todas as suas interfaces. Neste caso, um objeto *Adapter* pode adaptar a interface da superclasse.

O *pattern Bridge* tem uma estrutura similar ao *pattern Adapter*, mas o *Bridge* tem uma intenção diferente: ele separa uma interface de sua implementação para que elas possam ser alteradas de forma fácil e independente. Um *Adapter* se propõe a trocar a interface de um objeto existente.

b) Bridge

O *pattern Bridge* tem como finalidade separar uma abstração de sua implementação de forma que as duas possam ser modificadas independentemente [GHJV95]. Quando uma abstração pode ter uma das diversas possíveis implementações, o modo usual de acomodá-las é através do uso de herança. Uma classe abstrata define a interface para a abstração e subclasses concretas a implementam em diferentes modos. Mas este modelo nem sempre é flexível o suficiente. A herança liga uma implementação a uma abstração permanentemente, o que provoca dificuldades para se modificar, estender e reusar abstrações e implementações independentemente. O *Bridge* resolve este problema colocando a abstração e implementação em hierarquias de classes separadas (parte-de), permitindo a elas variar independentemente. Na classificação de [GHJV95] *Bridge* é um *design pattern* de estrutura.

O uso do *pattern Bridge* se dá quando:

- Se quer evitar uma ligação permanente entre uma abstração e sua implementação, por exemplo quando a implementação deve ser selecionada em tempo de execução;

- Tanto as abstrações quanto suas implementações devem ser extensíveis através de subclasses, de forma independente;
- Trocas na implementação de uma abstração não devem ter impacto sobre os clientes, ou seja, o código não deve ser recompilado;
- Se quer compartilhar uma implementação entre múltiplos objetos, mas de forma transparente para o cliente.

O uso do *Bridge* permite separar interface e implementação, permitindo que a implementação de uma abstração possa ser configurada em tempo de execução. Permite ainda aumentar a extensibilidade e trata de forma transparente detalhes de implementação de seus clientes. Já o *pattern Adapter* permite que classes não relacionadas trabalhem em conjunto e é geralmente aplicado em sistemas após serem reprojetados, enquanto o *Bridge*, por sua vez, é usado durante o projeto para permitir que abstrações e suas implementações variem independentemente.

c) Broker

O *pattern Broker* estrutura sistemas de *software* distribuídos, separando os componentes que interagem entre si através de chamada de serviços remotos [BMR+96]. Um componente *Broker* é responsável por coordenar a comunicação, incluindo o envio de requisições, a transmissão de resultados e de exceções. O *Broker* reduz a complexidade envolvida no desenvolvimento de aplicações distribuídas, fazendo esta distribuição ser transparente para o desenvolvedor. Na classificação de [BMR+96] *Broker* é um *pattern* arquitetural.

O uso de *Broker* se dá quando:

- Componentes devem ser capazes de acessar serviços fornecidos por outros, através da chamada de serviços remotos com transparência de localização;
- Se necessita trocar, adicionar ou remover componentes em tempo de execução;

- A arquitetura deve esconder detalhes do sistema e da implementação dos usuários de componentes e serviços.

Construir um sistema de *software* complexo como um conjunto de componentes separados que interoperam, resulta em maior flexibilidade e facilidade de manutenção e alteração, comparado a uma aplicação monolítica. Particionando funcionalidades em componentes independentes, o sistema torna-se potencialmente distribuído e extensível, podendo garantir portabilidade e interoperabilidade. Uma aplicação que usa um objeto deve somente ver a interface oferecida pelo objeto, não sendo necessário saber nada sobre detalhes de sua implementação ou sua localização física.

Broker é uma estrutura de larga escala, não sendo, geralmente, usado em aplicações simples, mas sim naquelas que servem como uma plataforma para um conjunto de famílias de aplicações, enviando e monitorando solicitações, sem observar quem enviou ou mesmo o conteúdo da solicitação.

d) Facade

O *pattern Facade* fornece, para um conjunto de interfaces de um sub-sistema, uma interface unificada de alto nível que facilita o uso do sub-sistema [GHJV95]. A estruturação de um sistema em sub-sistemas auxilia a redução da complexidade. Como um dos objetivos de um projeto é minimizar a comunicação e dependências entre os sub-sistemas, a introdução de *Facade* fornece uma interface única e simplificada para uma maior facilidade de comunicação com o sub-sistema. Na classificação de [GHJV95] *Facade* é um *design pattern* de estrutura.

O uso do *pattern Facade* se dá quando:

- Se quer fornecer uma interface simples para um sub-sistema complexo;
- Existem muitas dependências entre clientes e as classes de implementação de uma abstração;
- Quando os sub-sistemas estão dispostos em níveis hierárquicos diferentes.

A comunicação com os sub-sistemas é realizada através do *Facade* que transmite as requisições para o sub-sistema objeto apropriado, facilitando o uso dos sub-sistemas. Pode ser necessário que o *Facade* transforme sua interface de acordo com a interface do sub-sistema. Com o uso de *Facade*, não é necessário o acesso aos sub-sistemas objetos diretamente. *Facade* auxilia estruturar um sistema em camadas e também as dependências entre objetos, podendo também simplificar o porte de sistemas para outras plataformas.

3.7 – Síntese do Capítulo

Neste capítulo discutimos os problemas relacionados a integração de dados heterogêneos e distribuídos. Apresentamos os conceitos e técnicas existentes, como *frameworks*, componentes e *design patterns*.

No próximo capítulo é descrito o ambiente CoDIMS, que foi especificado e desenvolvido com base nos conceitos discutidos neste capítulo, bem como nas arquiteturas dos sistemas apresentadas no capítulo 2.

Capítulo 4 – O Ambiente CoDIMS

Este capítulo apresenta o ambiente CoDIMS, descrevendo seus componentes, funcionalidades e os mecanismos de configuração e de escalonamento das tarefas.

4.1 – Introdução

Integração de dados heterogêneos e distribuídos é um tema de pesquisa atual e com muitos problemas em aberto. No capítulo 2 foram apresentados alguns sistemas existentes na literatura para integração de dados. O aparente vasto número de soluções pode, erroneamente, indicar uma área de pesquisa já resolvida. Pelo contrário, é uma área cada vez mais importante e onde não existe uma solução geral que seja adequada ou que se ajuste aos diversos problemas de integração, o que se constata pelo surgimento de novas propostas. Existe uma grande variedade de problemas, soluções e usuários nesta área, havendo a necessidade de se pesquisar novas soluções, flexíveis e configuráveis, que possam ser moldadas para um contexto específico.

A grande vantagem de um sistema flexível e configurável para a integração de dados heterogêneos é a facilidade de adaptação do mesmo para ser utilizado em diversos domínios de aplicação, que exigem diferentes tipos de serviços como por exemplo:

- Multimídia: tipos de dados, estrutura de armazenamento e métodos de acesso adequados; dados contínuos; consultas não convencionais para textos e imagens, mapas e gráficos; controle de concorrência; gerência de transações;
- *Web*: integração de dados estruturados, não estruturados e semi-estruturados;
- *Data Warehouse*: materialização, materialização parcial e não materialização de visões; atualização incremental;

- *Legacy Systems* e outros aplicativos: arquivos seqüenciais e indexados, planilhas, textos etc.

A experiência e o uso de novas técnicas adquiridas com os projetos HEROS e ECOHOOD, apresentados no capítulo 1, nos permitiu evoluir a pesquisa e desenvolver o ambiente CoDIMS. Nosso objetivo é, a partir de uma arquitetura básica, poder construir sistemas de integração configurados, através de um mecanismo que permita a integração de componentes adequados, previamente selecionados e customizados, voltados para uma aplicação específica. Suas características fundamentais são flexibilidade e configuração.

A flexibilidade permite integrar diferentes fontes de dados; utilizar diferentes formas de comunicação, tanto entre os componentes internos quanto entre as fontes de dados; utilizar diferentes modelos de dados como modelo integrador e utilizar diferentes linguagens de consulta, de acordo com os requisitos da aplicação. Esta flexibilidade é fornecida através do uso da técnica de *frameworks* para o desenvolvimento dos componentes.

A configuração permite gerar sistemas de integração configurados, que contenham apenas os componentes necessários e adequados para uma aplicação específica, o que denominamos no capítulo 1 de "*what you need is only what you get" (WYNWYG)*. A configuração permite a seleção e integração de um conjunto adequado de componentes que implementam serviços *middleware* de integração de dados (DIMS). A funcionalidade de cada componente é relacionada com os serviços DIMS que ele oferece.

4.1.1 – Exemplos de Configuração

Para exemplificar o uso de diferentes configurações para diferentes requisitos, considere uma aplicação que necessite integrar dados de duas fontes de dados: uma base de dados *XML* e um banco de dados relacional.

O modelo de dados integrador mais apropriado, neste caso, seria o relacional ou até mesmo *XML*. A utilização de um outro modelo de dados integrador, por exemplo o orientado a objetos, apresentaria um custo adicional

e problemas para as conversões entre os modelos. De qualquer forma, não importando qual seja o modelo de dados escolhido, todos os componentes do sistema devem implementar serviços que sejam compatíveis com o mesmo.

Caso esta aplicação necessite apenas realizar consultas (leitura) sobre os dados integrados, seriam necessários apenas os componentes Interface, Gerência de Metadados, Processamento de Consulta e Acesso aos Dados, conforme apresentado na figura 4.1.

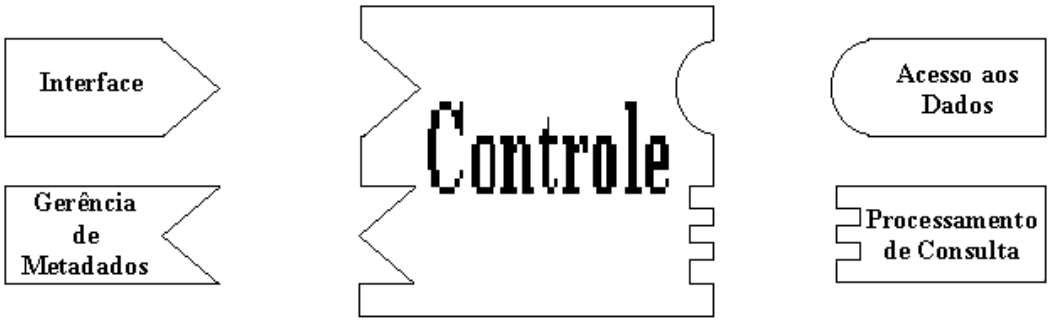

Figura 4.1 – Exemplo de Configuração

No caso da aplicação necessitar realizar alterações nos dados das fontes, torna-se necessário incluir os serviços de transação (componente Gerência de Transação). Além disso, se o sistema aceitar transações concorrentes, deve-se incluir serviços de controle de concorrência (componente Controle de Concorrência). Com a inclusão destes dois componentes, o sistema apresentaria a seguinte configuração (figura 4.2).

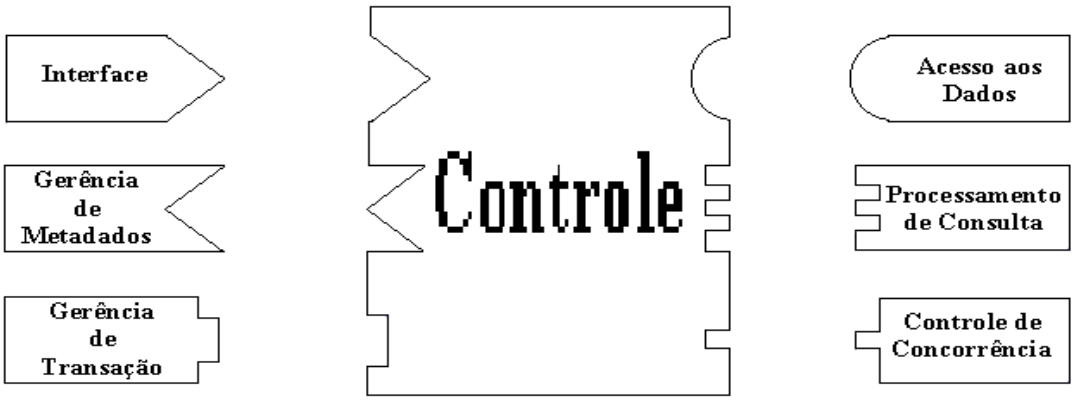

Figura 4.2 – Exemplo de Configuração

Aplicações mais complexas, como por exemplo multimídia, podem requisitar novas funcionalidades, que devem ser incorporadas quando do processo de configuração, conforme figura 4.3.

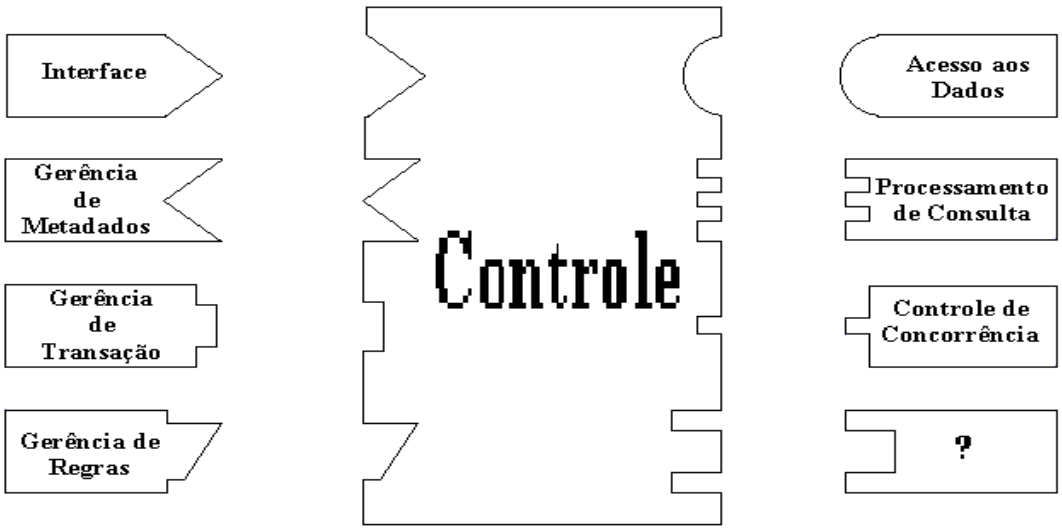

Figura 4.3 – CoDIMS: um ambiente configurável baseado em componentes

4.2 – Visão Geral do Ambiente

O ambiente CoDIMS apresenta uma nova abordagem para se gerar sistemas configurados para integração de dados, através da utilização das técnicas de *frameworks* e de componentes. A essência do ambiente é um componente de Controle, que permite definir e validar a configuração estática e gerenciar a invocação dos serviços durante a execução do sistema configurado. Como os serviços oferecidos podem diferir de uma configuração para outra, torna-se necessário definir quais serviços e em que ordem deverão ser executados em cada configuração. O escalonamento dos serviços é modelado através de um mecanismo baseado no conceito de *workflow* [JB96] [Koib97] [Lawr97]. Na realidade, um mesmo sistema configurado pode ter diferentes escalonamentos, para atender às diferentes requisições da mesma aplicação.

A abordagem proposta traz inovações através dos mecanismos de configuração lógica e física, permitindo construir sistemas configurados apenas com os serviços necessários e adequados à aplicação. O mecanismo de configuração física determina quais são os serviços oferecidos e quais os que são necessários para cada componente. O mecanismo de configuração lógica,

através do *workflow*, determina os serviços e a ordem de execução destes para cada tarefa solicitada.

4.2.1 – Configuração: Visão Conceitual

A grande contribuição dos SGBDs, principalmente os relacionais, foi extrair os metadados dos programas, criando um nível de abstração maior: o esquema. A partir dele, foi possível ter a separação física e lógica dos dados que estavam nos programas e dispor de linguagens de consulta de mais alto nível.

O mesmo benefício obtido na abstração de dados, expresso pelo esquema em um banco de dados, é proposto neste trabalho para abstrair a lógica de comunicação entre os serviços internos do sistema de integração, através de uma representação de alto nível, baseada no conceito de *workflow*. Tal representação é denominada de configuração lógica do sistema. Por outro lado, a configuração física permite selecionar e integrar os serviços adequados a uma aplicação.

Consegue-se, assim, ter uma separação física e lógica dos serviços oferecidos por um sistema: a ligação física entre os componentes através do mecanismo de configuração física, e a ligação lógica através da configuração lógica, modelada como um *workflow*. Estes mecanismos de configuração permitem separar as dependências diretas entre os componentes, oferecendo um modo mais flexível para a seleção e a integração destes para a criação de um sistema configurado.

Esta abordagem apresenta diversas vantagens, pois permite:

- O reuso, adaptação, troca e integração de componentes já existentes;
- O uso de diferentes serviços para aplicações específicas, como: linguagem de consulta, modelo de dados, técnicas de otimização de consultas etc.;
- Escalonamento dinâmico dos serviços a serem executados, de acordo com as pré e pós-condições estabelecidas no *workflow*.

4.2.2 – Configuração: Visão Operacional

Uma vez definidos e selecionados os componentes que farão parte do sistema a ser configurado, estes devem ser customizados de acordo com o ambiente

computacional, com os requisitos da aplicação e com as características das fontes de dados que se deseja integrar. A atividade de customização pode ser tão fácil quanto apenas escolher componentes *black-box* ou tão difícil quando for necessário implementar um novo componente. Visando facilitar e diminuir o tempo a ser gasto no processo de customização dos componentes do CoDIMS, estes foram especificados utilizando-se a técnica de *frameworks*, o que possibilita reuso e uma melhor qualidade do *software*.

Após a seleção e customização dos componentes, os mesmos devem ser registrados para compor a configuração do sistema. O processo de configuração é detalhado na seção 4.8.

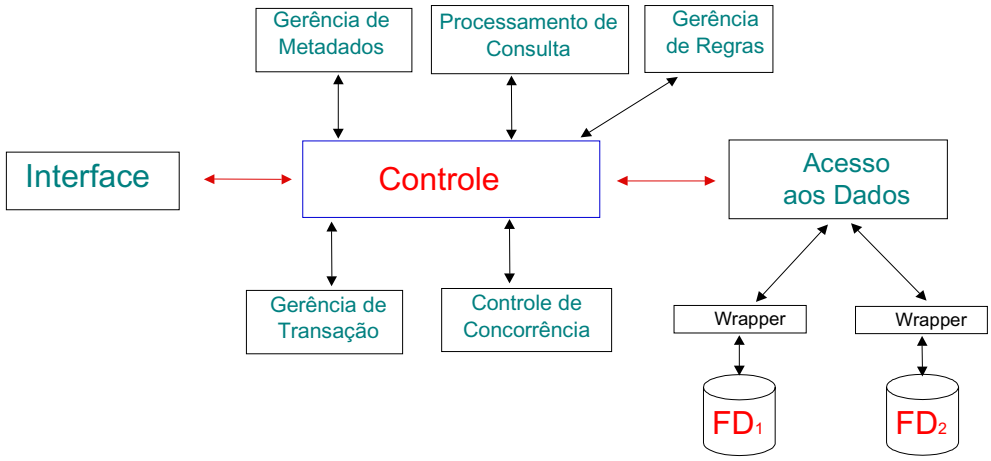

Figura 4.4 – O CoDIMS e seus componentes

A figura 4.4 apresenta uma visão geral do ambiente CoDIMS. Nas seções seguintes são descritos cada um de seus componentes.

4.3 – O Componente Controle

O componente Controle é a essência do ambiente CoDIMS por disponibilizar os mecanismos que implementam as configurações física e lógica, conforme descritas anteriormente.

O componente Controle tem como principais funções:

- Registrar a configuração do sistema, através da qual são definidas as configurações física e lógica;

- Garantir a consistência da configuração, verificando se todos os serviços a serem solicitados por um componente bem como aqueles definidos no *workflow* foram oferecidos por outro componente;
- Escalonar e gerenciar os serviços a serem executados pelo sistema, quando for submetida uma requisição pelo usuário;
- Repassar as mensagens recebidas de outros componentes para o componente destino, agindo como um *broker*;
- Receber os comandos submetidos pela aplicação cliente, através de uma *API*.

Na seção 4.8 o processo de configuração física e lógica é descrito com mais detalhes, enquanto que na seção 5.3 é apresentada a especificação do componente Controle, com uma descrição mais detalhada das suas operações.

4.4 – O Componente Gerência de Metadados

O componente Gerência de Metadados é responsável por armazenar, gerenciar e viabilizar o acesso às meta-informações do sistema integrador. A figura 4.5 mostra os quatro níveis dos metadados do CoDIMS: metadados de cada fonte de dados, metadados de exportação, metadados global e metadados externo (visões externas). O componente Gerência de Metadados implementa as mesmas funcionalidades de integração de esquemas de SGBDHs, conforme apresentado na seção 3.2.1.

4.4.1 – Metadados das Fontes de Dados

Cada fonte de dados possui seus próprios metadados, denominado de Metadados-FD. Estes metadados são estruturados de acordo com o tipo da fonte de dados e a forma de armazenamento e de organização de seus dados.

Para possibilitar a integração dos dados de uma fonte, os seus metadados têm de ser definidos no sistema integrador. Pode ocorrer de apenas parte de seus dados serem liberados para acesso, o que pode resultar em disponibilizar parte de seus metadados.

Sendo os metadados das fontes de dados a serem integradas heterogêneos, antes do processo de integração é necessário homogeneizar estes metadados, o que é possível através dos metadados de exportação.

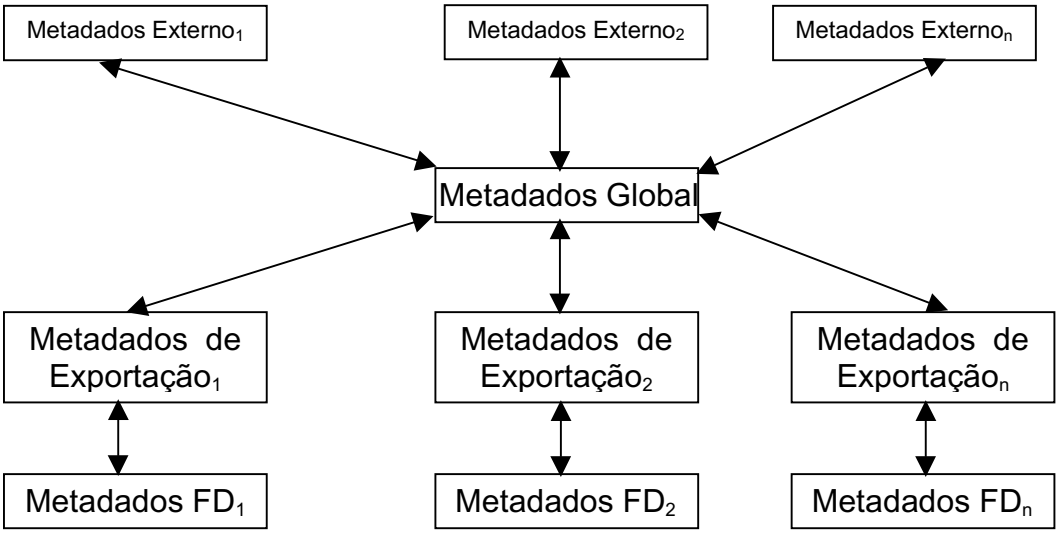

Figura 4.5 - Arquitetura de Metadados em 4 níveis

4.4.2 – Metadados de Exportação

Para homogeneizar os metadados das fontes de dados, cada um destes metadados-FD é mapeado em um respectivo Metadados de Exportação, descrito no modelo de dados integrador usado pelo sistema configurado. Neste sentido, um metadados de exportação corresponde a uma visão dos metadados pertencentes a uma fonte de dados. Cada um destes metadados de exportação será armazenado no componente Metadados. Esta informação será necessária para se produzir a sub-consulta para acesso aos dados da fonte específica.

Esta transformação dos metadados-FD para um mesmo modelo de dados, permite ao sistema ter uma visão homogênea dos dados, apesar dos diferentes tipos de dados existentes em cada uma das fontes de dados. Assim, para o sistema integrador, é como se todas as fontes de dados tivessem o mesmo modelo de dados, o que possibilita a utilização de uma mesma linguagem de consulta. No caso da fonte de dados não possuir metadados, deverá ser criada uma representação para seus metadados de exportação, de

acordo com os requisitos da aplicação e a forma de acesso aos dados a ser realizada.

4.4.3 – Metadados Global

A integração de todos os metadados de exportação forma o que denominamos de metadados global. Metadados global representa uma visão homogênea de todos os dados integrados: para a aplicação cliente é como se somente uma única fonte de dados estivesse disponível. Também neste caso, existem os mapeamentos entre os Metadados Globais e os Metadados de Exportação, que serão utilizados quando da otimização e execução de consultas globais, dividindo-as em sub-consultas a serem direcionadas a cada fonte de dados.

Os Metadados Globais devem atender aos requisitos de cada uma das aplicações cliente, através dos metadados externos (visão externa). Cada uma destas visões representa um sub-conjunto dos metadados globais.

O processo de integração dos metadados de exportação em metadados globais é um tema aberto de pesquisa, onde existem diversos problemas e propostas de solução, conforme descrito na seção 3.2.1 deste trabalho.

4.5 – O Componente Processamento de Consulta

O Componente Processamento de Consulta é responsável por transformar uma consulta escrita em uma linguagem de alto nível, disponibilizada pelo sistema integrador, em uma estratégia eficiente de execução, considerando as características das fontes de dados. A consulta de entrada pode apresentar uma grande complexidade, que depende das facilidades fornecidas pela linguagem. Esta consulta deve ser analisada e decomposta em partes para facilitar e otimizar a sua execução.

Por ser um problema crítico para a performance, as pesquisas sobre o processamento de consulta tem recebido considerável atenção durante os últimos anos [ÖV99] [YM98]. Este problema é maior no caso de distribuição e heterogeneidade, devido a um grande número de parâmetros que podem afetar a performance durante a execução de consultas distribuídas.

O processamento de consulta deve interpretar e analisar a consulta global recebida, produzir as sub-consultas que serão submetidas a cada uma das fontes de dados, gerar um plano de execução global otimizado e proceder a execução da consulta e a composição dos resultados.

Por efetuar várias tarefas específicas, o serviço de processamento de consulta foi subdividido, para efeito deste trabalho, em quatro módulos: analisador, re-escritor, otimizador e processador (figura 4.6). De acordo com as funcionalidades requeridas e a sua especificação, o componente processamento de consulta poderá ter vários outros módulos, como apresentado em [ÖV99], [YM98] e [Moer99]. Como o enfoque deste trabalho não é em processamento de consulta, somente serão descritas as funcionalidades básicas dos módulos analisador, re-escritor, otimizador e processador. Na seção 5.6, é apresentada a especificação do componente Processamento de Consulta. No capítulo 6 são apresentados em mais detalhes todas as etapas necessárias ao processamento de uma consulta.

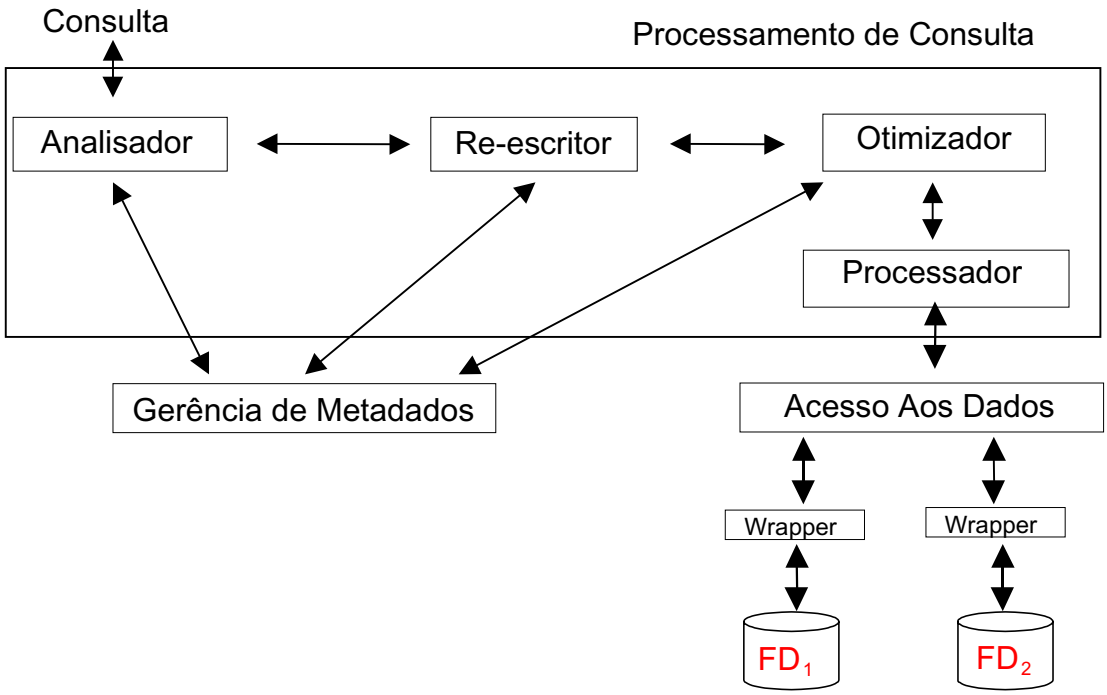

Figura 4.6 – Processamento de Consulta

4.5.1 – Analisador

Responsável pela análise léxica e sintática da consulta global submetida, de acordo com as informações contidas nos metadados, existindo a possibilidade de rejeição da consulta no caso de erros de sintaxe, de semântica ou de tipos incorretos. Quando acontece a rejeição, a consulta é retornada à aplicação cliente, com o respectivo motivo da rejeição (código de retorno).

No caso de *SQL*, a rejeição por tipos incorretos ocorre quando atributos ou nomes de relações não estão definidos nos metadados ou se operações estão sendo aplicadas a atributos de tipo incompatível. Uma consulta é semanticamente incorreta se ela não apresenta todas as informações para a geração de um resultado.

No caso de se adotar o modelo relacional como modelo integrador, geralmente a consulta submetida é transformada em um grafo de consulta global contendo um conjunto de operações de seleção, projeção e junção. No grafo, um dos nós é o de resultado e os outros, indicam as relações globais. Uma aresta entre dois nós (exceto o nó resultado) representa uma junção, enquanto que uma aresta ao nó resultado indica projeção [ÖV99].

O grafo de consulta é útil para determinar a corretude da consulta: se o grafo é não conectado, a causa é uma ausência de um predicado de junção (cláusula *Where* em *SQL*) na consulta e, neste caso, a consulta deve ser rejeitada. Na seção 6.5.1, a figura 6.7 apresenta um exemplo de um grafo de consulta, para o estudo de caso deste trabalho.

4.5.2 – Re-escritor

A partir do grafo de consulta global e com base nas informações dos metadados globais e dos metadados de exportação das fontes de dados envolvidas na consulta, o componente re-escritor, como o próprio nome diz, re-escreve a consulta submetida, agrupando as informações que devem ser fornecidas por cada fonte de dados. O grafo de consulta global é transformado em uma árvore de operação onde cada nó folha representa uma fonte de dados. O nó raiz representa o resultado da consulta e os outros nós não folhas representam os resultados intermediários.

O objetivo do re-escritor é agrupar as sub-consultas por fonte de dados de maneira que uma, e apenas uma, sub-consulta, seja enviada a cada fonte de dados para cada consulta global submetida. Na seção 6.5.2, a figura 6.8 apresenta um exemplo de uma árvore de operação.

4.5.3 – Otimizador

A partir da árvore de operação gerada pelo componente re-escritor, o componente otimizador produz um plano de execução otimizado para a consulta. A permutação da ordem das operações dentro da árvore de operação pode fornecer muitas estratégias de execução, umas aceitáveis e outras inviáveis. O papel do otimizador de consulta é encontrar a ordenação ótima destas operações, o que nem sempre é possível, pois de acordo com a consulta e o número de fontes de dados envolvidas, poderiam levar a um problema do tipo N-P. Assim, o objetivo do otimizador é encontrar uma boa estratégia ou, até mais importante que isso, evitar as péssimas estratégias.

A seleção de uma estratégia requer um cálculo estimado dos custos de execução entre as ordenações previamente definidas como candidatas. O plano selecionado minimiza a função custo. Os planos são equivalentes, no sentido que os mesmos resultados são obtidos, mas diferem na ordem de execução das operações e no modo que estas operações são definidas, visando atingir melhor performance. O modelo de custo estima o custo de um dado plano de execução e para isso deve ter bom conhecimento sobre o ambiente, como: comunicação e estatísticas sobre os dados das fontes de dados, tamanho estimado dos resultados intermediários etc.

O plano de execução otimizado para a consulta recebe o nome de árvore de execução. Na seção 6.5.3, a figura 6.9 apresenta um exemplo de uma árvore de execução.

4.5.4 – Processador

Tem como função encaminhar cada sub-consulta existente no plano de execução ao componente Acesso aos Dados, receber os resultados destas

sub-consultas, realizar a integração destes resultados e fornecer o resultado final da consulta à aplicação cliente.

No capítulo 6 é apresentado um estudo de caso onde todas estas etapas são ilustradas.

4.6 – O Componente Acesso aos Dados

O componente Acesso aos Dados é responsável pela comunicação com as fontes de dados a serem integradas. Para permitir flexibilidade no acesso às fontes de dados existentes em diferentes ambientes e permitir o uso de diversas tecnologias, o componente Acesso aos Dados deve permitir diferentes formas de comunicação, como *JDBC*/*ODBC, ORB, RPC, RMI, HTTP* etc. Para utilização de *RPC, ORB ou RMI*, a fonte de dados deve ser encapsulada e viabilizar uma interface compatível para permitir o acesso.

Para cada fonte de dados definida, deve haver um *wrapper* específico declarado junto com os metadados. O *wrapper* traduz a sub-consulta, que está na linguagem utilizada pelo sistema configurado, para a linguagem de acesso à fonte de dados. Devido ao ambiente ser heterogêneo, as linguagens de cada fonte de dados poderão ser diferentes entre si.

Ao receber uma solicitação para executar uma sub-consulta, o *wrapper* realiza os procedimentos necessários, de acordo com as características da fonte de dados, tais como: tipo (banco de dados, arquivo texto, página *HTML*…), forma de comunicação, linguagem etc. A complexidade do *wrapper* está relacionada à dificuldade de tradução da linguagem do sistema configurado para a linguagem da fonte de dados e a forma de acesso a ela.

4.7 – Os Componentes Gerência de Transação, Controle de Concorrência e Gerência de Regras

O contexto deste trabalho é permitir configurar um sistema integrador que permita a consulta (leitura) de diversas fontes de dados integradas. Assim, os componentes Transação, Controle de Concorrência e Regra, apesar de constantes no *framework*, não serão detalhados, devendo ser melhor

estudados e especificados em outros trabalhos dirigidos à este fim, considerando a complexidade e especificidade de tais linhas de pesquisa.

4.7.1 - Gerência de Transação

O componente Gerência de Transação é responsável pela gerência e garantia das propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade) quando da atualização dos dados (escrita) das fontes de dados. As funcionalidades básicas deste componente incluem procedimentos para delimitar o início de uma nova transação e o final de uma transação, através de um término anormal (*abort*) ou término com sucesso (*commit*).

Diversos são os problemas encontrados para a utilização do conceito de transação em sistemas integradores de dados. Um estudo sobre estes problemas e uma proposta de solução encontra-se em [Silv94].

4.7.2 - Controle de Concorrência

O componente Controle de Concorrência provê um mecanismo de controle quando da ocorrência de pelo menos um pedido de atualização sobre um mesmo dado entre transações concorrentes. O objetivo é evitar erros de consistência quando da execução de transações concorrentes. Diversos são os protocolos propostos para efetivação do controle de concorrência em SGBDs, sendo o de bloqueio o mais utilizado. Neste caso as suas funcionalidades básicas são poder bloquear um objeto para que o mesmo não seja utilizado por uma outra transação e proceder seu desbloqueio para que o mesmo tenha seu uso liberado.

Em sistemas integradores, devido à autonomia das fontes de dados, implementar este controle é bem complexo. Controle de concorrência está diretamente relacionado ao conceito de transação. Mas em aplicações onde não ocorre a execução de transações concorrentes, o componente Controle de Concorrência não precisa estar presente no sistema a ser configurado.

4.7.3 – Gerência de Regras

A presença do componente Gerência de Regras enriquece o sistema integrador com um comportamento ativo, isto é, ter a capacidade de reagir automaticamente à ocorrência de determinados eventos, tal como proposto nos sistemas de bancos de dados ativos [WC96]. Para isso, é necessário definir situações que, uma vez reconhecidas pelo sistema, disparam automaticamente a execução de ações pré-determinadas.

As funcionalidades básicas deste componente são: a definição das regras, determinação das situações e das respectivas ações a serem executadas, e a própria execução da regra quando da ocorrência de tais situações.

4.8 – O Processo de Configuração

Para o processo de configuração, customização e modelagem de um sistema de integração de dados, são necessárias as seguintes etapas:

ETAPA 1 – Projeto do Sistema

Nesta etapa são selecionados e definidos os componentes adequados que deverão fazer parte do sistema a ser configurado. Eles são customizados de acordo com o ambiente operacional e os requisitos da aplicação.

No processo de seleção, definição e customização dos componentes, deve ser considerada a compatibilidade entre os componentes no que se refere não somente à interface, mas, principalmente, com relação ao modelo de dados e a linguagem a serem utilizados no sistema configurado. Também devem ser verificadas e resolvidas as incompatibilidades, que podem repercutir nos outros componentes envolvidos. Este procedimento é manual e realizado pelo projetista do sistema, não existindo uma solução para a verificação automática nestes casos.

ETAPA 2 – Configuração Física

Nesta etapa são registrados, no sistema a ser configurado (no componente Controle), todos os componentes que farão parte da configuração, incluindo

para cada componente, uma lista das operações que ele disponibiliza (implementa) e outra lista das operações que serão por ele requisitadas. Esta atividade é realizada pelo configurador do sistema.

Deve ser editado um arquivo de *script* contendo todas as informações necessárias, como: o nome do componente, as operações fornecidas e as solicitadas e, para cada operação, seu nome e os tipos dos parâmetros. A partir deste arquivo, será definida a configuração do sistema. A descrição do formato do arquivo de *script* é apresentada na seção 5.3.

Após a definição dos componentes, o componente Controle automaticamente verifica a consistência da configuração, para garantir que todas as operações a serem requisitadas por um componente estejam sendo disponibilizadas por outro componente. Assim, para cada operação a ser requisitada por um componente, o sistema consulta a definição do componente fornecedor na configuração, para verificar se a operação foi declarada.

ETAPA 3 – Configuração Lógica: Definição do *Workflow*

Para cada comando da linguagem a ser utilizada, será definido um *workflow* para estabelecer a ordem de execução das tarefas necessárias para permitir o processamento do comando submetido.

A etapa de definição do *workflow* é análoga a definição da configuração, sendo também realizada pelo configurador do sistema. Um arquivo de *script* deve ser gerado contendo a seqüência das operações a serem executadas e, para cada uma delas, indicando a que componente pertence. Durante a operação do sistema, o *workflow* para o comando submetido será então recuperado para que o mesmo posso ser executado. A descrição do formato do arquivo de *script* é apresentada na seção 5.3.

Após a definição do *workflow* para todos os comandos da linguagem a ser utilizada, o componente Controle automaticamente verifica a consistência do *workflow* para garantir que todos os serviços (operações dos componentes) a serem executados foram declarados, quando da definição da configuração dos componentes do sistema.

ETAPA 4 – Carga dos Metadados

Concluído o processo de configuração, a próxima atividade é a definição dos metadados do sistema. Esta atividade é realizada pelo projetista da aplicação. Um arquivo de *script* deve ser gerado contendo as definições dos metadados, através de uma Linguagem de Definição de Dados (*DDL*). O detalhamento para o procedimento de carga dos metadados é apresentado na seção 5.5.

A partir deste ponto, o sistema já está pronto para receber as consultas para processamento, submetidas pela aplicação cliente.

ETAPA 5 – Execução de Consultas

Quando uma consulta é submetida para execução, o Controle identifica o comando, consulta o *workflow* e gera o escalonamento inicial dos serviços a serem executados para o respectivo comando. Este escalonamento pode ser alterado dinamicamente, de acordo com o *workflow* e as condições encontradas durante a execução dos serviços. Após a execução de todas as operações, o resultado final da consulta é repassado à aplicação cliente.

4.9 – Síntese do Capítulo

Neste capítulo foi apresentado o ambiente CoDIMS, seus componentes, os mecanismos de configuração lógica e física.

No capítulo seguinte é apresentada a especificação do CoDIMs, através da linguagem *UML* e do uso de uma notação estendida. São ainda detalhados os processos para as definições das configurações física e lógica.

Capítulo 5 – Especificação do CoDIMS

Este capítulo apresenta a especificação do CoDIMS, através da linguagem UML e de uma extensão desta linguagem para a representação de frameworks.

5.1 – Introdução

No capítulo anterior foi apresentada uma descrição detalhada dos componentes e das funcionalidades do CoDIMS. Neste capítulo será apresentada a sua especificação através da *UML (Unified Modeling Language*) [BRJ00] e de uma notação estendida da *UML*, proposta em [Font99], para a representação dos pontos de flexibilização dos *frameworks*.

Para a especificação serão utilizados os Diagramas de Componentes, Diagramas de Classes, Diagramas de Seqüência e de Casos de Uso. Serão utilizados alguns *Design Patterns* conforme descritos no capítulo 3.

Os recentes esforços para padronização da *UML* ofereceram uma motivação para que ela fosse adotada como uma notação básica para os projetos de desenvolvimento de *frameworks* [Font99]. A *UML* é uma linguagem de objetivo geral com muitos construtores notacionais, entretanto o seu padrão atual não fornece construtores apropriados para modelar *frameworks* [FPR00]. Uma linguagem para projeto de *frameworks* tem de atender a alguns requisitos que não são diretamente oferecidos pela *UML* [Font99]. Os desenvolvedores de *frameworks* devem fornecer uma documentação que descreva quais partes do sistema devem ser alteradas (*hot-spots*) para a criação de instâncias válidas, evitando que usuário tenha de percorrer todo o código do *framework* - geralmente formado por grandes e complexas hierarquias - para escrever o código adequado.

Existem três tipos de *hot-spots* em *frameworks* orientados a objetos: métodos de variação, classes de extensão e interfaces. Métodos de variação são métodos que têm assinatura bem definida, mas sua implementação pode variar dependendo da instanciação do sistema. Classes de extensão são

classes que podem ter suas interfaces estendidas durante o processo de instanciação do *framework*. *Hot-spots* de interface são interfaces ou classes abstratas que permitem a criação de sub-classes concretas durante a instanciação do *framework*. A instanciação deste último tipo de *hot-spot* acontece através da criação de novas classes, as classes de instância que existem somente nas instâncias do *framework*. Todos estes três tipos de *hot-spots* podem ser estáticos ou dinâmicos (reconfigurados em tempo de execução.

Para atender a esta necessidade foram então propostos novos elementos a serem incorporados à *UML*, mas somente serão aqui apresentados aqueles utilizados nos diagramas deste trabalho. São eles:

- **Instance Class**: Se aplica a classes que existem somente na instância do *framework*, ou seja para as novas classes definidas no processo de instanciação;

- **Variable**: Se aplica a um método quando sua implementação depende da instanciação. É usado para identificar métodos de variação;

- **Extensible**: Se aplica a classe. A interface da classe depende da instanciação: novos métodos podem ser definidos para estender a funcionalidade da classe. É usado para identificar classes de extensão;

- **Static**: Se aplica a *hot-spot* de interface, método de variação e classe de extensão. O *hot-spot* não precisa ser reconfigurado em tempo de execução;

- **Incomplete**: Se aplica a generalização. Novas sub-classes podem ser adicionadas durante a instanciação do *framework*;

- **Disjoint**: Se aplica a generalização. Apenas uma das sub-classes pode estar presente na instanciação do *framework*.

5.2 – Os Componentes do CoDIMS

Conforme descrito no capítulo 1, o objetivo desta tese é apresentar uma nova abordagem para a construção de sistemas de integração de dados, através da seleção, customização e integração de um conjunto adequado de componentes, baseados nos serviços de gerência de banco de dados. Esta abordagem permite gerar sistemas *middleware* configurados específicos e enxutos para atender aos requisitos das aplicações.

A figura 5.1 apresenta o diagrama de componentes do CoDIMS, ressaltando que a comunicação entre eles é realizada pelo componente Controle. Assim, o controle também tem a funcionalidade de um *broker* [BMR+96].

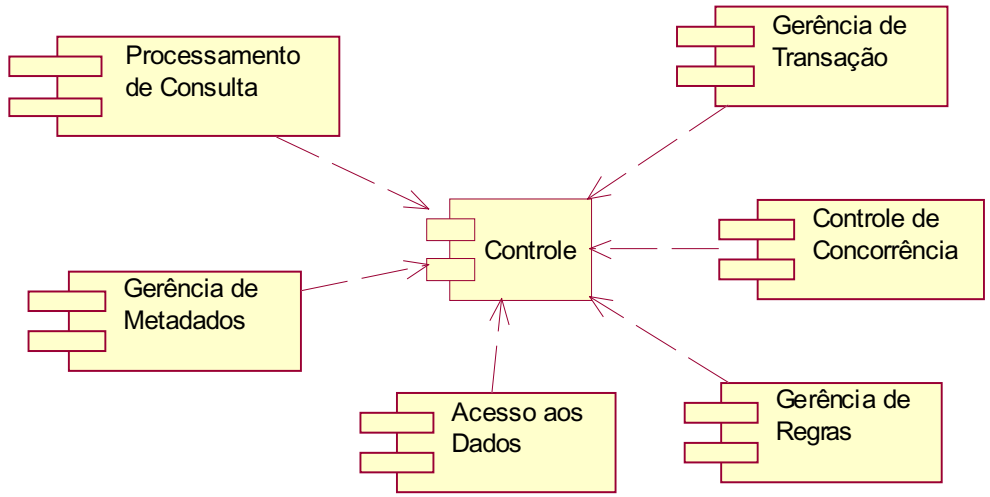

Figura 5.1 – Diagrama de Componentes do CoDIMS

Para cada componente foi definida uma fachada (figura 5.2), a partir da utilização do *design pattern Facade* [GHJV95], que apresenta a interface pública de cada um deles. Assim, toda e qualquer chamada a operações dos componentes será sempre realizada via fachada. Isto isola as funcionalidades internas dos componentes, facilitando a troca destes por outros que apresentem a mesma interface.

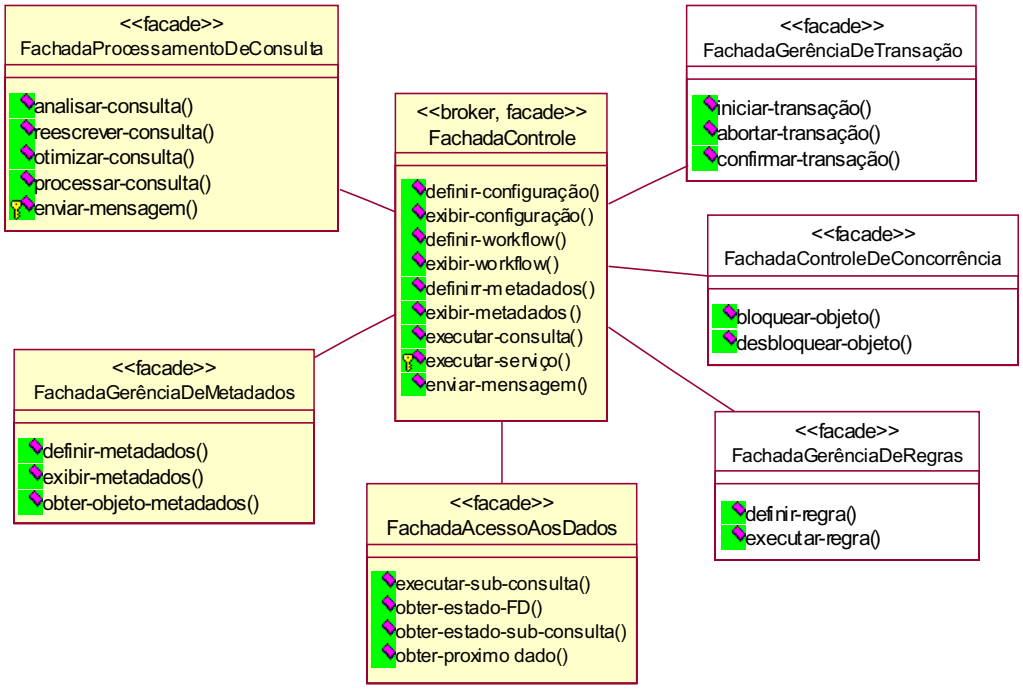

Figura 5.2 – As Fachadas dos Componentes do CoDIMS

Cada um destes componentes será especificado, a partir da próxima seção, enquanto que as operações estão descritas Anexo A.

5.3 – O Componente Controle

A figura 5.3 apresenta o diagrama de classes do componente Controle. As funcionalidades deste componente foram descritas no capítulo anterior.

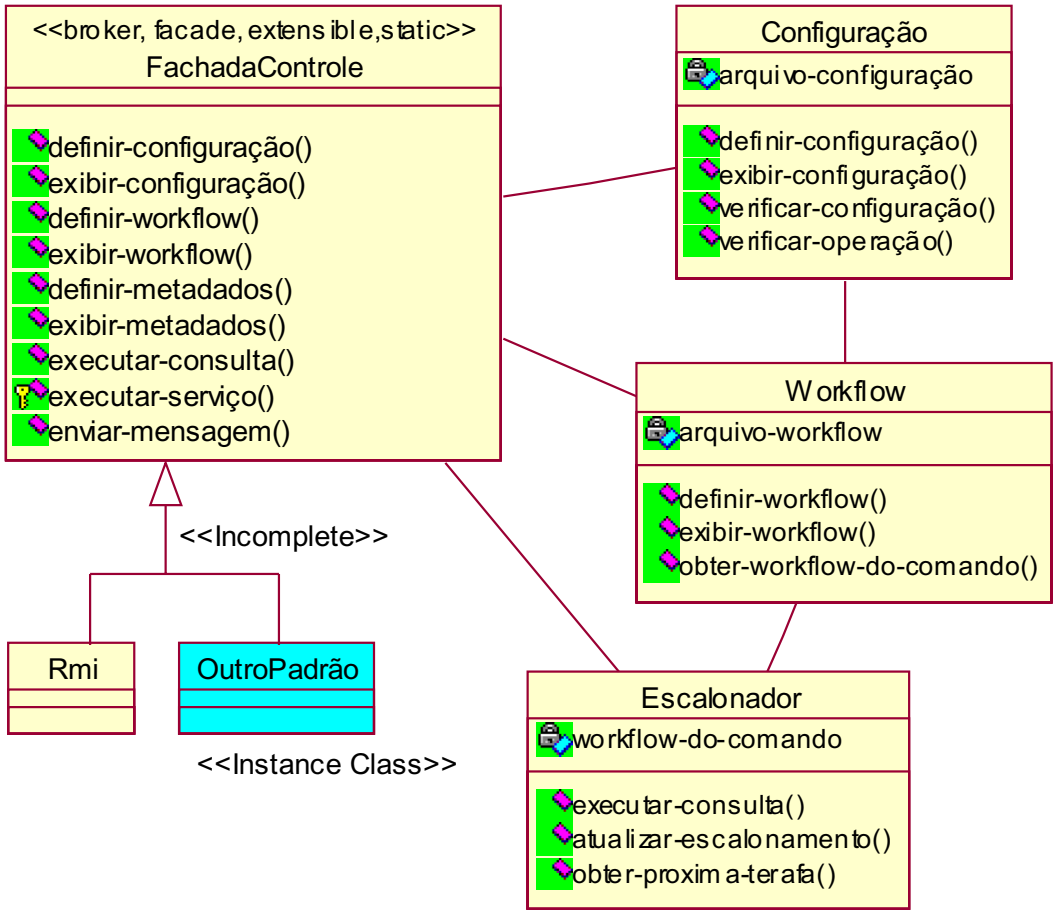

Figura 5.3 – Diagrama de classes do componente Controle

As chamadas externas às operações internas do componente Controle serão via sua fachada, de forma que as possíveis alterações nas suas funcionalidades internas, não precisam ser, necessariamente, percebidas pelos outros componentes. A FachadaControle foi definida como *static* pois a mesma não será reconfigurada após gerado o sistema configurado; e *extensible* pois pode ser necessária sua extensão para atender a uma necessidade de um sistema configurado especifico, neste caso com a definição de novas operações. Esta é uma das flexibilidades do *framework*.

A classe Configuração é responsável pelo registro e gerência da configuração física do sistema. Para a configuração deve ser editado um arquivo de *script* contendo o nome de cada componente que fará parte da configuração e, para cada um deles, devem ser registradas as operações fornecidas (*Offered-Operations)* e as que serão solicitadas (*Requested-Operations)*. Para cada operação deve ser fornecido seu nome e os seus parâmetros (tipo e nome). A partir deste arquivo será definida a configuração do sistema, quando da execução da operação ***definir-configuração***. O formato do arquivo é mostrado abaixo, devendo ter tantas ocorrências quantos forem os números de componentes.

Define Component *nome-do-componente*

Offered-Operations

nome-da-operação (tipo nome {;tipo nome})

{; nome-da-operação (tipo nome {;tipo nome})};

[Requested-Operations

nome-do-componente, nome-da-operação (tipo nome

{;tipo nome})

{;nome-do-componente, nome-da-operação (tipo nome

*{;tipo nome})};**]***

End-Component.

O formato é baseado na notação *BNF* (*Backus-Naur Form*) [Sebe96]. As informações entre chaves { } significam repetições, sejam quantas for. Já as informações entre colchetes [ ] são opcionais. No caso acima, um componente pode ter várias operações oferecidas e requisitadas, e para cada uma delas será indicado o tipo de seus parâmetros, para verificação de tipagem. Um componente pode não necessitar de operações de outros componentes, por isso *Requested-Operations* foi definido como opcional.

Após a definição dos componentes deve ser verificada a consistência da configuração para garantir que todas as operações a serem requisitadas por um componente estejam sendo disponibilizadas por outro componente. Esta

verificação é realizada através da execução da operação ***verificar-configuração***.

A classe *Workflow* é responsável pela configuração lógica do sistema configurado. Ela registra, para cada comando da linguagem utilizada pelo sistema configurado, a seqüência de operações necessárias para a completa execução do comando submetido. Esta seqüência de operações é denominada *workflow* do comando. A partir dele será definido o escalonamento dos serviços durante a execução do sistema.

A etapa de definição do *workflow* é análoga a definição da configuração física: também ocorre através de um arquivo de *scrip*t. A partir deste arquivo será definido um *workflow* para cada um dos comandos, quando da execução da operação ***definir-workflow***. O formato do arquivo é mostrado abaixo, devendo ter tantas ocorrências quanto for o número de comandos da linguagem. A ordem de definição das operações corresponde à sua ordem de execução. A estrutura de definição do *workflow* foi baseada na Linguagem de Modelagem de *Workflow MSL (Mobile Script Language*), descrita em [JB96], também em formato *BNF*.

Define Workflow nome-do-comando

Operations nome-do-componente (nome-da-operação)

{ ***; nome-do-componente (nome-da-operação)*** **}**

End-Operations.

A classe Escalonador recebe as consultas submetidas ao sistema e gerencia a execução. O comando especificado na consulta é identificado e o *workflow* do referido comando é recuperado a partir da operação ***obter-workflow-do-comando*** para permitir o escalonamento das tarefas a serem executadas. As operações são chamadas para execução na seqüência estabelecida, através da operação ***executar-serviço*** da FachadaControle. Ao final da execução de todas as tarefas, o resultado é repassado à aplicação cliente.

A Classe Comunicação tem como função realizar a comunicação do componente Controle com todos os outros componentes. Para permitir flexibilidade, diferentes formas de comunicação podem ser adotadas, como *RMI*, *CORBA* e mensagens, sendo que uma delas deve ser definida na etapa de instanciação do *framework.* Por isso a utilização da restrição *unique*.

5.4 – O Componente Acesso aos Dados

O componente Acesso aos Dados, figura 5.4, é responsável pela comunicação com as fontes de dados a serem integradas, sendo que as funcionalidades deste componente foram descritas no capítulo anterior.

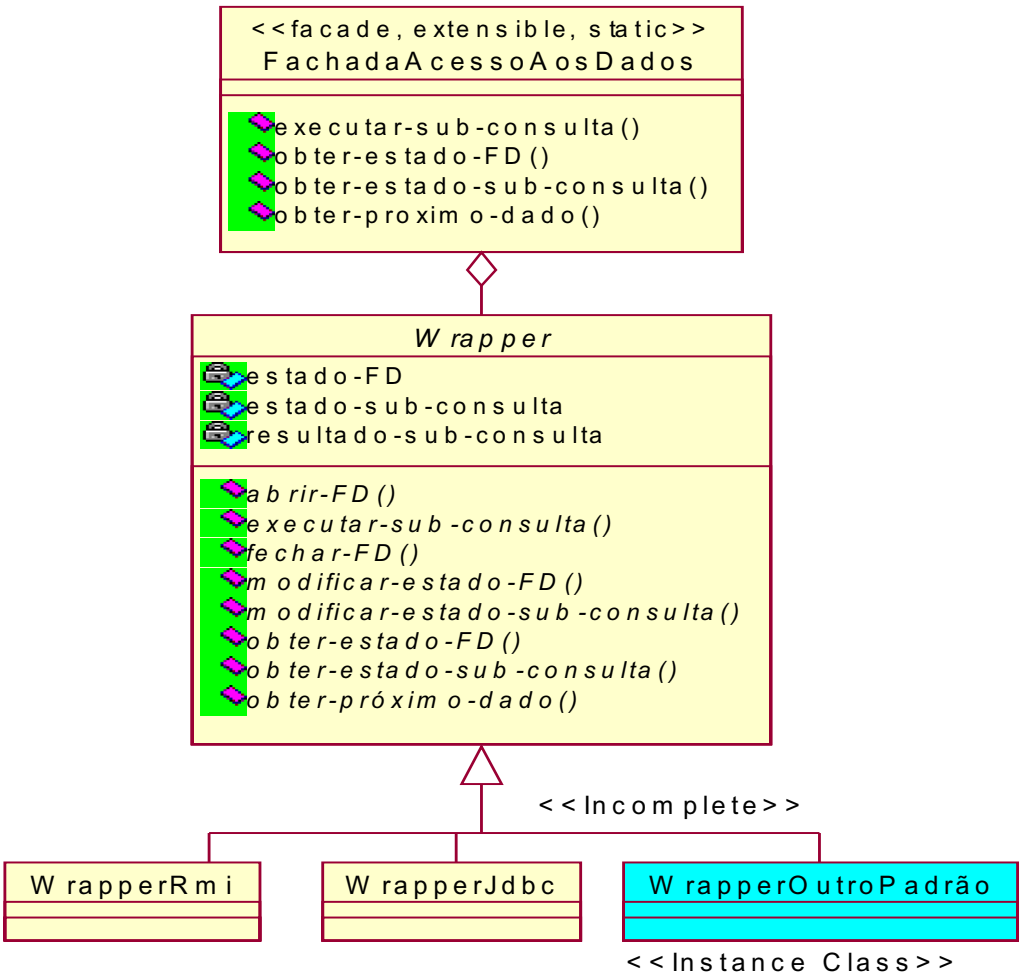

Figura 5.4 – O componente Acesso aos Dados

As chamadas externas serão feitas através da fachada, de forma a solar as funcionalidades internas. A FachadaAcessoAosDados foi definida como *static* pois não será reconfigurada após a criação do sistema; e *extensible* pois pode ser necessária sua extensão para atender a um sistema configurado especifico, neste caso com a definição de novas operações.

Para permitir flexibilidade no acesso às fontes de dados existentes em diversos ambientes, diferentes formas de comunicação podem ser adotadas como: *JDBC*/*ODBC*, *CORBA*, *RPC*, *RMI*, *HTTP* etc. Para utilização de *RPC*, *CORBA* ou *RMI*, a fonte de dados deve ser encapsulada e ser viabilizada uma interface compatível para proceder o acesso através destas tecnologias. A especificação *incomplete* no diagrama se refere a utilização de novas formas de comunicação, além das definidas.

Para cada fonte de dados a ser integrada pelo sistema deve haver um *wrapper* específico, declarado junto com os metadados. O *wrapper* traduz a requisição de acesso aos dados, da linguagem utilizada pelo sistema configurado, na linguagem de acesso à fonte de dados e executa o acesso solicitado. Devido ao ambiente heterogêneo, as linguagens de cada fonte de dados poderão ser diferentes entre si, o que torna necessário a presença do *wrapper* e o processo de tradução.

Ao receber uma solicitação para executar uma sub-consulta, o *wrapper* realiza os procedimentos necessários, de acordo com as características da fonte de dados, como: tipo (banco de dados, arquivo texto, página *HTML*…), forma de comunicação, linguagem etc. A complexidade do *wrapper* está relacionada à dificuldade de tradução da linguagem do sistema configurado para a linguagem da fonte de dados e a forma de acesso a ela.

5.5 – O Componente Metadados

O componente Metadados é responsável por armazenar, gerenciar e viabilizar o acesso às meta-informações do sistema. Cada fonte de dados possui metadados (Metadados-FD) de acordo com seu modelo de dados. Os Metadados-FD são mapeados para os metadados de exportação e estes

compõem o metadados global. Visões do metadados global formam os metadados externos, conforme apresentado no capítulo 4.

A Figura 5.5 apresenta o diagrama de classes do componente Metadados. As classes MetadadosExportação, MetadadosGlobal e MetadadosExterno são relacionadas aos respectivos metadados.

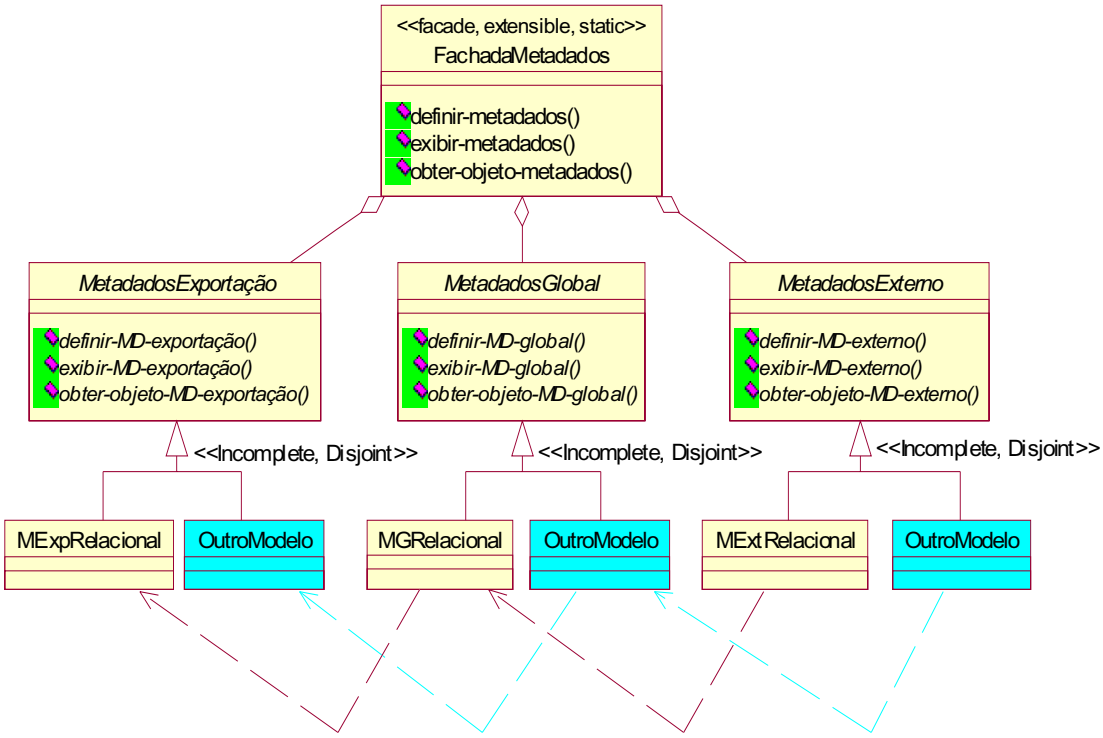

Figura 5.5 - Diagrama de classes do componente Metadados.

As chamadas externas ao componente Metadados são sempre realizadas através da fachada, permitindo flexibilidade no projeto e implementação de suas funcionalidades internas. A FachadaMetadados foi definida como *static* pois não permite reconfiguração dinâmica; e *extensible* pois pode ser necessária sua extensão para atender a um sistema configurado especifico com a definição de novas operações.

Mantendo-se a mesma fachada de acesso, pode-se internamente utilizar diferentes linguagens de consulta e diferentes modelos de dados. A especificação da restrição *disjoint* no diagrama se refere a de apenas um modelo de dados em um sistema configurado.

5.6 – O Componente Processamento de Consulta

O Componente Processamento de Consulta em sistemas integradores de dados é responsável por transformar uma consulta, escrita em uma linguagem de alto nível utilizada no sistema, em uma estratégia eficiente de execução, considerando as características das fontes de dados. A figura 5.6 apresenta o diagrama de classes do componente Processamento de Consulta.

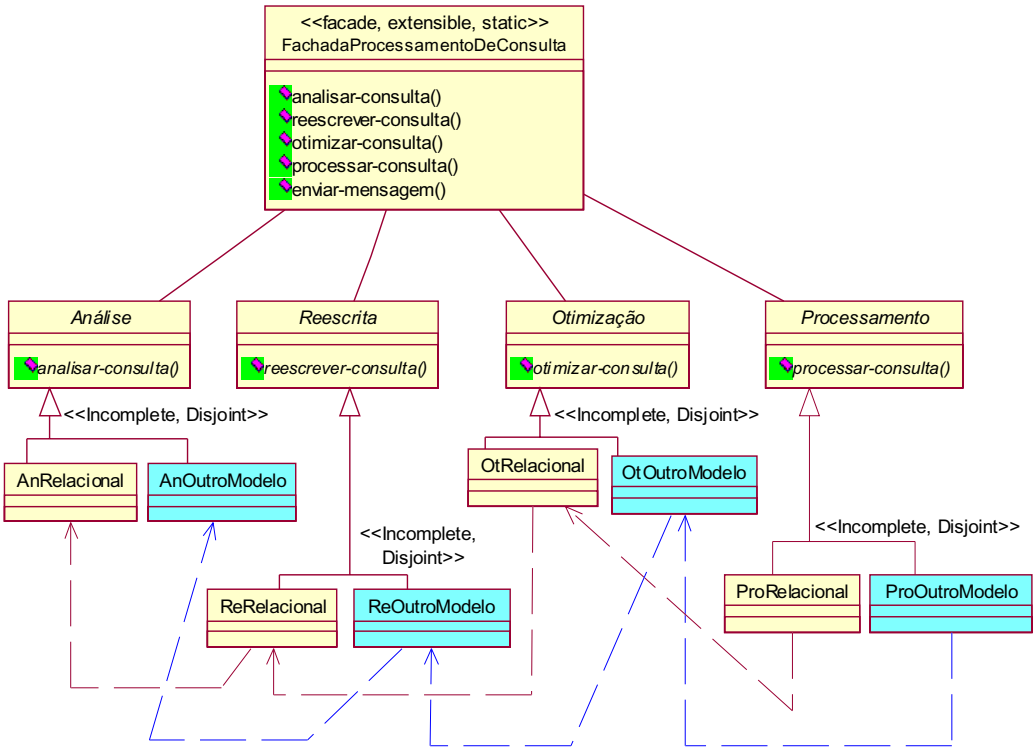

Figura 5.6 – Componente Processamento de Consulta

A consulta de entrada pode apresentar uma grande complexidade que depende das facilidades fornecidas pela linguagem. O processamento de consulta deve interpretar e analisar a consulta global recebida, definir as sub-consultas que serão submetidas a cada uma das fontes de dados, produzir um plano de execução global otimizado, gerenciar a execução da consulta e proceder à composição dos resultados. Por efetuar várias tarefas específicas, o serviço processamento de consulta foi subdividido, para efeito deste trabalho, em quatro módulos: análise, re-escrita, otimização e processamento.

As chamadas de outros componentes ao componente Processamento de Consulta são sempre realizadas através da fachada, permitindo flexibilidade no projeto e implementação de suas funcionalidades internas, como por exemplo a utilização de diferentes técnicas para otimização e execução de consultas. A FachadaProcessamentoDeConsulta foi definida como *static* pois não permite reconfiguração dinâmica; e *extensible* pois pode ser necessária sua extensão para atender a um sistema configurado especifico com a definição de novas operações, como a necessidade de se dividir os módulos aqui definidos ou a inclusão de outros.

Seus pontos de flexibilização são as diferentes linguagens de consulta a serem utilizadas e as técnicas e estratégias para a geração do plano de execução otimizado.

5.7 – Os Componentes Gerência de Transação, Controle de Concorrência e Gerência de Regras

Conforme apresentado no capítulo 4, no contexto deste trabalho, os componentes Gerência de Transação, Controle de Concorrência e Gerência de Regras, apesar de constantes na especificação do *framework*, não serão aqui especificados, devendo ser melhor estudados e detalhados em outros trabalhos dirigidos para este fim, considerando a complexidade e especificidade de tais linhas de pesquisa.

5.8 – Diagramas de Casos de Uso

São os seguintes os casos de uso do ambiente CoDIMS: Projeto do Sistema, Configuração, Projeto da Aplicação e Utilização.

5.8.1 – Projeto do Sistema

Por projeto do sistema entende-se selecionar (definir) quais serviços componentes estarão presentes em uma determinada configuração, de acordo com as características (funcionalidades) a serem disponibilizadas pelo sistema integrador a ser configurado, a partir da análise de requisitos da aplicação. Os componentes Controle, Interface, Acesso aos Dados, Metadados e

Processamento de Consulta sempre estarão presentes em qualquer configuração. O que pode ocorrer é que em diferentes configurações eles apresentem diferentes interfaces, conforme descrito anteriormente.

Definidos os componentes, por serem *frameworks*, torna-se necessário instanciá-los através do preenchimento de seus *hot-spots*, customizando, assim, o sistema integrador de acordo com as características da aplicação. Neste momento são definidas as formas de comunicação com a aplicação cliente, o modelo de dados e a linguagem a serem utilizados etc. A atividade de implementação é executada pelo projetista do sistema (figura 5.7).

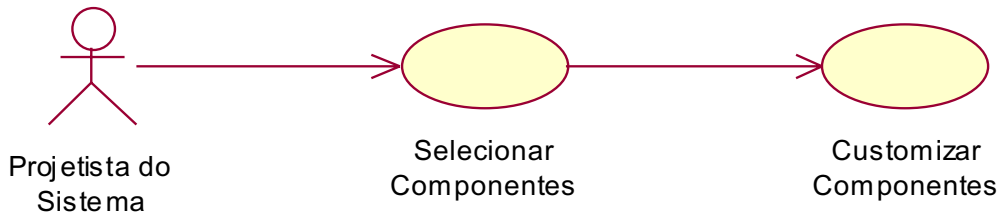

Figura 5.7 – Caso de Uso: Projeto do Sistema

5.8.2 – Configuração Física e Lógica

No processo de configuração física são registrados no Controle os componentes que farão parte do sistema configurado. Na configuração lógica é definido o *workflow*. Os procedimentos para a configuração do sistema foram descritos no capítulo 4. A atividade de configuração é executada pelo configurador do sistema (figura 5.8).

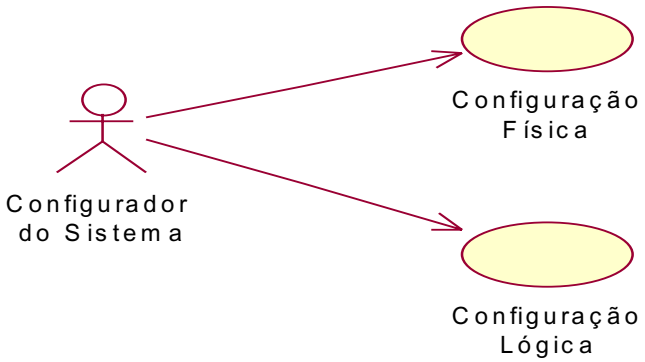

Figura 5.8 – Caso de Uso: Configuração

5.8.3 – Projeto da Aplicação

Configurado o sistema, o mesmo está apto para receber o projeto da aplicação, ou seja, deve ser definido o metamodelo da aplicação, através da declaração dos metadados, o global e os de exportação. A atividade de modelagem é executada pelo projetista da aplicação (figura 5.9).

Figura 5.9 – Caso de Uso: Projeto da Aplicação

5.8.4 – Utilização

Finalmente o sistema está pronto para ser utilizado, através da submissão de um comando de consulta pelo usuário final da aplicação cliente (figura 5.10).

Figura 5.10 – Caso de Uso: Utilização

5.9 - Diagramas de Seqüência

Cada caso de uso possui diagramas de seqüência que serão apresentados a seguir.

5.9.1 – Definir Configuração

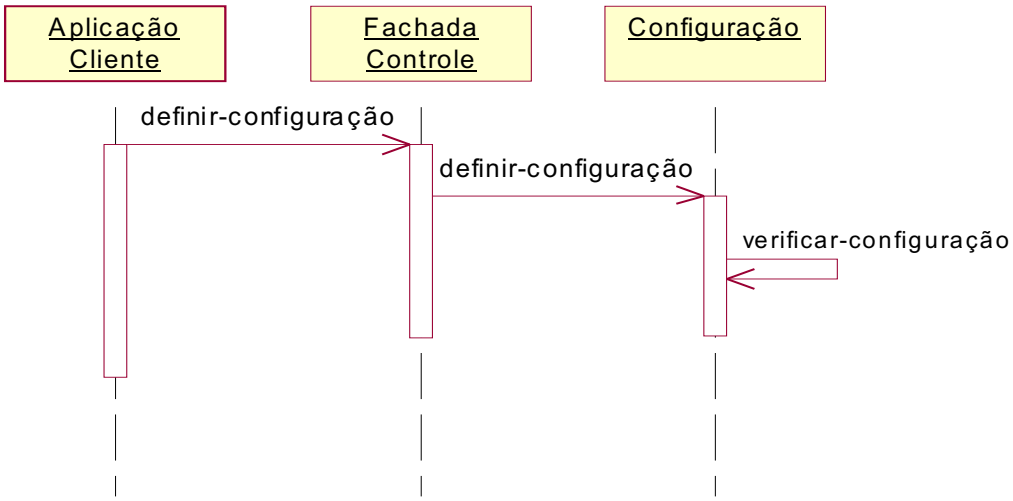

Figura 5.11 – Diagrama de Seqüência: Configuração

5.9.2 – Definir *Workflow*

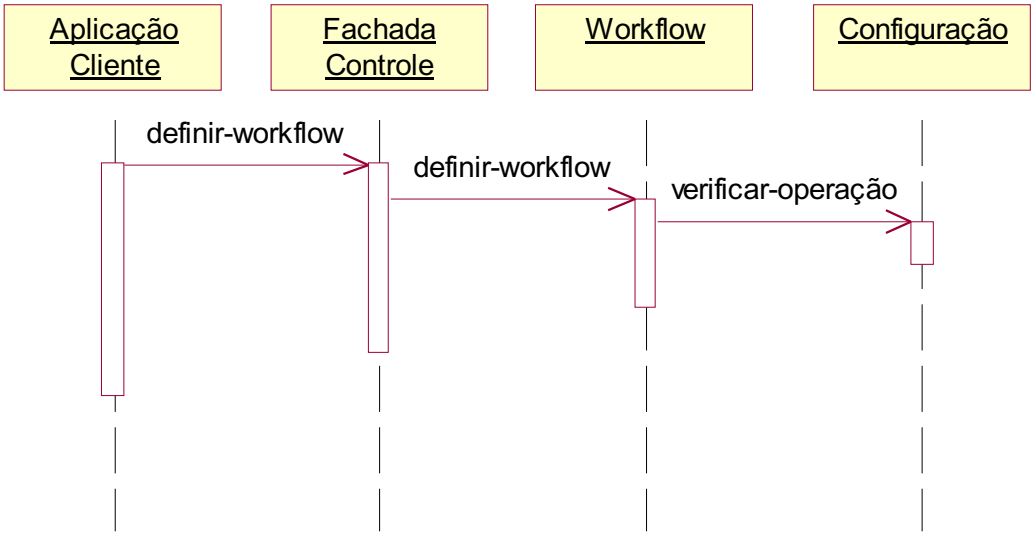

Figura 5.12 – Diagrama de Seqüência: Definir *Workflow*

5.9.3 – Definir Metadados

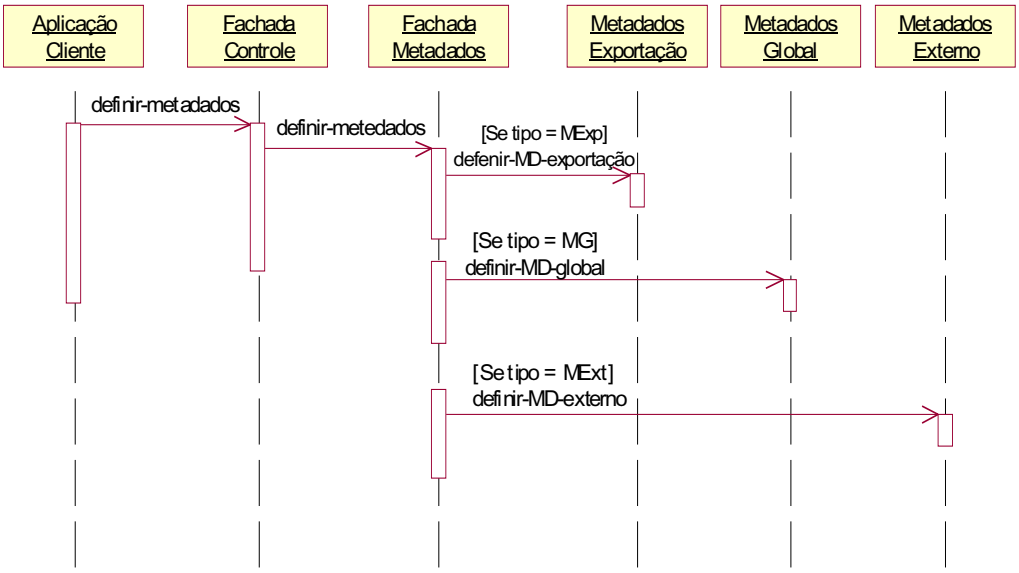

Figura 5.13 – Diagrama de Seqüência: Definir Metadados

5.9.4 – Executar Consulta

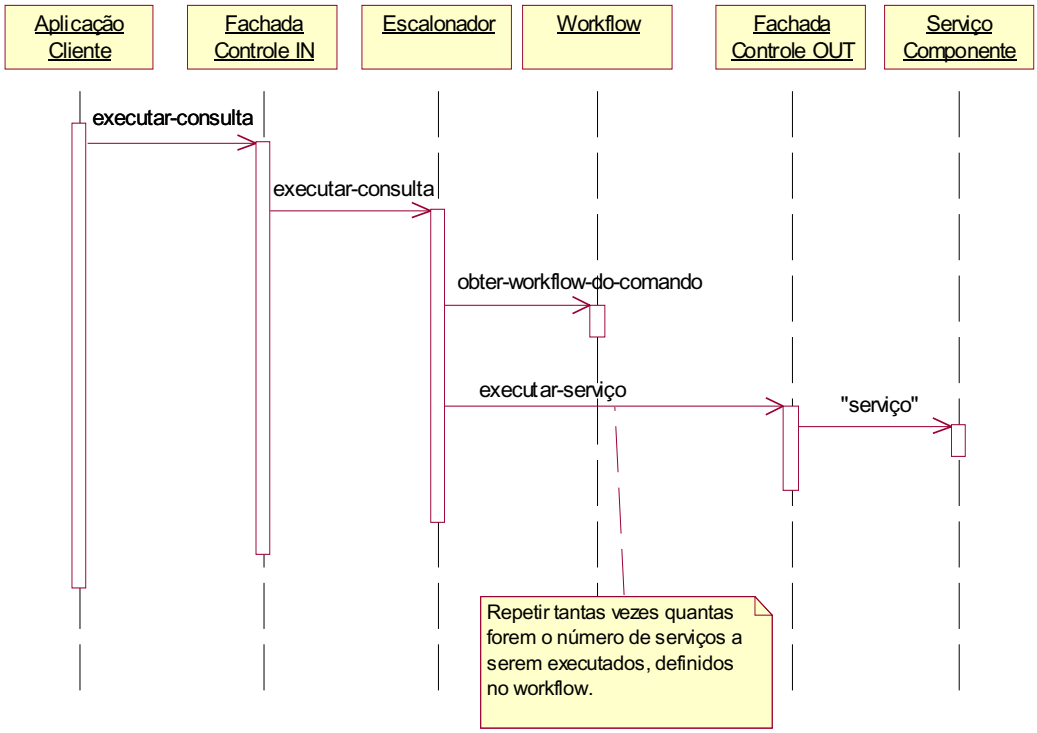

Figura 5.14 – Diagrama de Seqüência: Executar Consulta

5.9.5 – Enviar Mensagem

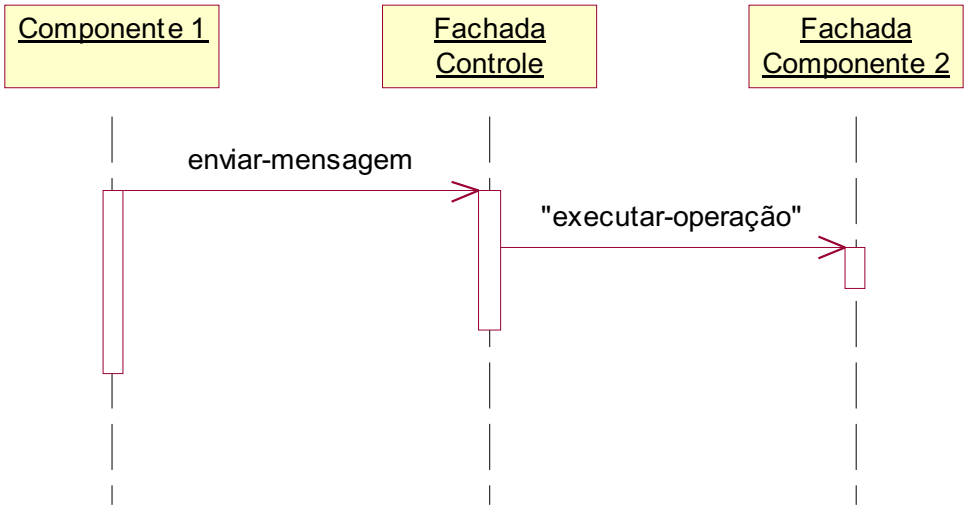

Figura 5.15 – Diagrama de Seqüência: Enviar Mensagem

5.10 – Síntese do Capítulo

Neste capítulo foi apresentada a especificação do CoDIMS através da *UML* e de uma notação estendida. Foram ainda detalhados os processos para as definições das configurações física e lógica.

No próximo capítulo é apresentado um estudo de caso, onde é mostrado, detalhadamente, todas as etapas e os procedimentos necessários para a geração de um sistema configurado, bem como o processo de execução de uma consulta.

Capítulo 6 – Estudo de Caso

Este capítulo apresenta um estudo de caso descrevendo de forma detalhada todas as fases necessárias para os processos de configuração do ambiente CoDIMS, o projeto da aplicação, a submissão de consultas e comentários sobre a implementação.

6.1 – Introdução

O objetivo deste capítulo é apresentar um exemplo de configuração e customização do ambiente CoDIMS, ilustrando os casos de uso: o processo de seleção/definição dos componentes, as instanciações destes componentes de acordo com a aplicação, o processo de configuração e de definição do *workflow*, a definição dos metadados e por último a submissão de consultas. Também são feitos alguns comentários sobre a implementação do protótipo.

O caso em estudo é o mesmo apresentado em [Silv99], com algumas adaptações, dentro do objetivo do projeto ECOBASE de integração de dados heterogêneos e distribuídos voltados para a solução de problemas de meio ambiente.

6.2 – Descrição da Aplicação

Uma Organização Não Governamental (ONG) “Rios e Lagos” deseja controlar a pesca predatória e necessita integrar informações dos desembarques pesqueiros realizados em uma determinada região e dos dados científicos das espécies de peixes que se deseja controlar. Em cada local de desembarque existe um sistema de entrada de dados que alimenta uma base de dados central que armazena informações sobre cada um dos desembarques realizados. Em um banco de dados de uma empresa de pesquisa do governo, existe informações sobre as diversas espécies de peixes que habitam a região.

O que a ONG deseja conhecer são informações sobre a pesca de espécies em período de reprodução e quantidade pescada de cada uma das espécies. Para isso é necessário o acesso e a integração destas duas fontes de dados distribuídas e heterogêneas.

A base de dados “Desembarque” tem seus dados armazenados segundo o modelo *XML*, com a seguinte descrição:

```
<!DOCTYPE Desembarque>
<!ELEMENT Pesca>
<!ATTLIST Pesca
    cod-desembarque          ID        #REQUIRED
    cod-local                CDATA     #REQUIRED
    dia-desembarque          CDATA     #REQUIRED
    mes-desembarque          CDATA     #REQUIRED
    ano-desembarque          CDATA     #REQUIRED
    cod-especie              CDATA     #REQUIRED
    qtde-Kg                  CDATA     #REQUIRED
    cod-tipo-embarcação      CDATA     #REQUIRED
    cod-aparelho-pesca       CDATA     #REQUIRED
    qtde-pescadores          CDATA     #REQUIRED
    qtde-dias-pesca          CDATA     #REQUIRED>
```

O banco de dados “Pesquisa” está armazenado em um ambiente relacional, com o seguinte esquema:

Tabela	Atributo	Tipo
espécie	cod-espécie	char(5)
	mes-reprodução	char(2)
	nome-científico	char(30)
	nome-vulgar	char(30)

6.3 – Projeto do Sistema

Inicialmente, deve ser especificado, pelo projetista do sistema, o projeto do sistema a ser configurado. De acordo com a descrição da aplicação, somente será necessário ao sistema integrador a ser configurado realizar consultas nas fontes de dados. Assim, somente serão necessários estar presentes no sistema, juntamente com o Controle, os componentes Gerência de Metadados, Processamento de Consulta e Acesso aos Dados (Figuras 6.1 e 6.2).

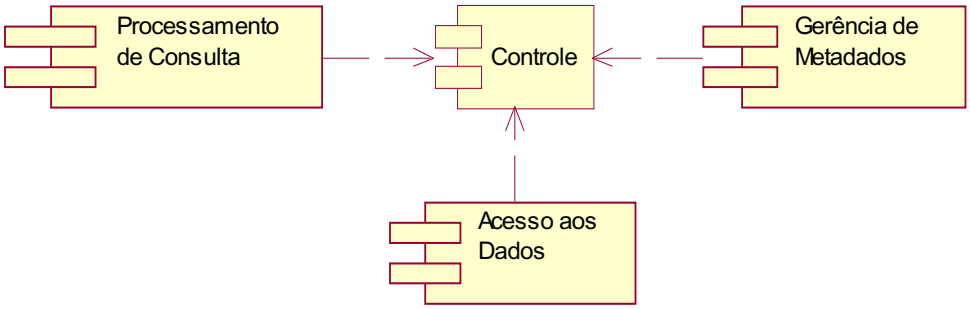

Figura 6.1 – Diagrama de Componentes

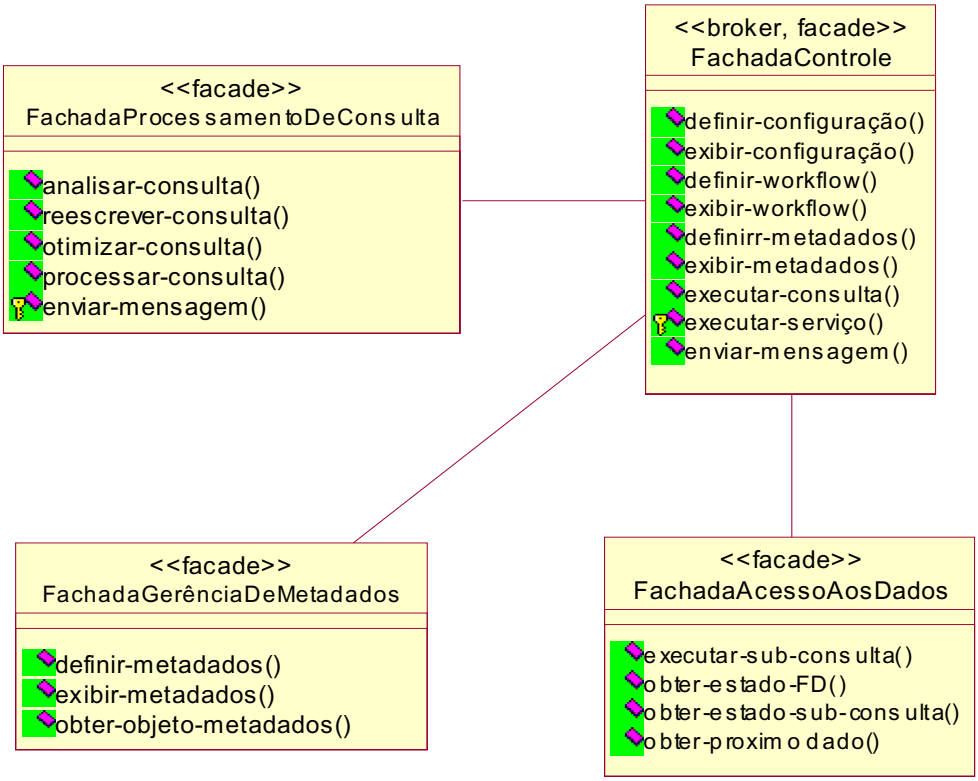

Figura 6.2 – Diagrama de Fachadas

Selecionados os componentes, deve ser realizada a customização dos mesmos, de acordo com as características do sistema a ser configurado. No caso em questão, serão adotadas as seguintes características:

- Será utilizado o modelo relacional como modelo de dados comum (MDC);
- A linguagem de consulta será *SQL*;
- A comunicação entre os componentes será através de *RMI (Remote Method Invocation*);
- A comunicação entre o sistema configurado e o banco de dados "Científico" será feita utilizando *JDBC*;
- A comunicação entre o sistema configurado e a base de dados "Desembarque" será feita utilizando *RMI*.

A seguir, são apresentados os diagramas de classes dos componentes selecionados e customizados.

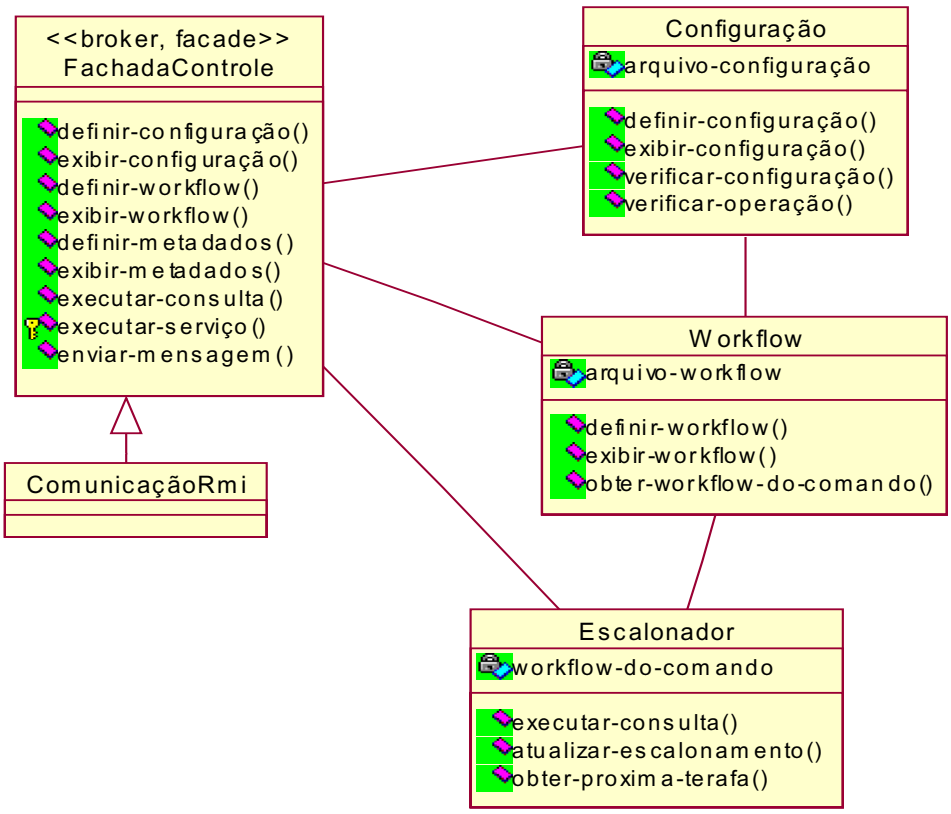

Figura 6.3 – O Componente Controle

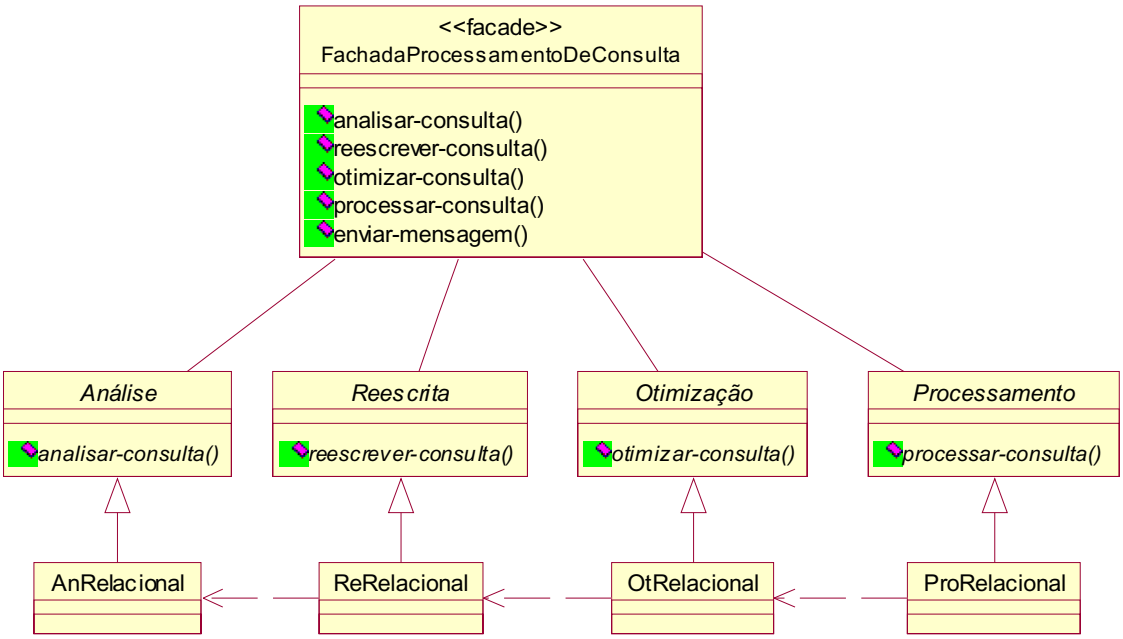

Figura 6.4 – O Componente Processamento de Consulta

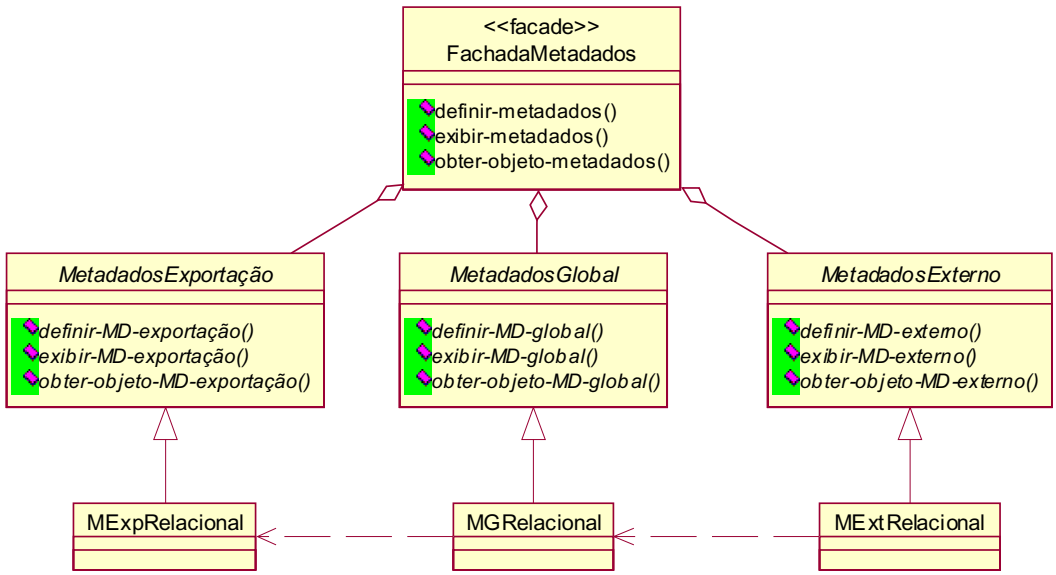

Figura 6.5 – O Componente Metadados

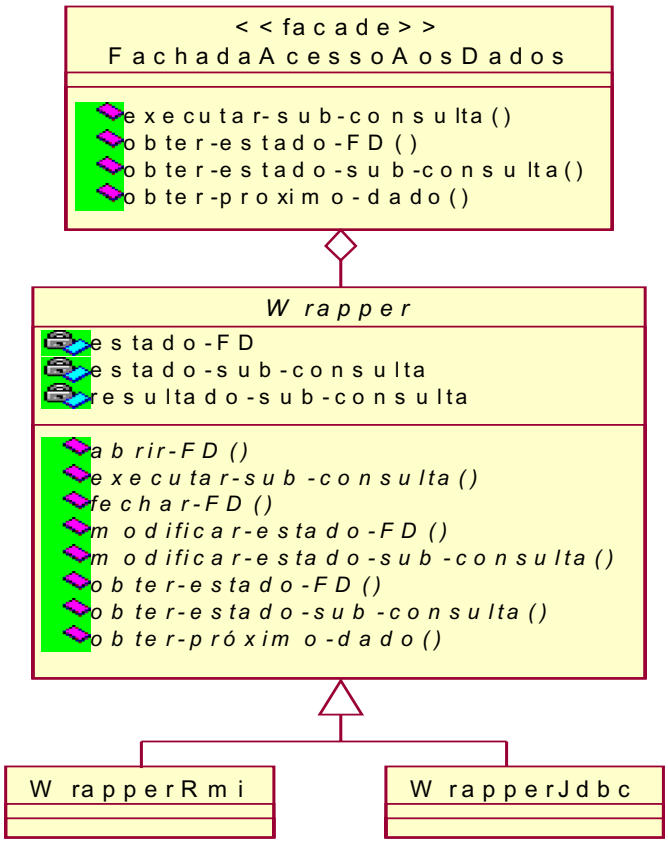

Figura 6.6 – O Componente Acesso Aos Dados

6.4 – Configuração do Sistema

Selecionados e customizados os componentes, a etapa seguinte é a de configuração, física e lógica.

6.4.1 – Configuração Física

A configuração física ocorre com a definição no Controle dos componentes que farão parte do sistema configurado. Deve ser criado um arquivo de *script*, conforme apresentado a seguir, especificando, para cada componente, as operações a serem oferecidas e as que serão requisitadas. Para cada operação é fornecida uma lista contendo seus parâmetros e tipos para checagem de tipos, incluindo um código de retorno (CR). A relação completa das operações se encontra no Anexo A.

Define Component ***Metadados***

Offered-Operations

definir-metadados *(string* tipo-MD*, string* arq-script*, int* CR*);*

exibir-metadados (*string* tipo-MD*, string* arq*, int* CR*);*

obter-objeto-MD *(int* id*, string* tipo-MD*, string* nome-obj*, string* obj*, int* CR*);*

End-Component.

Define Component ***AcessoAosDados***

Offered-Operations

executar-sub-consulta *(int* id*, string* nome-FD*, string* sub-cons*, string* arq-res*, int* CR*);*

obter-próximo-dado *(int* id*, string* nome-FD*, string* dados*, int* CR*);*

End-Component.

Define Component **ProcessamentoDeConsulta**

Offered-Operations

analisar-consulta *(int* id*, string* cons*, string* grafo-cons*, int* CR*);*

reescrever-consulta *(int* id*, string* grafo-cons*, string* arv-op*, int* CR*);*

otimizar-consulta *(in* id*t, string* arv-op*, string* arv-exec*, int* CR*);*

processar-consulta *(int* id*, string* arv-exec*, string* arq-res*, int* CR*)*

Requested-Operations

Metadados, obter-objeto-MD *(int* id*, string* tipo-MD*, string* nome-obj*, string* obj*, int* CR*);*

AcessoAosDados, executar-sub-consulta *(int* id*, string* nome-FD*, string* sub-cons*, string* arq-res*, int* CR*);*

AcessoAosDados, obter-próximo-dado *(int* id*, string* nome-FD*, string* dados*, int* CR*);*

End-Component.

Este arquivo de *script* é submetido através da operação ***definir-configuração*** que após definir os componentes, chama a operação ***verificar-***

configuração para checar se todas as operações a serem requisitadas foram, de fato, oferecidas.

6.4.2 – Configuração Lógica

O processo de configuração lógica refere-se à definição de um *workflow* para cada um dos comandos da linguagem de consulta a ser utilizada pelo sistema e que efetivamente serão submetidos para processamento. Neste estudo de caso, considerando que o sistema somente fará a leitura dos dados, apenas o comando ***Select***, do conjunto de comandos da *SQL*, terá aqui definido seu *workfow*, através da criação e submissão do seguinte arquivo de *script*:

Define Workflow ***Select***

Operations

ProcessamentoDeConsulta (analisar-consulta);

ProcessamentoDeConsulta (reescrever-consulta);

ProcessamentoDeConsulta (otimizar-consulta);

ProcessamentoDeConsulta (processar-consulta)

*End-Operations****.***

Este arquivo de *script* é submetido através da operação ***definir-workflow*** que ao seu final chama a operação ***verificar-operação*** para checar se todas as operações definidas no *workflow* estão, de fato, sendo oferecidas pelos respectivos componentes.

6.5 – Projeto da Aplicação

A partir da definição dos dados das fontes de dados, o projetista da aplicação deve definir os metadados do sistema: os de exportação e o metadados global. Os metadados externos correspondem a visões do metadados Global e não serão definidos neste trabalho.

6.5.1 – Metadados de Exportação

Os metadados de exportação definem os subconjuntos dos dados de cada fonte de dados que estará disponível para a integração, sendo análoga a uma definição de visão. Como o modelo de dados do sistema foi definido como Relacional, todos os metadados de exportação e o Global serão definidos neste mesmo modelo, neste caso sob a forma de tabelas.

Os metadados de exportação para "Desembarque" serão:

Tabela	Atributo	Tipo
Pesca-Exp	cod-desembarque	char(5)
	cod-local	char(2)
	dia-desembarque	char(2)
	mes-desembarque	char(2)
	ano-desembarque	char(2)
	cod-especie	char(5)
	qtde-Kg	char(5)
	cod-aparelho-pesca	char(2)

Os metadados de exportação para "Pesquisa" serão:

Tabela	Atributo	Tipo
Espécie-Exp	cod-espécie	char(5)
	Mes-reprodução	char(2)
	Nome-vulgar	char(30)

Para a definição das tabelas Pesca-Exp e Especie-Exp deverá ser editado um arquivo de *script*, contendo os seguintes comandos, baseados na *SQL*:

Create table Pesca_Exp

(cod_desembarque char (5) not null,
cod_local char (2),
dia_desembarque char (2),
mes_desembarque char (2),
ano_desembarque char (2),
cod_especie char (2),
qtde_kg char (5),
cod-aparelho-pesca char (2),
Data Source is "Desembarque")

Create table Especie_Exp

(cod_especie char (5) not null,
mes_reproducao char (2),
nome-vulgar char(30),
Data Source is "Pesquisa")

O projetista da aplicação envia este arquivo ao componente Controle através da operação ***definir-metadados***, com a opção definir-MD-exportação.

6.5.2 – Metadados Global

Nesta aplicação, dada a simplificação do modelo original da aplicação existente em [Silv99] e nos requisitos da aplicação, os metadados globais conterão as mesmas definições das tabelas existentes nos metadados de exportação. Os metadados globais são definidos como visões dos metadados de exportação. Em outras aplicações mais completas, pode ocorrer de várias tabelas pertencentes a diferentes metadados de exportação serem transformadas em

uma única tabela no metadados Global, com todos os problemas de integração de esquemas descritos no capítulo 3, o que não é o enfoque deste trabalho.

Tabela Global	Atributo Global	Tabela-Exportação	Atributo-Exportação	Tipo
Pesca-Glob	cod-desembarque	Pesca-Exp	cod-desembarque	char(5)
	cod-local	Pesca-Exp	cod-local	char(2)
	dia-desembarque	Pesca-Exp	dia-desembarque	char(2)
	mes-desembarque	Pesca-Exp	mes-desembarque	char(2)
	ano-desembarque	Pesca-Exp	ano-desembarque	char(2)
	cod-especie	Pesca-Exp	cod-especie	char(5)
	qtde-Kg	Pesca-Exp	qtde-Kg	char(5)
	cod-aparelho-pesca	Pesca-Exp	cod-aparelho-pesca	char(2)
Especie-Glob	cod-espécie	Especie-Exp	cod-espécie	char(5)
	mes-reprodução	Especie-Exp	mes-reprodução	char(2)
	nome-vulgar	Especie-Exp	nome-vulgar	char(30)

Para a definição das tabelas Pesca-Glob e Especie-Glob deverá ser editado um arquivo de *script*, com os seguintes comandos *SQL*, para permitir referenciar cada atributo global ao(s) atributo(s) de exportação correspondente(s).

```
Create view Pesca_Glob as
        Select cod_desembarque, cod_local, dia_desembarque,
                mes_desembarque, ano_desembarque, cod_especie,
                qtde_kg, cod-aparelho-pesca
        From Pesca_Exp

Create view Especie_Glob as
        Select cod_especie, mes_reproducao, nome_vulgar
        From Especie-Exp
```

O projetista da aplicação envia este arquivo ao componente Controle através da operação ***definir-metadados***, com a opção definir-MD-global.

6.6 – Utilização do Sistema

Após a realização de todas as etapas descritas anteriormente, o sistema está configurado e apto para receber as submissões de consultas.

Seja, então, a seguinte consulta em *SQL* sobre o metadados global:

Select cod_local, dia_desembarque, mes_desembarque, ano_desembarque,
cod_especie, qtde_kg
From Pesca_Glob P, Especie_Glob E
Where P.mes_desembarque = E.mes_reprodução and
P.cod_especie = E.cod_especie

A aplicação cliente submete esta consulta através da *API* do componente Controle, chamando a operação ***executar-consulta***, que a repassa ao Escalonador. Identificado que se trata do comando ***Select***, através da consulta ao *workflow* definido para o referido comando, cria-se um escalonamento para a execução. De acordo com a definição, as operações a serem executadas, por ordem, são: ***analisar-consulta***, ***reescrever-consulta***, ***otimizar-consulta*** e ***processar-consulta***, todas pertencentes ao componente Processamento de Consulta.

6.6.1 – Operação *analisar-consulta*

A operação ***analisar-consulta*** faz a análise léxica e sintática da consulta através de consulta ao componente Metadados, gerando o grafo de consulta apresentado na figura 6.7.

Os nós **P** e **E** correspondem às tabelas globais Pesca_Glob e Especie_Glob respectivamente. O outro nó presente no grafo é o nó resultado. As arestas correspondem às operações de junção e de projeção da álgebra relacional.

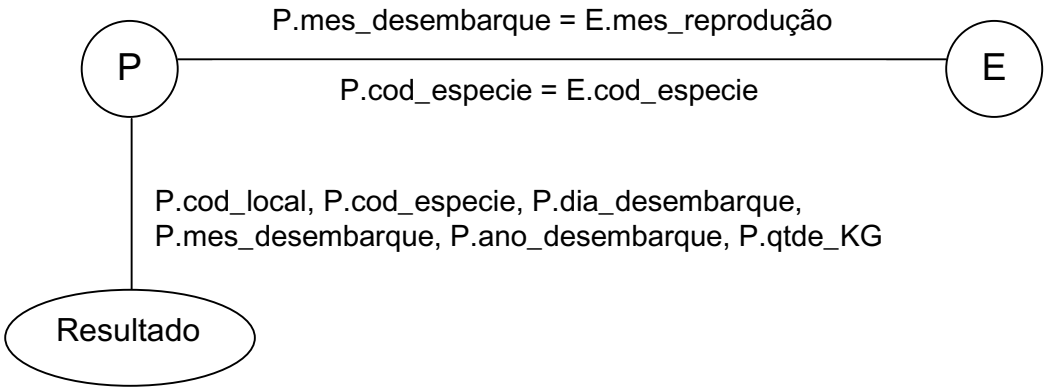

Figura 6.7 – Grafo de consulta

6.6.2 – Operação *reescrever-consulta*

A operação ***reescrever-consulta*** produz uma árvore de operação (figura 6.8) a partir dos dados a serem fornecidos por cada uma das fontes de dados. As sub-consultas que serão encaminhadas a cada uma das fontes de dados são geradas e agrupadas de maneira que a uma fonte de dados seja encaminhada apenas uma sub-consulta para cada consulta global recebida, para diminuir o número de acessos a cada fonte.

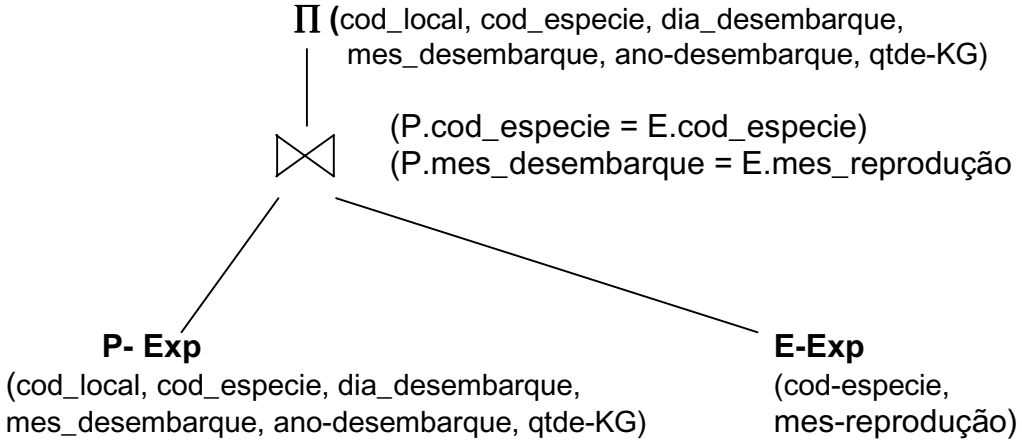

Figura 6.8 – Árvore de operação

Os nós **P-Exp** e **E-Exp** correspondem às tabelas de exportação Pesca_Exp e Especie_Exp respectivamente. A consulta global descrita no

grafo de consulta foi transformada em uma árvore de operação, separando os dados a serem acessados de acordo com as respectivas fontes de dados.

A operação ⋈ corresponde à junção das tabelas Pesca_Exp e Especie_Exp. Por fim, para formar o resultado é necessário um operador de projeção **∏.**

A árvore de operação é gerada sem a preocupação com a ordem de execução destas operações. Com o objetivo de otimizar a seqüência de execução destas operações, a árvore de operação deve ser transformada em uma árvore de execução, como será visto a seguir.

6.6.3 – Operação *otimizar-consulta*

A operação ***otimizar-consulta*** gera, a partir da árvore de operação, um plano de execução global otimizado, denominado árvore de execução, onde são ordenadas as operações a serem executadas com o objetivo de minimizar o custo.

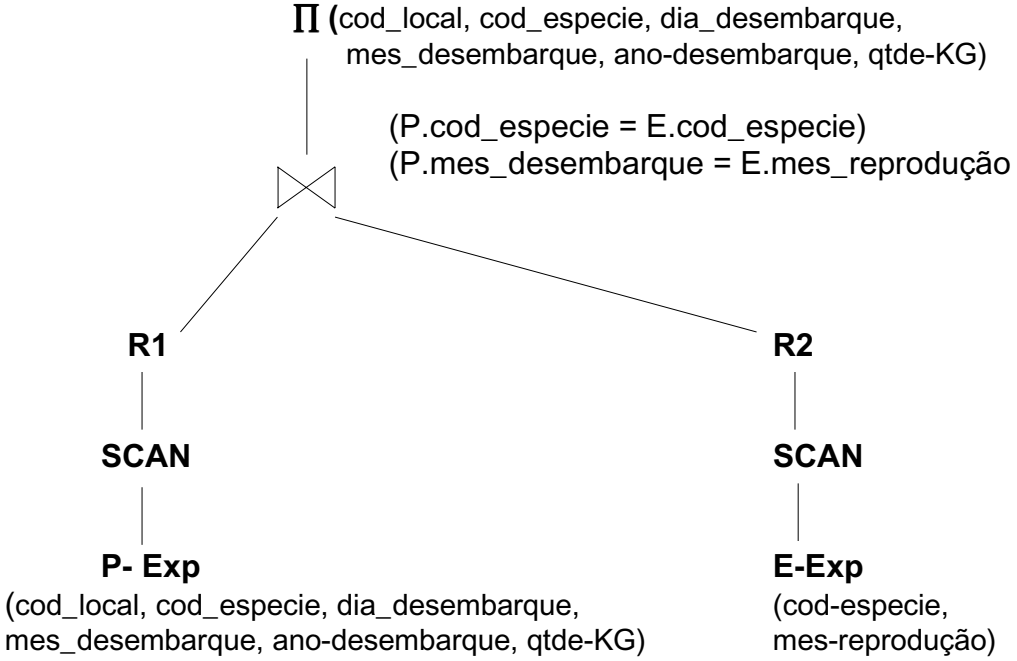

Figura 6.9 – Árvore de execução

Neste exemplo, a árvore de execução é semelhante à de operação descrita anteriormente pois existem apenas duas fontes de dados a serem

integradas. No caso de existirem mais de duas fontes, a árvore de execução gerada pelo otimizador deve estabelecer a hierarquia de execução das operações, principalmente das junções, de forma que árvore gerada, quando executada, leve a um custo mínimo.

Na figura 6.9, a operação SCAN representa o acesso propriamente dito à fonte de dados enquanto que R1 e R2 correspondem aos resultados da execução de cada uma das sub-consultas.

6.6.4 – Operação *processar-consulta*

A operação ***processar-consulta*** realiza a execução da consulta, a partir da árvore de execução.

Primeiramente são geradas as sub-consultas que serão encaminhadas a cada *wrapper*. Neste exemplo, são duas sub-consultas: uma para P-Exp (Pesca-Exp) a ser encaminhada à fonte de dados **Desembarque** e a outra sobre E-Exp (Especie-Exp) a ser encaminhada à fonte de dados **Pesquisa**.

Em seguida, estas sub-consultas são encaminhadas ao componente Acesso aos Dados através da invocação da operação ***executar-sub-consulta***.

Quando do retorno dos resultados das sub-consultas, ocorre a integração (junção) dos dados. A execução da operação continua até que toda a árvore tenha sido processada. O resultado final é armazenado em um arquivo a ser acessado posteriormente pela aplicação cliente.

6.7 – Incluindo Novas Funcionalidades no Sistema Configurado

Para melhor exemplificar a flexibilidade e a importância do mecanismo de configuração lógica do sistema, através do *workflow*, analisamos o caso de ser necessário incorporar ao sistema configurado, apresentado neste capítulo, três novas funcionalidades, através dos seguintes comandos *SQL-like*:

- ***Show Plan***: dada uma consulta *SQL* através da cláusula *Select*, exibe o plano otimizado gerado.
- ***No Optmize***: dada uma consulta *SQL* através da cláusula *Select*, executa a consulta sem otimizá-la. Neste caso, a seqüência de execução das junções seria a ordem em que as tabelas/atributos aparecem no comando.

- ***Analize***: dada uma consulta *SQL* através da cláusula *Select*, apenas faz a análise do comando.

Para a inclusão destas novas funcionalidades no sistema configurado, haveria a necessidade de se incorporar a definição de um *workflow* para cada um destes novos comandos, de forma a gerar um escalonamento específico para cada um deles em tempo de execução. Assim, o arquivo de *script* para a definição da configuração lógica seria o seguinte:

Define Workflow ***Select***
Operations
ProcessamentoDeConsulta (analisar-consulta);
ProcessamentoDeConsulta (reescrever-consulta);
ProcessamentoDeConsulta (otimizar-consulta);
ProcessamentoDeConsulta (processar-consulta)
End-Operations.

Define Workflow ***Show Plan***
Operations
ProcessamentoDeConsulta (analisar-consulta);
ProcessamentoDeConsulta (reescrever-consulta);
ProcessamentoDeConsulta (otimizar-consulta)
End-Operations.

Define Workflow ***No Optmize***
Operations
ProcessamentoDeConsulta (analisar-consulta);
ProcessamentoDeConsulta (reescrever-consulta);
ProcessamentoDeConsulta (processar-consulta)
End-Operations.

Define Workflow ***Analize***

Operations

ProcessamentoDeConsulta (analisar-consulta)

End-Operations.

Assim, constatamos que podemos ter diferentes escalonamentos para atender às diferentes requisições, bastando para isso a definição adequada do *workflow*. De forma geral, o mecanismo de configuração lógica permite desacoplar os serviços entre si. Desta forma uma mesma configuração pode ser adaptada para atender a novos requisitos do usuário, o que mostra mais um aspecto da flexibilidade do sistema.

6.8 – Implementação do Protótipo

Um protótipo do sistema foi implementado no TecBD (Laboratório de Tecnologias em Banco de Dados), do Departamento de Informática da PUC-Rio, utilizando a linguagem *Java* no ambiente *Windows-NT*. O protótipo implementa o estudo de caso apresentado neste capítulo, sendo dado enfoque ao componente Controle que permite, em tempo de execução, a chamada das operações oferecidas pelos componentes de acordo com o especificado no *workflow* do comando a ser executado.

Os componentes foram desenvolvidos e compilados separadamente. A comunicação entre eles foi implementada através de *RMI* o que inclui mais uma facilidade ao sistema: pode ser gerado um sistema configurado com o uso de componentes remotos já disponíveis.

Foi utilizado o SGBD relacional Oracle 8i para o armazenamento dos dados da tabela **Pesquisa** e um arquivo texto contendo a base de dados **Desembarque** no formato *XML*.

6.9 – Síntese do Capítulo

Neste capítulo foi apresentado um exemplo de configuração e de customização do ambiente CoDIMS, através de um estudo de caso. Também foram feitos comentários sobre a implementação do protótipo.

No próximo capítulo são apresentadas as conclusões e as principais contribuições deste trabalho. São ainda listados alguns tópicos para futuros trabalhos.

Capítulo 7– Conclusões, Contribuições e Trabalhos Futuros

Este capítulo apresenta as conclusões deste trabalho, suas principais contribuições e a indicação de alguns trabalhos futuros para a continuidade deste pesquisa.

7.1 – Conclusões

O objetivo deste trabalho é a especificação e o desenvolvimento do CoDIMS: um ambiente flexível e configurável, com a finalidade de gerar sistemas configurados para a integração de dados heterogêneos e distribuídos. O CoDIMS possibilita a integração de componentes que implementam serviços *middleware* de integração de dados, denominados DIMS. Componentes DIMS são baseados em serviços típicos de gerência de dados, tais como aqueles existentes nos SGBDs. Neste contexto, um sistema configurado é obtido através da seleção, customização e integração de um conjunto adequado de componentes DIMS.

A configuração permite a cada sistema configurado resultante, utilizar o menor número possível de componentes, minimizando o código a ser gerado, tornando-se, então, um sistema *light weight*. Assim, o sistema gerado conterá apenas os componentes adequados e necessários para atender aos requisitos de uma aplicação, o que denominamos no capítulo 1 de "*what you need is only what you get" (WYNWYG)*.

A ênfase do CoDIMS é em flexibilidade e configuração, através do uso das técnicas de *frameworks* e de componentes. O uso de componentes, permitindo separar a interface de sua implementação, fornece flexibilidade e configuração ao sistema gerado. Para uma mesma interface podem existir diversas implementações, ou seja, diversos componentes. Da mesma forma, um componente pode implementar diversas interfaces. O nosso modelo de

configuração permite a um componente estender sua interface, bastando para isso apenas defini-la no arquivo de configuração.

Os componentes DIMS foram especificados e desenvolvidos com a utilização da técnica de *frameworks*. Componentes como *frameworks* tornam-se unidades de encapsulamento permitindo que sejam modificados e recolocados isoladamente, sem afetar os componentes ou as aplicações existentes. *Frameworks* permitem aumentar a flexibilidade devido a uma maior facilidade de customização, de acordo com os requisitos da aplicação. Usar *frameworks* possibilita reuso de análise, de projeto e de código, aumenta a qualidade do *software*, reduz o esforço e o tempo de desenvolvimento de novas aplicações e de manutenção.

A essência da abordagem do CoDIMS é o componente Controle, que permite configuração física e lógica:

- Na fase de configuração física, os componentes requeridos pela aplicação são selecionados e customizados de acordo com os seus requisitos. Em seguida, eles são registrados no arquivo de configuração.
- Na fase de configuração lógica são definidos os mapeamentos entre os serviços (operações) oferecidos pelos componentes e o escalonamento necessário para atender a cada requisição global. A configuração lógica é baseada no conceito de *workflow*.

O Controle faz a verificação da configuração do sistema gerado, de forma que todas as operações requisitadas por um componente, estejam definidas como sendo oferecidas por outro. Também é verificado se todas as operações definidas no *workflow* estão sendo oferecidas pelo respectivo componente.

O CoDIMS é altamente flexível em três contextos principais:

- Na escolha e integração dos componentes adequados;
- Na customização de cada componente, possibilitada pelo uso da técnica de *frameworks*;
- Na configuração lógica, que permite o escalonamento de diferentes serviços, para atender às diferentes requisições submetidas pela aplicação.

Esta flexibilidade, possibilitada pelos mecanismos de configuração física e lógica, permite gerar sistemas de integração de dados com diferentes funcionalidades e serviços, como por exemplo:

- Adotar diferentes modelos de dados como modelo de dados comum;
- Utilizar diferentes linguagens de consulta;
- Usar estratégias específicas para a geração do plano de execução otimizado;
- Disponibilizar serviços específicos para o acesso a diferentes tipos e modelos de dados.

A abordagem proposta traz inovações, através dos mecanismos de configuração física e lógica, permitindo construir sistemas configurados apenas com os serviços necessários e adequados à aplicação. Pode ser integrado um número variável de componentes e cada componente, por sua vez, pode oferecer um número variável de operações. Esta flexibilidade torna mais ampla e seletiva a escolha e a integração de componentes, bastando para isso escolher uma combinação adequada de serviços e de componentes, e alterar os arquivos de configuração e de definição do *workflow*. A estrutura do componente Controle se mantém a mesma.

Esta abordagem apresenta diversas vantagens, pois permite:

- O uso (reuso) de componentes já existentes;
- A troca de um componente por outro;
- A integração de novos componentes para atender a uma nova necessidade;
- A adaptação de um componente específico para atender a uma alteração nos requisitos da aplicação;
- O uso de diferentes serviços para aplicações específicas, como: linguagem de consulta, modelo de dados, técnicas de otimização de consultas etc.;
- O escalonamento dos serviços a serem executados, de acordo com o *workflow*.

Finalmente, no capítulo 2 foram apresentados alguns sistemas para integração de dados, dos vários existentes na literatura. O aparente vasto

número de soluções pode, erroneamente, indicar uma área de pesquisa já resolvida. Pelo contrário, é uma área cada vez mais importante e onde não existe uma solução geral que seja adequada ou que se ajuste aos diversos problemas de integração, o que se constata pelo surgimento de novas propostas. Entretanto, tais sistemas não provêem a flexibilidade oferecida pelo ambiente CoDIMS.

7.2 – Contribuições

As principais contribuições deste trabalho são:

- A proposta de um ambiente configurável e flexível para integração de dados heterogêneos e distribuídos. Um mecanismo de configuração física permite a seleção e a integração de um conjunto variável e adequado de componentes DIMS enquanto que um mecanismo de configuração lógica, baseado no conceito de *workflow*, gera o escalonamento dos serviços a serem executados. Os mecanismos de configuração possibilitam o reuso de componentes DIMS já disponíveis, enquanto que a adoção da técnica de *frameworks* para especificação dos componentes DIMS, possibilita maior flexibilidade e facilidade de adaptação dos mesmos a uma aplicação específica.

- Para nosso grupo de pesquisa, a especificação de um ambiente que permita o desenvolvimento, em paralelo, dos componentes DIMS, possibilitando sua composição futura através das interfaces definidas.

- Para o projeto ECOBASE, a possibilidade do desenvolvimento e implementação de componentes com funcionalidades bem definidas e que possam ser compartilhadas com outras instituições, bem como a possibilidade de uso no CoDIMS de outros componentes já disponíveis e que sejam compatíveis.

- O desenvolvimento de um protótipo implementando o estudo de caso apresentado no capítulo 6.

Além disso, foram publicados dois artigos:

- ECOHOOD: Constructing Configured DBMSs based on Frameworks.
Melo, R.N.; Porto, F.A.M.; Lima, F. & Barbosa, A.C.P.
XIII Simpósio Brasileiro de Banco de Dados, Maringá-PR, Brasil, 1998.

- Configurable Data Integration Middleware System.
Barbosa, A.C.P. & Porto, F.A.M
Proceedings of International Workshop on Information Integration on the Web: Technologies and Applications (WIIW2001), Rio de Janeiro, Brasil, 2001.

7.3 – Trabalhos Futuros

A partir da arquitetura proposta e do protótipo implementado neste trabalho, podem ser especificados e desenvolvidos novos componentes/serviços DIMS, bem como o reuso de outros serviços já disponíveis, que sejam adequados às diferentes necessidades das diversas aplicações. Esta evolução natural certamente irá ocorrer quando da utilização do CoDIMS em outras aplicações. A seguir, propomos alguns trabalhos específicos que podem ser pesquisados e desenvolvidos:

- **Especificação e desenvolvimento dos componentes Gerência de Transação, Controle de Concorrência e Gerência de Regras:**

Os componentes Gerência de Transação, Controle de Concorrência e Gerência de Regras devem ser melhor estudados e detalhados em outros trabalhos dirigidos para este fim, considerando a complexidade e especificidade de tais linhas de pesquisa.

- **Integração semântica dos componentes/*frameworks* DIMS:**

Para a identificação e solução dos possíveis problemas de integração semântica dos componentes/serviços DIMS, devem ser pesquisadas novas técnicas de engenharia de *software* para a garantia da consistência do sistema configurado, quando da integração dos componentes/*frameworks* DIMS.

- **Alteração do escalonamento em tempo de execução:**

Investigar a possibilidade de dinamicamente modificar o escalonamento inicial para uma certa requisição, tomando-se por base condições encontradas durante a execução de cada serviço. Por exemplo, seja o caso de uma consulta global submetida ao sistema configurado definido no capítulo 6, onde os dados requeridos estejam localizados em somente uma fonte de dados. O Controle poderia dinamicamente gerar um novo escalonamento contendo apenas as operações *analisar-consulta* e *processar-consulta*. A execução das operações *otimizar-consulta* e *reescrever-consulta* poderiam ser canceladas.

Como um outro exemplo, seja o caso em que, durante a execução de uma consulta com base em um plano ótimo gerado, uma fonte de dados não possa ser acessada. Uma opção seria invocar novamente a operação *otimizar-consulta* para gerar um novo plano de execução, levando-se em conta esta nova informação.

Considerando-se estes aspectos, deve-se estender os conceitos básicos de *workflow* definidos e utilizados neste trabalho, para a gerência dinâmica dos serviços DIMS.

- **Especificação do mecanismo de troca de mensagens entre os componentes:**

Deve ser especificado um mecanismo para a troca de mensagens e de dados entre os componentes DIMS do sistema configurado, com a possibilidade de uso da linguagem *XML*.

- **Desenvolvimento e (re)uso de componentes e de *wrappers*:**

Desenvolver novas aplicações, através da especificação e implementação de novos componentes/serviços DIMS e de *wrappers* para outros tipos de fontes de dados, bem como pesquisar a existência de outros já existentes e disponíveis na rede.

Referências Bibliográficas

[BBB+97] Bayardo, R.J.; Bohrer, W.; Brice, R. et al. "**The InfoSleuth Project**". Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, pp 543-545, May 1997.

[BBC+98] Bernstein, P.; Brodie, M.; Ceri, S. et al. "**The Asilomar Report on Database Research**". SIGMOD Record, Vol. 27, No. 4, December 1998.

[BBH+00] Bouguettaya, A.; Benatallah, B.; Hendra, L. et al. "**Supporting Dynamic Interactions among Web-Based Information Sources"**. IEEE Transactions on Knowledge and Data Engineering, Vol.12, No 5, pp 779-801, September/October 2000.

[BBR+90] Batory, D.S., Barnett, J.R., Garza, J.F. et al. "**GENESIS: An Extensible Database Management System**". *Readings on Object-Oriented Database Systems*, Maier, D. & Zdonik, S. (editors), Morgan-Kaufmann, 1990.

[BE96] Bukhres, O.A. & Elmagarmid, A.K. "**Object-Oriented Multidatabase Systems – A solution for Advanced Applications**". Prentice Hall, 1996.

[BGK+97] Baumer, D.; Gryczan, G.; Knoll, R. et al. "**Framework Development for Large Systems**". Communications of the ACM, Vol. 40, No. 10, October 1997.

[Bill98] Billard, D. "**Transactional Services for the Internet**". Extending Database Technology (EDBT) Workshop on the Web and Databases, March 1998, Valencia, Spain.
http://cui.unige.ch/tios/transactions.html

[BL00] Barbosa, A.C.P. & Lucena, C.J.P. "**Integração de Frameworks de Software**". Série Monografias em Ciência da Computação, No. 02/00, PUC-Rio, Brasil, 25p, Janeiro 2000.

[BM99] Barbosa, A.C.P. & Melo, R.N. "**Usando SGBDH para Acessar e Disponibilizar Informações na WEB**". Série Monografias em Ciência da Computação, No. 29/99, PUC-Rio, Brasil, 29p, Dezembro 1999.

[BMA97] Brugali, D.; Menga, G. & Aarsten, A. "**The Framework Life Span**". Communications of the ACM, Vol. 40, No. 10, October 1997.

[BMMB98] Bosch, J.; Molin, P.; Mattsson, M. & Bengtsson, P. "**Object-Oriented Frameworks: Problems & Experiences**". John Wiley, 1998.
http://www.ipd.hk-r.se/michaelm/papers/ex-frame.ps - 24/11/1999.

[BMR+96] Buschmann, F.; Meunier, R; Rohnert, H. et al. "**Pattern Oriented Software Architecture – A System of Patterns**". John Wiley, 1996.

[Bosch98] Bosch, J. "**Specifying Frameworks and Design Patterns as Architectural Fragments**". Proceedings TOOLS ASIA'98.
http://www.ide.hk-r.se/~bosch/ - 24/11/1999.

[BP01] Barbosa, A.C.P. & Porto, F.A.M. "**Configurable Data Integration Middleware System**". Proceedings of International Workshop on Information Integration on the Web - Technologies and Applications (WIIW2001). Rio de Janeiro, Brasil, 2001.

[BR87] Biggerstaff, T. & Richter, C. "**Reusability Framework, Assessment, and Directions**". IEEE Software, Vol. 4, No. 2, March, 1987.

[BRJ00] Booch, G.; Rumbaugh, J. & Jacobson, I. "**UML – Guia do Usuário**". Editora Campus, 2000.

[BT95] Batory, D.S. & Thomas, J. "**P2: A Light Weight DBMS Generator**". Technical Report TR-95-26, Department of Computer Sciences, University of Texas at Austin, August 1995.
http://www.cs.utexas.edu - 10/04/1998.

[BT99] Barbosa, A.C.P. & Tanaka, A.K. "**SGBDH como Integrador de Informações Ambientais Heterogêneas**". Série Monografias em Ciência da Computação, No. 30/99, PUC-Rio, Brasil, 37p, Dezembro 1999.

[Catt94] Cattell, R.G.G. "**Object Data Management: Object-Oriented and Extended Relational Database Systems**". Revised edition. Addison-Wesley, 1994.

[CDG+90] Carey, M.J., DeWitt, D.J., Graefe, G. et al. "**The EXODUS Extensible DBMS Project: an Overview**". *Readings on Object-Oriented Database Systems,* Maier, D. & Zdonik, S. (editors), Morgan-Kaufmann, 1990.

[CDSV97] Codenie, W.; De Handt, K.; Steyaert, P. & Vercammen, A. "**From Custom Applications to Domais-Specific Frameworks**". Communications of the ACM, Vol. 40, No. 10, October 1997.

[CEH+97] Conrad, S.; Eaglestone, B.; Hasselbring, W. et al. "**Research Issues in Federated Database Systems"**, Report of EFDBS'97 Workshop, SIGMOD Record, Vol. 26, No. 4, December 1997.

[Cerq00] Cerqueira, R.F.G. "**Um modelo de Composição Dinâmica entre Sistemas de Componentes de Software**". Tese de Doutorado, Departamento de Informática, PUC-Rio, Brasil, 2000.

[CHS+97] Carey, M.J.; Haas, L.M.; Schwarz, P.M. et al. "**Towards Heterogeneous Multimedia Information Systems: The Garlic Approach**". The Garlic Project. http://www.almaden.ibm.com/cs/garlic/homepage.html - 16/04/2000

[Clin96] Cline, M.P. "**The Pros and Cons of Adopting and Applying Design Patterns in the Real World**". Communications of the ACM, Vol. 39, No. 10, October 1996.

[DMNS96] Demeyer, S.; Meijler, T.D.; Nierstrasz, O. & Steyaert, P. "**Design Guidelines for 'Tailorable' Frameworks**". Communications of the ACM, Vol. 99, No. 10, October 1996.

[DPSM93] Duarte, C.H.C., Pacitti, E., Silva, S.D. & Melo, R.N. "**HEROS: Um Sistema de Bancos de Dados Heterogêneos Orientado a Objetos**". VIII Simpósio Brasileiro de Banco de Dados, Campina Grande-PB, pp. 383-394, Brasil, 1993.

[DW99] D´Souza, D.F. & Wills, A.C. "**Objects, Components and Frameworks with UML: the Catalysis Approach**". Addison Wesley, 1999.

[Ecob99] Projeto Ecobase: http://www.uniriotec.br/ecobase

[EN94] Elmasri, R. & Navathe, S.B. "**Fundamentals of Database Systems**", second edition, Addison Wesley, 1994.

[Eske99a] Eskelin, P. "**Layering Frameworks in Component-Baesed Development**". Pattern Languages of Programs Conference - PLoP´99, USA, August 1999.
http://st-www.cs.uiuc.edu/~plop/plop99/proceedings/ - 21/10/1999.

[Eske99b] Eskelin, P. "**Component Interaction Patterns**". Pattern Languages of Programs Conference - PLoP´99, USA, August 1999.
http://st-www.cs.uiuc.edu/~plop/plop99/proceedings/ - 21/10/1999.

[FFK+97] Fernandez, M.; Florescu, D.; Kang, Jaewoo et al. "**STRUDEL: A Web-site Management System**". Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Arizona, USA, May 1997.

[FFL+00] Fernandez, M.; Florescu, D.; Levy, A. & Suciu, D. "**Declarative Specification of Web Sites With STRUDEL**". The VLDB Journal, No. 9, pp 38-55, 2000.

[FGL+98] Fankhauser, P.; Gardarin, G.; Lopez, M. et al. "**Experiences in Federated Databases: From IRO-DB to MIRO-Web**". Proceedings of the 24th VLDB Conference, New York, USA, 1998.

[FHB00] Fayad, M.E.; Hamu, D.S. & Brugali, D. "**Enterprise Frameworks Characteristics, Criteria, and Challenges**". Communications of the ACM, Vol. 43, No. 10, pp. 39-46, October 2000.

[Fing00] Fingar, P. "**Component-Based Frameworks for E-Commerce**". Communications of the ACM, Vol. 43, No. 10, pp. 61-66, October 2000.

[FJ99] Fayad, M.E. & Johson, R.E. "**Domain-Specific Application Frameworks – Frameworks Experience by Industry**". John Wiley & Sons, Inc. 1999.

[FLM98] Florescu, D.; Levy, A. & Mendelzon, A. "**Database Techniques for the World-Wide Web: A Survey**". SIGMOD Record, Vol. 27, No. 3, pp 59-74, September 1998.

[Font99] Fontoura, M.F.M.C. "**Uma Abordagem Sistemática para o Desenvolvimento de Frameworks**". Tese de Doutorado, Departamento de Informatica, PUC-Rio, Brasil, 1999.

[FPR00] Fontoura, M.F.; Pree, W. & Rumpe, B. "**UML-F: A Modeling Language for Object-Oriented Frameworks**". 14th European Conference on Object Oriented Programming (ECOOP 2000), Lecture Notes in Computer Science 1850, Springer, pp. 63-82, Cannes, France, 2000.
http://www.cs.princeton.edu/~fontoura/ - 27/06/2000.

[Frat99] Fraternali, P. "**Tools and Approaches for Developing Data-Intensive Web Applications: A Survey**". ACM Computing Surveys, Vol. 31, No. 3, pp. 227-263, September 1999.

[FS97] Fayad, M.E. & Schmidt, D.C. **"Object-Oriented Applications Frameworks**". Communications of the ACM, Vol. 40, No. 10, October 1997.

[FSJ99a] Fayad, M.E.; Schmidt, D.C. & Johson, R.E. "**Building Application Frameworks – Object-Oriented Foundations of Frameworks**". John Wiley & Sons, Inc. 1999.

[FSJ99b] Fayad, M.E.; Schmidt, D.C. & Johson, R.E. "**Implementing Application Frameworks – Object-Oriented Frameworks at Work**". John Wiley & Sons, Inc. 1999.

[GFF97] Gardarin, G.; Finance, B. & Fankhauser, P. "**Federating Object-Oriented and Relational Databases: The IRO-DB Experience**". Proceedings of the Second IFCIS - International Conference on Cooperative Information Systems, Kiawah Island, South Carolina, USA, pp 12-13, June 1997,

[GHJV95] Gamma, E.; Helm, R.; Johnson, R. & Vlissides, J. "**Design Patterns – Elements of Reusable Object-Oriented Software**". Addison-Wesley professional computing series, 1995.

[GKD97] Genesereth, M.; Keller, A. M. & Duschka, O. M. "**Infomaster: An Information Integration System**". Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, pp. 539-542, May 1997.

[Gogu86] Goguen, J.A. "**Reusing and Interconnecting Software Components**". Computer, Vol. 19, No. 2, February 1986.

[Hass00] Hasselbring, W. "**Information System Integration**". Communication of the ACM, Vol. 43, No. 6, pp. 33-38, June 2000.

[HFAN99] Huck, G.; Fankhauser, P.; Aberer, K. & Neuhold, E. "**Jedi: Extracting and Synthesizing Information from the Web**". http://www.darmstadt.gmd.de – 03/05/2000.

[HHW98] Hoek, A. van der; Heimbigner, D. & Wolf, A.L. "**Software Architecture, Configuration Management, and Configurable Distributed Systems: A Ménage a Trois**". Technical Report CU-CS-849-98, Department of Computer Science, University of Colorado, 1998. http://www.cs.colorado.edu/serl/arch/papers.html – 22/05/2000.

[HMN+99] Haas, L.M.; Miller, R.J.; Niswonger, B. et al. "**Transforming Heterogeneous Data with Database Middleware: Beyond Integration**". The Garlic Project. http://www.almaden.ibm.com/cs/garlic/homepage.html - 16/04/2000.

[Hopk00] Hopkins, J. "**Component Primer**". Communications of the ACM, Vol. 43, No. 10, pp. 27-30, October 2000.

[IBM93] IBM, Java Education. “**Leveraging Object-Oriented Frameworks**”. http://www.ibm.com/java/education/oobuilding/index.html - 24/11/1999.

[IBM99] IBM, Java Education. “**Building Object-Oriented Frameworks**”. http://wwwibm.com/java/education/oobuilding/index.html 24/11/1999.

[Inmo96] Inmon, W.H. “**Building the Data Warehouse**”. John Wiley & Sons Inc., 2nd. Edition, 1996.

[Java01] “**Enterprise JavaBeans**”. http://java.sun.com/products/ejb/ 10/01/2001.

[JB96] Jablonski, S. & Bussler, C. “**Workflow Management – Modeling Concepts, Architecture and Implementation**”. International Thomson Computer Press, 1996.

[John92] Johnson, R. “**Documenting Frameworks using Patterns**”. In Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'92), Canada, October 1992.

[John97] Johnson, R. “**Frameworks = (Components + Patterns)**”. Communications of the ACM, Vol. 40, No. 10, October 1997.

[Ju97] Ju, P. “**Databases on the Web – Designing and Programming for Network Access**”. M&T Books, 1997.

[Kim95] Kim, W. “**Modern Databse Systems : the Object Model, Interoperability and Beyond**”. ACM press, Addison Wesley, 1995.

[Kimb96] Kimbal, R. “**The Data Warehouse Toolkit: Practice Techniques for Building Dimensional Data Warehouses**”. John Wiley & Sons Inc., 1996.

[Koib97] Koibielus, J.G. “**Workflow Strategies**”. IDG Books Worldwide, 1997.

[Krai00] Kraiser, T. “**Serviço de Tolerância a Falhas para o Ambiente CORBA**”. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Brasil, 2000.

[Kram97] Kramer, R. “**Databases on the Web: Technologies for Federation Architectures and Case Studies**”. Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, Tucson, Arizona, pp. 503-506, May 1997.

[Krue92] Krueger, C.W. “**Software Reuse**”. ACM Computing Surveys, Vol. 24, No. 2, pp. 131-183, June 1992.

[Lars00] Larsen, G. “**Component-Based Enterprise Frameworks**”. Communications of the ACM, Vol. 43, No. 10, pp. 25-26, October 2000.

[Lawr97] Lawrence, P. “**Workflow Handbook**”. J Wiley & Sons, 1997.

[LeSe99] “**LE SELECT: a Middleware System for Publishing Autonomos and Heterogeneos Information Sources**”. INRIA, French, 1999. http://www-caravel.inria.fr/~xhumari/LeSelect/ - 19/04/2000

[Ligh99] Light, R. “**Iniciando em XML**”. Makron Books, 1999.

[Lima97] Lima, I.N. “**O Ambiente Web Banco de Dados: Funcionalidades e Arquiteturas de Integração**”. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Brasil, 1997.

[Lint97] Linthicum, D.S. “**Next Generation Middleware**”. http://www.dbmsmag.com/9709d14.html – 13/03/2001

[Lint99] Linthicum, D.S. “**Moving Forward with EAI**”. Component Strategies, Vol. 2, No. 1, pp 44-50, July, 1999.

[LMR90] Litwin, W.; Mark, L. & Roussopoulos, N. “**Interoperability of Multiple Autonomous Databases**”. ACM Computing Surveys, Vol. 22, No. 3, pp. 267-293, September 1990.

[MAM+98] Mecca, G.; Atzeni, P.; Masci, A. et al. **“The Araneus Web-Base Management System**”. ACM SIGMOD 1998. http:www.dia.uniroma3.it/Araneus/articles.html - 19/04/2000.

[Matt00] Mattson, M. “**Evolution and Composition of Object-Oriented Frameworks**”. PhD Thesis. Department of Software Engineering and Computer Science, University of Karlskrona/Ronneby, Sweden, 2000.
http://www.ipd.hk-r.se/michaelm/papers/index.html – 25/04/2000

[MB97] Mattsson, M. & Bosch, J. “**Framework Composition: Problems, Causes and Solutions**”. Proceedings TOOLS USA '97.
http://www.ipd.hk-r.se/michaelm/papers/frwkcomp.ps - 21/10/1999

[MB98] Mattsson, M. & Bosch, J. “**Frameworks as Components: A Classification of Framework Evolution**”. Proceedings of NWPER´98 Nordic Workshop on Programming Environment Research, pp. 163-174, Ronneby, Sweden, August 1998.
http://www.ipd.hk-r.se/michaelm/papers/nwper98.ps – 24/11/1999.

[Meye89] Meyer, B. “**Reusability: The Case for Object-Oriented Design**”. Software Reusability, Vol. II – Applications and Experience, Ted J. Biggerstaff and Alan J. Perlis (editors), ACM Press, 1989.

[MHI+97] Molina, H. G.; Hammer, J.; Ireland, K. et al. “**Integrating and Accessing Heterogeneous Information Sources in TSIMMIS**”. Journal of Intelligent Information Systems, Vol.8 , No. 2, pp. 177-232, March, 1997.
http://www-db.stanford.edu/tsimmis/tsimmis.html – 16/04/2000.

[Moch00] “**The MOCHA Project**”.
http://www.cs.umd.edu/projects – 25/01/01.

[Moer99] Moerkotte, G. “**Building Query Compilers” .** Draf.
www.informatik.uni-mannheim.de, 1999.

[MPLB98] Melo, R.N.; Porto, F.; Lima, F. & Barbosa, A.C.P. “**ECOHOOD: Constructing Configured DBMSs based on Frameworks**”. XIII Simpósio Brasileiro de Banco de Dados, Maringá-PR, Brasil, 1998.

[MR00] Martinez, M.R. & Roussopoulos, N. "**MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources**". ACM SIGMOD International Conference on Management of Data, Dallas, USA, 2000. http://www.cs.umd.edu/projects/mocha/ - 17/04/2000.

[MST97] Melo, R.N., Silva, D.S. & Tanaka, A.K. "**Banco de Dados em Aplicações Cliente-Servidor**". Livraria e Editora Infobook S.A., Brasil, 1997.

[ND95] Nierstrasz, O. & Dami, L. "**Component-Oriented Software Technology".** *Object-Oriented Software Composition*, Chapter 1, edited by Nierstrasz, O & Tsichritzis, D. - Prentice-Hall Inc., 1995.

[OBJS96] Object Services and Consulting, Inc. "**Wrappers**". http://www.objs.com/reports.html – 21/10/1999

[OBJS97] Object Services and Consulting, Inc. "**Web + DBMS Integration**". http://www.objs.com/reports.html – 21/10/1999

[ODMG00] Object Database Management Group http://www.odmg.org - 26/07/2000

[OGC98] Open GIS Consortium. "**The OpenGIS Guide: Introduction to Interoperable Geoprocessing and the OpenGIS Specification**". Third Edition, 1998. http://www.opengis.org - 21/10/1999

[OHE96] Orfali, R.; Harkey, D. & Edwards, J. "**The Essential Distributed Objects – Survival Guide**". John Wiley & Sons, Inc. 1996.

[Oliv97] Oliveira, E.S. "**HEROS: A Interoperabilidade de Seus Componentes Usando CORBA**". Dissertação de Mestrado, Departamento de Informática, PUC-Rio, 1997.

[OMG00] Object Management Group http://www.omg.org – 26/07/2000

[OQ97] Odenthal, G. & Quibeldey-Cirkel, K. "**Using Patterns for Design and Documentation**". http://www.ti.et-inf.uni-siegen.de /Entwurfsmuster/artikelimport/index_ecoop.htm – 05/08/1999.

[ÖV99] Özsu, M. T. & Valduriez, P. "**Principles of Distributed Database Systems**". Second Edition, Prentice-Hall Inc, 1999.

[PBE95] Pitoura, E.; Bukhres, O. & Elmagarmid, A. "**Object Orientation in Multidatabase Systems**". ACM Computing Surveys, Vol. 27, No. 2, pp. 141-195, June 1995.

[Pesc98] Pescio, C. "**Deriving Patterns from Design Principles**". The Journal of Object-Oriented Programming, Vol. 11, No. 6, October 1998.

[Picc98] Piccinini, H.S. "**Integrando Acervos da Internet/Intranet a Bancos de Dados Heterogêneos**". Dissertação de Mestrado, Departamento de Informática, PUC-Rio, 1998.

[PK99] Pree,W & Koskimies, K. "**Framelets – Small is Beautifull**". In Building Application Frameworks – Object-Oriented Foundations of Frameworks. John Wiley & Sons, Inc. 1999.

[Pree95] Pree, W. "**Design Patterns for Object-Oriented Software Development".** Addison-Wesley, 1995.

[Pree96] Pree, W. "**Framework Patterns**". SIGS Books & Multimedia, 1996.

[Pree97] Pree, W. "**Component-Based Software Development - A New Paradigm in Software Engineering?**" Software-Concepts and Tools (18), Springer-Verlag, 1997.

[Rieh00] Riehle, D. "**Framework Design – A Role Modeling Approach**". PhD Thesis. Swiss Federal Institute of Technology Zurich, Swiss, 2000.
http://www.riehle.org/diss/index.html – 25/04/2000

[RJ96] Roberts, D. & Johnson, R. "**Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks**". Pattern Languages of Programs Conference - PLoP´96, USA, September 1996.
http://st-www.cs.uiuc.edu/users/droberts/evolve.html – 21/10/1999.

[RÖH99] Roth, M.T.; Özcan, F. & Haas, L.M. "**Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System**". The Garlic Project.
http://www.almaden.ibm.com/cs/garlic/homepage.html 16/04/2000

[Ryme96] Rymer, J.R. “**The Muddle in the Middle**”. Byte, Vol. 21, No. 4, pp 67-70, April 1996

[Schm97] Schmid, H.A. “**Systematic Framework Design by Generalization**”. Communications of the ACM, Vol. 40, No. 10, October 1997.

[Schn99] Scheneider, J-G. “**Components, Scripts, and Glue: A Conceptual Framework for Software Composition**”. PhD Thesis. Institute of Computer Science and Applied Mathematics, University of Bern, Swiss, 1999. http://www.iam.unibe.ch/~scg/ - 03/05/2000

[Sebe96] Sebesta, R.W. “**Concepts of Programming Languages**”. Addison-Wesley, 1996.

[SF97] Schmidt, D.C. & Fayad, M.E**. “Building Reusable OO Frameworks for Distributed Software”**. Communications of the ACM, Vol. 40, No. 10, October 1997.

[SG96] Shaw, M. & Garlan, D. “**Software Architecture**”. Prentice-Hall, 1996.

[Shor97] The Shore Project Group: “**An Overview of Shore**”. Computer Sciences Department, University of Wisconsin, 1994-97. http://www.cs.wisc.edu/shore/doc/overview/ - 03/03/2001.

[Silv94] Silva, S.D. “**Sistemas de Bancos de Dados Heterogêneos: Modelo de Gerência de Transações**”. Tese de Doutorado, Departamento de Informática, PUC-Rio, Brasil, 1994.

[Silv99] Silva, D.S. “**Uma Arquitetura de Sistemas de Data Warehouse usando HEROS: Um Sistema de Gerência de Bancos de Dados Heterogêneos**”. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Brasil, 1999.

[Siqu99] Siqueira, S.W.M. “**Uma Arquitetura de Database Marketing utilizando o HEROS - Um SGBDH**”. Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Brasil, 1999.

[SKS99] Silberschatz, A.; Korth, H.F. & Sudarshan, S. “**Sistema de Banco de Dados**”, terceira edição, MAKRONBOOKS, 1999.

[SL90] Shet, A.P. & Larson, J.A. "**Federated Database Systems for Managing Distributed, Heterogeneous and Autonomous Databases**". ACM Computing Surveys, Vol. 22, No. 3, pp. 183-236, September 1990.

[SM98] Schmid, A.H. & Mueller, F. "**Patterns for Extending Black-Box Frameworks**". The Journal of Object-Oriented Programming, Vol. 11, No. 3, June 1998.

[Srin97] Srinivasan, U. "**A Framework for Conceptual Integration of Heterogeneous Databases**". PhD Thesis. School of Computer Science and Engineering University, of New South Wales, Sydney, Austrália, 1997.
http://www.library.unsw.edu.au/~thesis/adt-NUN/public -15/04/2000

[SSU96] Silberschatz, A.; Stonebraker, M. & Ullman, J. "**Database Research: Achievements and Opportunities Into the 21st Century**". SIGMOD Record, Vol. 25, No. 1, pp. 52-63, March 1996.

[Stev94] Stevens, W.R. "**TCP/IP Illustrared, Volume 1 – The Protocols**". Addison-Wesley, 1994.

[SZ97] Silberschatz, A. & Zdonic, S. "**Database Systems – Breaking Out the Box**". SIGMOD Record, Vol. 26, No. 3, September 1997.

[Szyp98] Szyperski, C. "**Component Software – Beyond Object-Oriented Programming**". Addison-Wesley, 1998.

[TAB+97] Tomasic, A.; Amouroux, R.; Bonnet, P.; Kapitskaia, O.; Naacke, H. & Raschid, L. "**The Distributed Information Search Component (Disco) and the World Wide Web**". Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data, pp. 546-548, Tucson, Arizona, May 1997.

[TRV98] Tomasic, A.; Raschid, L. & Valduriez, P. "**Scaling Access to Heterogeneous Data Source with DISCO**". IEEE Transactions on Knowledge and Data Engineering, Vol. 10, No. 5, pp. 808-823, September 1998.

[Tsim99] "**The TSIMMIS Project**". http://www-db.stanford.edu/ - 15/09/99

[TTC+90] Thomas, G.; Thompson, G.R.; Chung, C.-W.; Barkmeyer, E.; Carter, F.; Templeton, M.; Fox, S. & Hartman, B. "**Heterogeneous Distributed Database System for Production Use**". ACM Computing Surveys, Vol. 22, No. 3, Sept. 1990, pages 237-265.

[Ucho94] Uchôa, E.M.A. "**HEROS – Um Sistema de Bancos de Dados Heterogêneos: Integrando Esquemas**". Dissertação de Mestrado, Departamento de Informática, PUC-Rio, Brasil, 1994.

[Ucho99] Uchôa, E.M.A. "**Framework para Integração de Sistemas de Bancos de Dados Heterogêneos**". Tese de Doutorado, Departamento de Informática, PUC-Rio, Brasil, 1999.

[UML96] Uchôa, E.M.A.; Melo, R.N. & Lifschitz, S. "**Interoperabilidade em Sistemas de Bancos de Dados Heterogêneos**". Monografia (PUC-RioInf.MCC45/96). PUC-Rio, Brasil, 1996, 31p.

[USM94] Uchôa, E.M.A.; Silva, S.D. & Melo, R.N. "**Modelo de Dados do HEROS - Sistema de Bancos de Dados Heterogêneos Orientado a Objetos**". IX Simpósio Brasileiro de Banco de Dados, São Carlos-SP, Brasil, 1994.

[WBT92] Wells, D.L.; Blakeley, A. & Thompson, C.W. "**Architecture of an Open Object-Oriented Database Management System**". IEEE Computer, October 1992.

[WC96] Widom, J. & Ceri, S., editors. "**Active Database Systems - Triggers and Rules For Advanced Database Processing**". Morgan-Kaufmann, 1996.

[WG97] Wiederhold, G. & Genesereth, M. "**The basis for Mediation**". http://WWW-DB.Stanford.edu/pub/gio/paperlist.html- 24/11/1999.

[Wied98] Wiederhold, G. "**Value-added Middleware: Mediators**". http://WWW-DB.Stanford.edu/pub/gio/paperlist.html - 24/11/1999.

[Wied99] Wiederhold, G. "**Mediation to Deal with Heterogeneous Data Sources**". http://www-db.stanford.edu/pub/gio/1999/Interopdoc.html - 24/11/1999.

[WN95] Waldén, K. & Nerson, J-M. “**Seamless Object-Oriented Software Architecture – Analysis and Design of Reliable Systems**”. Prentice Hall, 1995.

[XML01] “**Extensible Markup Language** ”. http://w3.org/XML/ - 17/01/2001.

[YM98] Yu, C. & Meng, W. “**Principles of Database Query Processing for Advanced Applications** ”. Morgan Kaufmann Publishers, Inc, 1998.

ANEXO A

Neste anexo são descritas as operações básicas que são oferecidas pelos componentes do CoDIMS.

A1 – Componente Controle

- **definir-configuração**: define a configuração do sistema, com a definição de cada um dos componentes que farão parte da configuração. Ao final é executada uma operação para a verificação da configuração.

⇒ Parâmetros de entrada: arquivo de *script* de configuração

⇒ Parâmetros de saída: código de retorno

- **verificar-configuração:** após a definição dos componentes, verifica se todas as operações requisitadas por cada componente estão sendo oferecidas pelos outros.

⇒ Parâmetros de entrada: (não há)
⇒ Parâmetros de saída: código de retorno

- **exibir-configuração**: retorna informações sobre os componentes declarados na configuração.

⇒ Parâmetros de entrada: (não há)
⇒ Parâmetros de saída: arquivo de configuração
código de retorno

- **definir-workflow**: define o *workflow* para os comandos da linguagem do sistema, com a seqüência das tarefas (operações) a serem executadas. É verificado se as operações estão definidas na configuração do sistema.

⇒ Parâmetros de entrada: arquivo de *script* de *workflow*
⇒ Parâmetros de saída: código de retorno

- **verificar-operação**: verifica se uma dada operação incluída no *workflow* foi definida como sendo oferecida por um componente.

⇒ Parâmetros de entrada: nome do componente
nome da operação
⇒ Parâmetros de saída: código de retorno

- **exibir-workflow**: retorna informações sobre o *workflow* dos comandos da linguagem do sistema.

⇒ Parâmetros de entrada: (não há)
⇒ Parâmetros de saída: arquivo de workflow
código de retorno

- **obter-workflow-do-comando**: retorna *workflow* do comando solicitado.

⇒ Parâmetros de entrada: nome do comando
⇒ Parâmetros de saída: *string* (*workflow* do comando)
código de retorno

- **definir-metadados**: é através desta operação que o projetista da aplicação define os metadados do sistema configurado.

⇒ Parâmetros de entrada: tipo metadados (global, exportação, externo)
arquivo de *script* de metadados
⇒ Parâmetros de saída: código de retorno

- **exibir-metadados**: retorna informações sobre os metadados definidos no sistema.

⇒ Parâmetros de entrada: tipo metadados (global, exportação, externo)
⇒ Parâmetros de saída: arquivo de metadados
código de retorno

- **enviar-mensagem**: é através desta operação que os componentes se comunicam com os outros.

⇒ Parâmetros de entrada: identificador da consulta
componente destino
operação destino
string (dados de entrada da operação)

⇒ Parâmetros de saída: *string* (dados de retorno)
código de retorno

- **executar-consulta**: é através desta operação que a aplicação cliente submete uma consulta para execução.

⇒ Parâmetros de entrada: *string* (consulta)

⇒ Parâmetros de saída: *string* (nome do arquivo resultado)
código de retorno

- **atualizar-escalonamento**: atualiza o *workflow* quando do término de uma tarefa.

⇒ Parâmetros de entrada: nome da operação encerrada

⇒ Parâmetros de saída: código de retorno

- **obter-próxima-tarefa**: pesquisa o *workflow* para identificar a próxima tarefa a ser executada.

⇒ Parâmetros de entrada: (não há)

⇒ Parâmetros de saída: nome da próxima operação
código de retorno

- **executar-serviço**: é através desta operação que o escalonamento passa ao Controle a operação a ser executada.

⇒ Parâmetros de entrada: nome do componente
nome da operação
string (dados)

⇒ Parâmetros de saída: *string* (resultado)
código de retorno

A2 – Componente Metadados

- **definir-MD-exportação**: define cada um dos metadados de exportação das fontes de dados.

⇒ Parâmetros de entrada: arquivo de *script* definindo os metadados de exportação das fontes de dados.

⇒ Parâmetros de saída: código de retorno

- **definir-MD-gobal**: define o metadados global.

⇒ Parâmetros de entrada: arquivo de *script* definindo os metadados.

⇒ Parâmetros de saída: código de retorno

- **definir-MD-externo**: define cada visão externa do metadados global.

⇒ Parâmetros de entrada: arquivo de *script* definindo os nomes das diferentes visões (metadados externos).

⇒ Parâmetros de saída: código de retorno

- **exibir-MD-exportação**: retorna os metadados de exportação das fontes de dados.

⇒ Parâmetros de entrada: não há.

⇒ Parâmetros de saída: arquivo de metadados
código de retorno

- **exibir-MD-externo**: retorna as visões externas do metadados externo.

⇒ Parâmetros de entrada: não há.
⇒ Parâmetros de saída: arquivo de metadados
código de retorno

- **exibir-MD-global**: retorna o metadados global.

⇒ Parâmetros de entrada: não há.
⇒ Parâmetros de saída: arquivo de metadados
código de retorno

- **obter-objeto-MD-exportação**: retorna o objeto especificado do metadados de exportação.

⇒ Parâmetros de entrada: identificação da consulta
nome do objeto
⇒ Parâmetros de saída: string (objeto)
código de retorno

- **obter-objeto-MD-externo**: retorna o objeto especificado de um metadados externo.

⇒ Parâmetros de entrada: identificação da consulta
nome do objeto
⇒ Parâmetros de saída: *string* (objeto)
código de retorno

- **obter-objeto-MD-global**: retorna o objeto especificado do metadados global.

⇒ Parâmetros de entrada: identificação da consulta
nome do objeto
⇒ Parâmetros de saída: *string* (objeto)
código de retorno

A3 – Componente Processamento de Consulta

- **analisar-consulta**: é através desta operação que uma consulta é enviada para a fase de análise.

⇒ Parâmetros de entrada: identificação da consulta
string (consulta)
⇒ Parâmetros de saída: *string* (grafo de consulta)
código de retorno

- **reescrever-consulta**: é através desta operação que uma consulta é re-escrita.

⇒ Parâmetros de entrada: identificação da consulta
string (grafo da consulta)
⇒ Parâmetros de saída: *string* (árvore de operação)
código de retorno

- **otimizar-consulta**: é através desta operação que uma consulta é otimizada antes de sua execução.

⇒ Parâmetros de entrada: identificação da consulta
string (árvore de operação)
⇒ Parâmetros de saída: *string* (árvore de execução)
código de retorno

- **processar-consulta**: é através desta operação que uma consulta passa a ser executada.

⇒ Parâmetros de entrada: identificação da consulta
string (árvore de execução otimizada da consulta)
⇒ Parâmetros de saída: *string* (arquivo resultado)
código de retorno

- **enviar-mensagem**: envia mensagem ao Controle para ser encaminhada ao componente destino.

⇒ Parâmetros de entrada: identificação da consulta
componente destino
operação destino
componente origem
operação origem
string (dados de entrada da operação)

⇒ Parâmetros de saída: *string* (dados de retorno)
código de retorno

A4 – Componente Acesso aos Dados

- **executar-sub-consulta**: recebe a sub-consulta a ser executada e realiza os procedimentos necessários à sua execução.

⇒ Parâmetros de entrada: identificação da sub-consulta
nome da fonte de dados
string (sub-consulta)

⇒ Parâmetros de saída: *string* (arquivo resultado)
código de retorno

- **obter-estado-FD**: retorna estado da fonte de dados.

⇒ Parâmetros de entrada: nome da fonte de dados

⇒ Parâmetros de saída: estado da fonte de dados.
código de retorno

- **obter-estado-sub-consulta**: retorna estado da sub-consulta.

⇒ Parâmetros de entrada: identificação da sub-consulta
nome da fonte de dados

⇒ Parâmetros de saída: estado da sub-consulta
código de retorno

- **obter-próximo-dado**: fazer a leitura do próximo dado do arquivo resultado.

⇒ Parâmetros de entrada: identificação da sub-consulta
nome da fonte de dados

⇒ Parâmetros de saída: *string* (dados)
código de retorno

- **abrir-FD**: estabelece conexão com a fonte de dados para permitir leitura.

⇒ Parâmetros de entrada: nome da fonte de dados

⇒ Parâmetros de saída: código de retorno

- **fechar-FD**: encerrar conexão com a fonte de dados.

⇒ Parâmetros de entrada: nome da fonte de dados

⇒ Parâmetros de saída: código de retorno

- **modificar-estado-FD**: modifica o estado da fonte de dados.

⇒ Parâmetros de entrada: novo estado

⇒ Parâmetros de saída: código de retorno

- **modificar-estado-sub-consulta**: modifica o estado da sub-consulta.

⇒ Parâmetros de entrada: novo estado

⇒ Parâmetros de saída: código de retorno