UNIVERSIDADE FEDERAL DA PARAÍBA - UFPB
CENTRO DE CIÊNCIAS E TECNOLOGIA - CCT
DEPARTAMENTO DE SISTEMAS E COMPUTAÇÃO - DSC
COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA - COPIN

DISSERTAÇÃO DE MESTRADO

ANÁLISE, PROJETO E IMPLEMENTAÇÃO DE UM GERENTE WEB DE METADADOS EXTENSÍVEL PARA UM PROCESSO DE *DATA WAREHOUSING*

Methanias Colaço Rodrigues Júnior

(Mestrando)

Prof. Dr. Marcus Costa Sampaio

(Orientador)

CAMPINA GRANDE, JUNHO DE 2002

JÚNIOR, Methanias Colaço Rodrigues Júnior

N244I

Análise Projeto e Implementação de um Gerente Web de Metadados Extensível para um Processo de *Data Warehousing*.

Dissertação de Mestrado, Universidade Federal da Paraíba, Centro de Ciências e Tecnologia, Coordenação de Pós-Graduação em Informática, Campina Grande, Paraíba, Junho de 2002.

145p. Il.

Orientador: Dr. Marcus Costa Sampaio

1. Banco de Dados
2. Data Warehousing
3. Metadados
4. *Framework*

CDU - 681.3.07B

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, pelas constantes bênçãos derramadas na minha vida.

Agradecimentos especiais a todos da minha família e ao meu orientador, Doutor Marcus Costa Sampaio.

Ao professor Doutor Jacques Sauvé pelos conhecimentos transmitidos e por mostrar no início do curso o verdadeiro caminho que um aluno de mestrado deve seguir.

Ao professor Doutor Marcus Costa Sampaio pela notável eloqüência, conhecimento profundo da minha área de concentração e conseqüente excelente orientação prestada.

À funcionária da Copin, Ana, a famosa Aninha, pelo dinamismo e atenção.

À minha ex-aluna e agora para minha satisfação, colega de profissão e mestrado, Maria de Fátima, pela amizade e companhia nas madrugadas de estudos.

Ao professor Doutor Asterio Tanaka, pelo grande incentivo e confiança em mim depositados.

Resumo

Sistemas de Apoio à Decisão baseados em *Data Warehouses* possuem um componente crítico: os metadados. Conhecidos como “dados sobre dados”, os metadados são necessários para o entendimento dos dados através do tempo e, principalmente, constituem o instrumental necessário para transformá-los em conhecimento. Como as organizações cresceram, os problemas com metadados se multiplicaram. Entretanto, os próprios problemas com metadados apresentam boas oportunidades para os fabricantes de software. Isto é mais evidente em ambientes de *Data Warehouses* corporativos, pois os usuários participam do processo de tomada de decisão das empresas, o que exige o conhecimento de toda a semântica de formação dos dados. Este trabalho apresenta a análise, projeto e implementação de um Gerente de Metadados Extensível — Metha Manager. O Metha Manager é um *framework* para construção e uso de um Repositório de Metadados para *Data Warehouses*, e tem como característica principal a extensibilidade dos seus metadados para prover informações às diversas ferramentas que participam de um processo de *Data Warehousing*.

Abstract

Decision Support Systems based in Data Warehouses has a critical component: the metadata. Known as “information about data”, metadata are required for understanding of data within time and, mainly, make the necessary instrument to turn them into knowledge. As organizations grew up, the problem with metadata has multiplied itself. However, metadata own problem presents good opportunity for software developers. In fact, this is more evident in the corporate Data Warehouse environment, because the users work on the process of taking companies decision, that acquire knowledge about all semantic involved in data’s formation. This work presents the analyses, design and implementation, of an Extensible Metadata Manager – The Metha Manager. The Metha Manager is a *framework* built for construction and use of a Metadata Repository for Data Warehouses, and it has, as main characteristic, the extensibility of its metadata to provide information for different types of tools that work on the process of Data Warehousing.

“ A chave para ter sucesso nos negócios é ter informações que ninguém mais tem. ”

Aristóteles Onassis

Sumário

Índice de Figuras

Capítulo 1

Introdução

Devido à grande quantidade de informações que circulam dentro de uma empresa, foi necessário aprimorar e descobrir novas formas do uso de bancos de dados. O conceito de *Data Warehouse* — DW — surgiu principalmente para integrar essas informações entre diferentes ambientes dentro da empresa [Inmon, 1997]. Adicionalmente, para viabilizar a tecnologia, adota-se Metadados – dados sobre dados – para prover um maior detalhamento e profundidade nas consultas realizadas em um DW. Usuários exigem, cada vez mais, melhores e mais rápidas condições de acesso a dados, uma documentação completa do que aconteceu e deixou de acontecer na empresa, uma tecnologia que seja capaz de armazenar e tratar informações de forma inteligente e, muitas vezes, um histórico de acontecimentos passados, envolvendo pessoas e lugares de diferentes tipos [Sachdeva, 1999].

Os metadados são definidos como dados sobre dados, porém a complexidade desses dados no *Data Warehouse* aumenta muito. Num sistema OLTP gera-se documentos somente sobre o levantamento dos dados, banco de dados e o sistema que alimenta o mesmo. No *Data Warehouse* além do banco, gera-se uma documentação muito maior. Além de descrever sobre o levantamento de dados e o banco, temos ainda o levantamento dos relatórios a serem gerados, de onde vem os dados para alimentar o DW, processos de extração, tratamento e rotinas de carga dos dados. Ainda podem ser gerados metadados sobre as regras de negócio da empresa e todas as mudanças que elas podem ter sofrido, e também a frequência de acesso aos dados. Os metadados englobam o DW e mantêm as informações sobre localização dos dados. São exemplos de metadados para um DW:

- A estrutura dos dados segundo a visão do programador;
- A estrutura dos dados segundo a visão dos analistas de negócio;
- A fonte de dados que alimenta o DW;

- A transformação sofrida pelos dados no momento de sua migração para o DW;
- O modelo de dados;
- O relacionamento entre o modelo de dados e o DW;
- O histórico das extrações de dados.

Acrescentamos ainda os dados referentes aos relatórios que são gerados pelas ferramentas OLAP, assim como os que são gerados nas camadas semânticas.

Os metadados podem surgir de vários locais durante o decorrer do projeto. Eles provêm de repositórios de ferramentas de modelagem, os quais geralmente já estão estruturados, facilitando a integração da origem dos metadados e um repositório dos mesmos. Essa fonte de metadados é riquíssima. Outros dados que devem ser considerados metadados são os materiais que surgirão das entrevistas com os usuários. Destas entrevistas podem obter-se informações preciosas que não estão documentadas em nenhum outro lugar além de regras para validação dos dados depois de carregados no *Data Warehouse*.

As pesquisas na área de metadados para DW estão aumentando em função da necessidade extrema das organizações de conhecer melhor os dados que elas mantêm, e também conhecer com mais detalhes os dados de outras organizações, através de intranets e extranets. Sem uma documentação eficiente dos dados, é dificultada aos usuários a localização de dados necessários para suas aplicações. Também, os dados ficam sujeitos à superposição de esforços de coleta e manutenção, e vulneráveis a problemas de inconsistências. O não uso ou uso impróprio da informação será altíssimo [Sherman, 1997].

Dentro do contexto de DW podemos dividir os metadados em duas categorias básicas: *metadados técnicos* e *metadados de negócio*.

Metadados técnicos provêem confiabilidade na acuracidade do DW para os desenvolvedores. Descrevem os dados necessários pelas várias ferramentas utilizadas para armazenar, manipular ou movimentar dados. São altamente estruturados e geralmente tratados via uma ferramenta de repositório. Adicionalmente, os metadados técnicos suportam a manutenção do DW permitindo sobretudo fácil escalabilidade ao mesmo. Temos como exemplos de metadados técnicos:

- Controles de Auditoria;
- Nomes das tabelas do DW (em caso de DW relacional).

Metadados de negócio perfazem a junção entre o mundo codificado de um DW e o mundo do usuário de negócios. Este tipo de metadado permite facilitar a utilização do DW pelos usuários, auxiliando-os em suas consultas *ad hoc*. São tanto não-estruturados quanto estruturados, tornando-os difíceis de serem tratados e integrados por uma ferramenta estruturada tipo repositório. Podemos citar como exemplos de metadados de negócio:

- Mapeamento dos campos das tabelas de um DW relacional para atributos conceituais;
- Regras para operações de DW *drill-down*, *drill-up* e *drill-across*, conhecidas como operações OLAP ("*On-Line Analytical Processing*");
- Informações sobre sumários e transformações de dados;
- Nomenclaturas para os dados que possam ser facilmente entendidas pelos analistas de negócios ou executivos.

Existem diversos problemas a serem enfrentados com o uso e gerência de metadados em ambientes de DW. Em primeiro lugar, é necessário delimitar o escopo de atuação, pois a decisão de quais metadados devem ser coletados e mantidos é complexa [Marco, 2000]. Os principais problemas pertinentes à gerência de metadados estão relacionados com:

- Redundância: diversas ferramentas de um ambiente de DW ainda armazenam seus próprios metadados em formato proprietário;
- Séries históricas: o conceito de um dado pode evoluir, ou o ramo estratégico do negócio pode mudar completamente;
- Falta de estruturação de alguns metadados: aquisição e transferência de metadados de negócios não-estruturados que contextualizam eventos externos e internos à empresa, os quais provocam discrepâncias nas informações estratégicas. Residem em agendas particulares, e-mails, mídias alternativas, imagens e etc;

- Soluções particulares: em se tratando de metadados de negócios, não existe uma solução única;

- Suporte a usuários de diferentes perfis: diferentes comunidades irão propor, projetar e ser responsáveis por diferentes tipos de metadados;

- Criação de novos conjuntos de metadados (evolução do DW);

- Interoperabilidade de diferentes sistemas.

1.1 Objetivos da Dissertação

O principal objetivo deste trabalho é a criação de um Gerente de Metadados Extensível, Metha Manager para *Data Warehouses* e validação do mesmo através da provisão de metadados para um Servidor OLAP Extensível, Edwards [Jorge, 2001] e para uma Interface para Consultas a *Data Warehouses* Baseada em Linguagem Natural [Nascimento, 2001].

Edwards é baseado na tecnologia de *framework*s para a construção de software. Um software assim construído é extensível e reusável. No Edwards, a título de ilustração, uma ferramenta OLAP é facilmente personalizada para qualquer tecnologia de armazenamento e gerência de dados (SGBDR's, SBDOR's, SGBDOO's e etc.).

O coração do *framework* Servidor OLAP Extensível é o seu repositório de metadados (Figura 1.1). Na versão atual do servidor, o repositório de metadados é um conjunto de classes Java e coleções de objetos segundo essas classes. Entretanto, os objetos não são persistentes, e inexiste uma verdadeira interface para atender aos diferentes tipos de usuário que precisam fazer consultas a metadados. Em resumo, o *framework* Servidor OLAP Extensível necessita de um verdadeiro gerente de metadados persistentes.

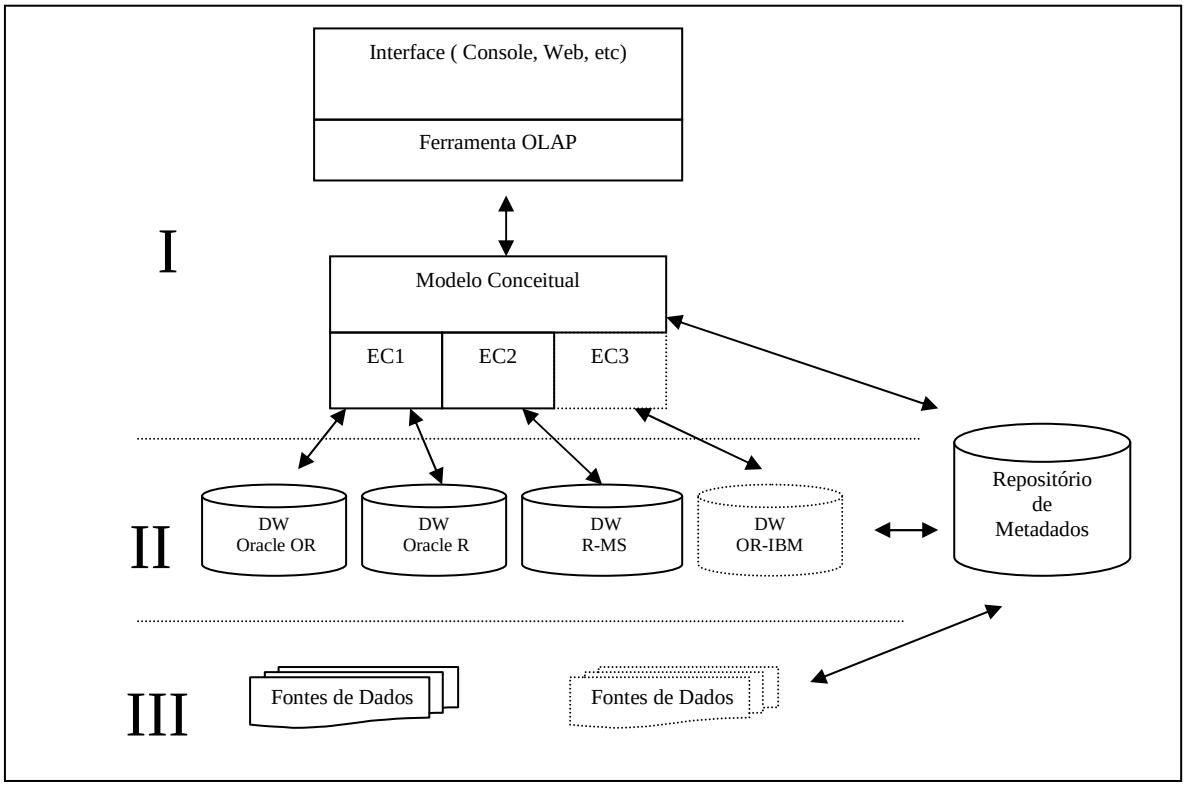

Figura 1.1: Arquitetura em 3 camadas do Servidor OLAP Extensível [Jorge, 2001]

A primeira camada (Camada Conceitual - I) diz respeito ao esquema conceitual do DW, bem como inclui todas as ferramentas e aplicações utilizadas pelos usuários para análise dos dados (OLAP) e navegação em metadados.

A segunda camada (Camada de Implementação - II) consiste nos modelos lógico e físico do DW, dependências entre modelos lógico e físico e informações sobre o processo de movimentação de dados dos sistemas legados ou fontes de dados para o DW.

A terceira camada (Camada Operacional - III) contém todos os sistemas envolvidos no gerenciamento dos dados operacionais: informações sobre tabelas e estrutura de registros, restrições, gatilhos etc.

Para uma maior validação do presente trabalho, além de prover metadados para o Servidor Edwards, o Metha Manager proverá metadados para uma Interface para Consultas a *Data Warehouses* Baseada em Linguagem Natural [Nascimento, 2001]. Esta arquitetura de interface em linguagem natural é uma arquitetura voltada para acesso a bancos de dados relacionais, e serve como meio de obtenção de informações a níveis de pesquisa ou gerencial para usuários casuais que não tenham conhecimento de linguagens como *SQL* e que não

sejam familiarizados com as atuais interfaces visuais de consulta. Através de uma abordagem com linguagem natural restrita, apoiada por metadados e diretamente relacionada com uma camada conceitual multidimensional, a arquitetura visa atender, também, a requisitos como transportabilidade entre domínios e etc. [Nascimento, 2001].

O Gerente de Metadados visa contemplar os três níveis de metadados abordados pelo Servidor Edwards, capturando informações necessárias para a análise, o projeto, a construção e a operação de um DW, bem como possuir interfaces personalizadas para os diferentes usuários (usuários finais, projetistas de DW, programadores de DW e administradores de banco de dados). A arquitetura do Gerente será melhor discutida no capítulo 5 e possuirá 3 camadas básicas:

- Camada de Negócio: Abrange os metadados de negócio;
- Camada Semântica: Composta de metadados conceituais;
- Camada Técnica: Abrangendo metadados técnicos tais como metadados do modelo lógico e físico do DW e metadados operacionais.

O Metha Manager deverá ser implantado no Banco do Estado de Sergipe, para auxiliar os analistas, programadores e usuários do seu DW através do armazenamento e gerência de metadados inexistentes até a finalização deste trabalho.

1.2 Relevância da Dissertação

As principais contribuições do nosso trabalho sobre gerência de metadados são:

- Extensibilidade de metadados conceituais e técnicos;
- Provisão de metadados para modelos de DW implementados com diversas tecnologias, incluindo as tecnologias relacional, objeto-relacional, orientada a objeto, e a tecnologia baseada em *arrays* multidimensionais;
- Interfaces adaptadas para os diferentes usuários do gerente de metadados (usuários finais, projetistas de DW, programadores de DW e administradores de banco de dados).

1.3 Estrutura da Dissertação

No capítulo 2, apresentamos o domínio dos Sistemas de Suporte à Decisão baseados em *Data Warehouses* com ênfase na importância do gerenciameto dos diversos metadados presentes em sistemas deste tipo.

No capítulo 3, discutimos todo o contexto que envolve uma arquitertura para gerência e armazenamento de metadados. Analisamos os requisitos de uma boa arquitetura, o processo de troca de metadados através de XML e a possibilidade de armazenamento de metadados através da tecnologia objeto-relacional.

No capítulo 4, abordamos o processo de construção de software baseado em *framework* utilizado no projeto apresentado nos capítulos posteriores.

No capítulo 5, descrevemos os requisitos e análise do Metha Manager.

No capítulo 6, é feita a descrição do projeto e implementação do Metha Manager.

Finalmente, no capítulo 7, concluímos o trabalho de dissertação e identificamos possíveis trabalhos futuros.

Capítulo 2

Metadados em um Processo de *Data Warehousing*

2.1 Sistemas de Suporte à Decisão Baseados em *Data Warehouses*

Paulatinamente ao longo das três últimas décadas, os Sistemas de Tecnologia da Informação(STI) têm se preocupado muito com problemas de negócios. Esta preocupação reside na necessidade de competição das empresas no mercado globalizado. As organizações devem ser capazes de analisar os dados disponíveis e tomar decisões rápidas e seguras. Para que esse grande volume de dados seja analisado, é preciso que haja um mecanismo apropriado para unificar as informações, já que estas vêm de diversas aplicações e com uma grande heterogeneidade.

Mesmo com a tendência natural de crescimento da integração entre aplicações operacionais, decisões de nível estratégico e tático exigem um conteúdo mais rico do que aquele encontrado no ambiente operacional, o qual apresenta inúmeros obstáculos para o processamento analítico [Chaudhuri & Dayal, 1997]. As empresas precisam de um ambiente exclusivo que armazene adequadamente os dados extraídos das diversas bases, disponibilizando as informações a qualquer instante. Este ambiente, que surgiu como solução para prover informações gerenciais para a tomada de decisões, foi denominado de "ambiente projetado de *Data Warehouse*" [Inmon, 1997] [Kimball, 1996].

A arquitetura padrão de um *Data Warehouse* funciona resumidamente da seguinte maneira [Inmon, 1997]: As informações vêm das várias fontes de aplicações e geralmente são tratadas por uma ferramenta ETL [1] — ferramenta responsável pela extração, transformação e carregamento dos dados no DW. Dois tipos de metadado são necessários. *Metadados operacionais* – pertencentes ao conjunto dos chamados metadados técnicos (ver introdução)

[1] *Extraction, Transformation and Load*

e que têm como funções básicas facilitar o trabalho das ferramentas ETL e auxiliar programadores de aplicações OLAP (ver seção 2.5) através da descrição do esquema de um DW, e *metadados de negócio* – responsáveis por descrever como as informações devem ser vistas pelos usuários do DW (Figura 2.1). O fluxo de dados começa nas aplicações fontes, e passa por uma *Staging Area* — área intermediária onde os dados sofrem integração e limpeza e eventual exportação para o DW. A integração consiste na consolidação dos dados de diversas origens, o que geralmente envolve diferentes codificações. Então os dados devem ser perfeitamente integrados para que ao serem armazenados assumam uma única convenção. A limpeza é a rejeição de valores inválidos, chaves repetidas ou registros com outros tipos de erro. Estas ações constituem a tarefa mais crítica na geração de um Data Warehouse, e o resultado destes processos, gera o ODS (*Operational Data Store)*. Em resumo, o ODS é uma base de dados com utilização previsível, parcialmente estruturada e analítica cujo histórico é de apenas 30 ou 60 dias e cujas informações estão organizadas por área de negócio. É um retrato da base obtida da extração, tranformação e limpeza de dados dos sistemas fontes operacionais da empresa e no início de sua concepção era visto como sendo um tipo de DW [Kimball *et al*., 1998].

Não existem ferramentas, hoje, capazes de fazer estes processos ETL automaticamente, por isso num projeto de construção de um *Data Warehouse*, consomem mais de 70% do tempo do projeto. Todo este processo normalmente é desenvolvido especificamente para cada empresa, devido à diversidade existente em termos de estruturas de dados nos sistemas fontes operacionais e também a falta de conhecimento e documentação dos mesmos. Após todos estes passos o ODS estará gerado.

Somente após a integração e limpeza os dados são exportados para o DW. Depois os dados são transmitidos para um *Data Mart* [2](ver seção 2.2) ou, numa abordagem centralizada, são consultados diretamente pelos usuários através de uma ferramenta OLAP, por exemplo. O *Data Mart* é geralmente descrito como um subconjunto dos dados contidos em um *Data Warehouse* extraído para um ambiente separado. *Data Marts* são muito úteis nas seguintes condições:

[2] O *Data Mart*, também conhecido como *Warehouse* Departamental, é uma abordagem descentralizada do conceito de *Data Warehouse* [Kimball *et al*., 1998].

- Os dados devem estar segregados para melhorar o desempenho do sistema do ponto de vista do usuário;

- Deve existir uma cópia dos dados onde só pessoas com autorização devem ter o privilégio de acessá-las;

- Em um ambiente corporativo, é importante fortalecer o conceito de propriedade dentro do banco de dados. Diferentes setores serão responsáveis por diferentes *Data Marts*.

Esses ambientes fisicamente distintos trazem benefícios, mas existe um preço a se pagar. Com a presença de muitos *Data Marts* pode haver o risco de redundância. A construção de *Data Marts* deve ter sempre a preocupação de compartilhamento de dados, tabelas e relatórios em comum entre os departamentos. A dificuldade de evitar a redundância de dados pode ir contra o paradigma de um *Data Warehouse* já que a separação física em diferentes grupos diminui essa habilidade de organização. Fica clara a necessidade de preservação da consistência das informações presentes nos *Data Marts* através da eliminação de redundâncias, pois relatórios em comum não podem possuir valores diferentes. Isto é uma característica da maioria dos sistemas legados operacionais das corporações e deve ser eliminada com a presença de um DW.

Em se tratando de metadados, a preservação da consistência é mais complicada, já que a interpretação dos dados é feita de maneira diferente em cada grupo, pois cada departamento tem suas próprias regras e conceitos [Marco, 2000]. Em um ambiente bancário, por exemplo, podemos ter significados diferentes para atributos compartilhados entre a área de recursos humanos e a área de contabilidade [Banese, 2000]. Um gerente de metadados deve ser capaz de armazenar as diversas interpretações dadas ao atributo, identificando os responsáveis pelas mesmas.

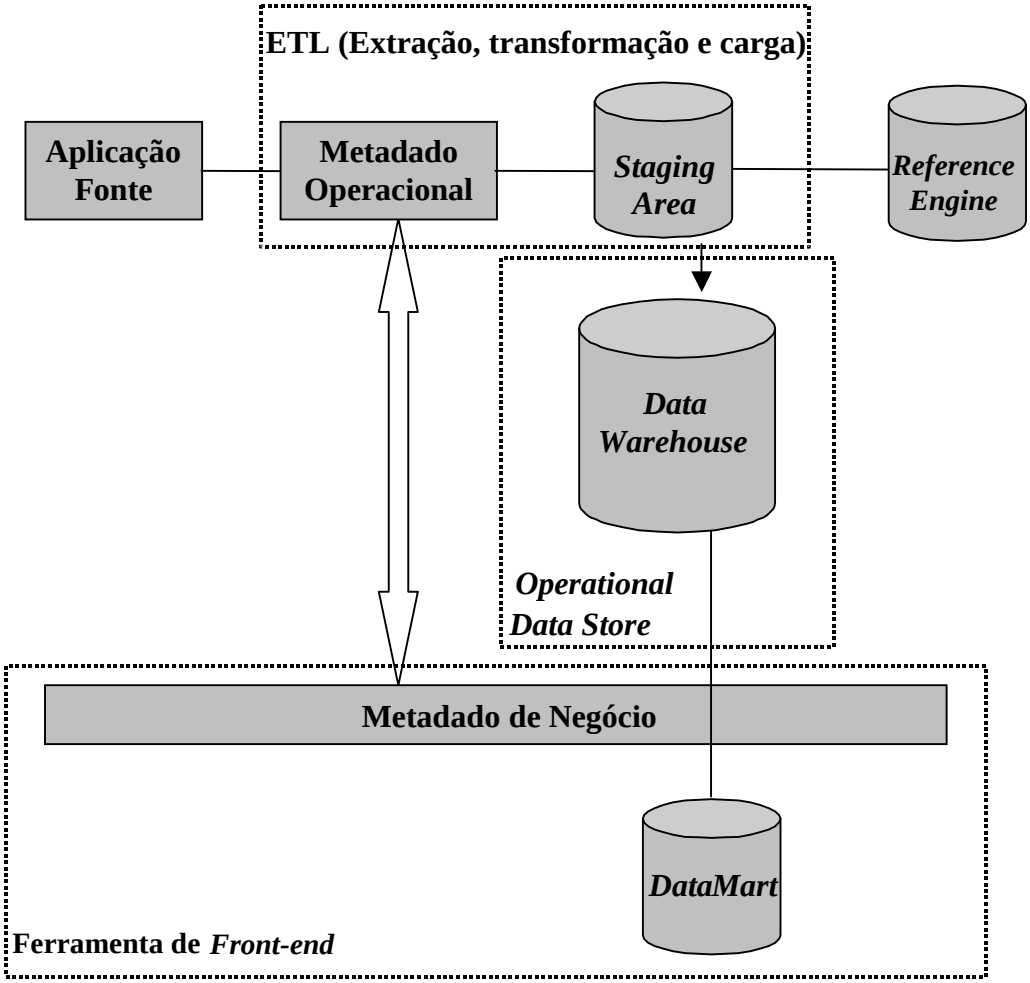

Figura 2.1: Uma Arquitetura Genérica de *Data Warehousing* para o Contexto de Metadados Operacionais e de Negócio

2.2 *Data Marts e Federated Data Warehouse*

Como um projeto de DW conceitualmente pressupõe uma arquitetura centralizada, sua implementação não é uma tarefa fácil. A implementação de um DW completo requer uma metodologia rigorosa e uma completa compreensão dos negócios da empresa. Esta abordagem pode ser longa e dispendiosa e por isto sua implementação exige um planejamento bem detalhado (em outras palavras: tempo longo). Neste contexto e com a necessidade de agilização de implantação dos projetos de DW, o *Data Mart* passou a ser uma opção de arquitetura interessante, pois os mesmos podem ser desenvolvidos para cada área da corporação num processo paulatino e gradativo. Em um projeto de DW, a cada desenvolvimento de um novo *Data Mart* a equipe adquire experiência e elimina os erros de implementações anteriores [Banese, 2000].

A tecnologia usada tanto no DW como no *Data Mart* é a mesma, as variações que ocorrem são mínimas, sendo em volume de dados e na complexidade de carga. A principal diferença é a de que os *Data Marts* são voltados somente para uma determinada área, já o DW é voltado para os assuntos da empresa toda.

Existem duas maneiras distintas de criação de *DW*: *top-down e bottom-up* [Kimball, 1996].

Top-down: a organização cria um DW e depois parte para a segmentação, ou seja, divide o DW em áreas menores gerando assim pequenos bancos orientados por assuntos departamentalizados.

Bottom-up: a situação é inversa. A organização por desconhecer a tecnologia, prefere primeiro criar um banco de dados para somente uma área. Com isso os custos são bem inferiores de um projeto de DW completo. A partir da visualização dos primeiros resultados parte para outra área e assim sucessivamente até resultar em um *Data Warehouse* completo.

Em ambas existem desvantagens. O desenvolvimento *Top-down* a partir de um DW já existente resulta em um alto custo de implementação, complexidade na modelagem e acarreta em dificuldades políticas e financeiras ao decorrer do projeto. Já no desenvolvimento *Bottom-up* na ausência de um DW central e coordenado, a flexibilidade certamente não será alcançada, além do mais, os pré-requisitos para desenvolver *Data Marts* independentes levariam estes ao uso de sistemas de informação operacionais individuais, contradizendo com a necessidade de uma visão mais ampla.

Como alternativa para as desvantagens apresentadas acima surgiu o conceito de *Federated Data Warehouse(*FDW) (Figura 2.2), que busca combinar as técnicas de desenvolvimento *bottom-up* e *top-down*, enquanto diminui os riscos de fracasso de um projeto de DW. Nesse modelo os *Data Marts* independentes, são controlados por um bloco único de metadados e coexistem com um EDW (*Enterprise Data Warehouse*) dependendo dos mesmos metadados [Jarke *et al.*, 1999].

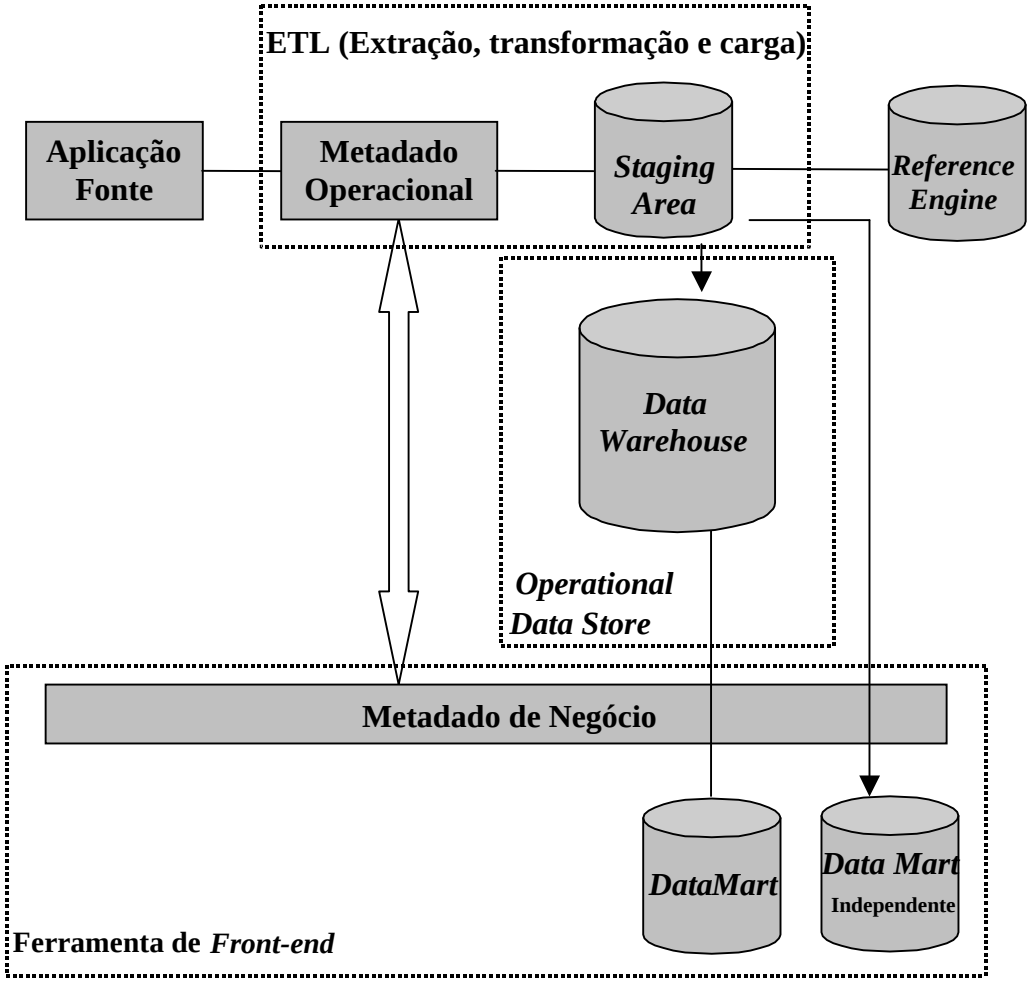

Figura 2.2: *Federated Data Warehouse*

O desenvolvimento de um FDW, devido à sua complexidade, é o mais dependente do controle de metadados para dar certo. Os metadados serão usados para assegurar que toda informação deverá ser enviada para todos os *Data Marts*, sendo eles independentes ou não do EDW.

Em linhas gerais, os metadados englobam as maiores funções de cada ciclo de vida dos dados dentro de um *Federated Data Warehouse*. Os metadados provêm uma linguagem comum para definir o comportamento dos dados. A sintaxe formal dos metadados permite definição robusta de regras de negócio e significados associados com os dados que estão sendo transferidos do sistema de informação operacional de origem para o EDW e eventualmente para dependentes ou independentes *Data Marts*. Isso se aproxima da visão tradicional dos metadados, descrevendo como vendas e clientes se relacionam, assegurando a consistência do grupo de clientes, definições universais, valores usados para moeda, entre outros.

Os metadados também controlam o fluxo dos dados. Os metadados técnicos vão conter não só definições dos dados, como também descrições formais da ocorrência dos fluxos, regras de transformação de dados, dependências entre as transformações, e fatores de tempo que afetam a transferência dos dados [Jarke *et al.*, 1999].

Finalmente os metadados vão comunicar a natureza dos dados dentro do *Data Warehouse* para os usuários e desenvolvedores com a necessidade de entender a base dos negócios. Essa habilidade de acessar a definição e o controle dos dados é vital para que a proliferação dos metadados ao longo das comunidades técnicas e não-técnicas seja efetiva e entendida.

O EDW pode suprir papéis diferentes nesse tipo de arquitetura. Ele provê um seguro e completo mecanismo pelo qual a qualidade dos dados pode ser posta em controle de consistência dentro dos *Data Marts*, mas evidentemente apresenta um nível significante de redundância de dados. Cada item incluso no EDW estará no mínimo em uma das estruturas dos *Data Marts*.

Todas as regras de negócio impostas pelo EDW ou pelos metadados técnicos dentro da *Staging Area* serão herdadas pelos *Data Marts*, em um caso ideal. Os metadados técnicos definem e controlam as regras de negócio que são aplicadas nos dados quando os mesmos entram em um ambiente de *Data Warehouse*. Uma vez dentro do ambiente, a interpretação que é posta sobre o dado pelos usuários é controlada pelos metadados de negócio, que também asseguram que o comportamento do dado em diferentes *Data Marts* seja consistente [Harding, 1998].

Na prática, isto implica que os metadados devem ser gerenciados de acordo com os seguintes aspectos:

- As regras contidas dentro dos metadados técnicos devem ser controladas para assegurar que elas estejam corretas. Regras sem necessidade devem ser omitidas e as regras ditas universais não devem ser parciais;

- As regras dentro dos metadados de negócio devem estar presentes em um formulário de fácil entendimento pelo usuário final daquele dado – o principal propósito desses metadados é assegurar que o usuário possa acessar e manipular o dado eficientemente e corretamente. Isso não vai acontecer menos

que os metadados de negócio sejam robustos e presentes nos termos de negócio;

- O mecanismo de controle deve assegurar que os níveis dos metadados de negócio e técnicos estejam consistentes. Se isso não acontecer, os metadados vão piorar a qualidade dos dados ao invés de melhorar.

2.3 Metadados Operacionais

Os metadados operacionais fazem parte do conjunto de metadados técnicos de um processo de *Data Warehousing.* Estes metadados exercem uma influência no ambiente de entrada de um *Data Warehouse*, determinam a aceitabilidade dos dados que são transmitidos para o *Data Warehouse* e controlam o processo pelo qual a transmissão é feita. É fundamental para a natureza de um ambiente de *Data Warehouse* que o escopo do sistema esteja em constante fluxo. Mudanças em relação a negócios, novas telas, ferramentas de análise oferecendo novas maneiras de análise de dados, e a consistência e completeza oferecidas por um *Data Warehouse* podem significar benefícios que irão contribuir para uma constante evolução.

Os metadados operacionais para suportarem essa constante evolução do *Data Warehouse* devem possuir um conjunto de classes mínimas tais como [Kimball et al., 1998] [Marco, 2000] [Inmon, 1996] [Vetterli, 2001]:

- Aplicação – Define qualquer sistema que atue como fonte de dados;

- Fluxo dos dados – Define a relação de fluxo do começo ao fim, para um segmento de trabalho independente, dentro do carregamento de um *Data Warehouse*;

- Ordem de Transferência – uma subdivisão do Fluxo de Dados, permitindo ao desenvolvedor entender como a transferência vai funcionar, a ordem em que as tarefas serão executadas, e sob quais circunstâncias ela poderá dar certo;

- Transformação – É uma atividade, que no nível de Ordem de Transferência, muda a estrutura ou conteúdo de alguma parte do dado que está sendo transferido;

- Classes Fonte – Define os objetos que existem dentro da aplicação de origem e que são acessados para procurar os dados que serão transferidos para o *Data Warehouse*;

- Aributos Fonte – atributos individuais que são usados como dados de origem nas Classes Fonte;

- Classe Destino – Provê uma definição para cada classe dentro do ambiente de DW;

- Atributo Destino – Provê uma definição em nível de atributo para cada item do dado;

- Armazenamento Físico – Fonte atual, intermediária e os objetos de destino usados na transferência;

- Regras – Definem as regras que estão alocadas em cada item do dado (geralmente a nível de atributo) enquanto passa pelo ambiente de *Data Warehouse*, possivelmente fazendo a validação, medição, dependências complexas com outros atributos, dependências com data e hora, e outros efeitos externos;

- Dependências – Dependências existentes na Ordem de Transferência – adicionando sofisticação para o conceito de fluxo de trabalho, permitindo o desenvolvedor definir a ordem de cada passo. Na verdade uma implementação mais moderna dessa característica seria possível com uma ferramenta de gerenciamento de projetos. Vale também considerar que a ferramenta ETL irá permitir a gravação e o controle desses modelos de trabalho.

Os elementos citados acima são basicamente estáticos. Eles não são afetados pelo banco de dados ou por outras influências externas. Muitas companhias oferecem ferramentas que possam aumentar a capacidade de gerenciar metadados operacionais. Em alguns casos, as ferramentas combinam gerenciamento de metadado técnico com a capacidade de uma ETL, definindo sua *Staging Area* como seu escopo.

Esses tipos de ferramentas confiam muito na captura e no gerenciamento dos metadados. Elas são vendidas como eficientes mecanismos para desenvolvimento e controle

de seqüências ETL, mas são na verdade avanços significativos na área de gerenciamento de metadados automativos. Em alguns casos, modelos de metadados básicos têm sido estendidos para inclusão de características dinâmicas que provêm informações como volume de dados, volatilidade, tempo de carga e marcadores para transferência de dados [Harding, 1998].

Estáticos ou dinâmicos, os metadados que controlam a extração, transformação, e carregamento dos dados devem idealmente ser capturados por uma interface gráfica, permitindo ao desenvolvedor ver o dado sendo carregado a partir de um diagrama, criando tarefas independentes, identificando e escolhendo a partir dos menus os campos de dados de origem. Por outro lado, sem um robusto modelo de metadados, é impossível controlar a carga, não importando o nível de modernidade que o *front-end* de uma ferramenta ETL possa oferecer.

2.4 Metadados de Negócio

Os metadados de negócio englobam informações requeridas pelos usuários finais para balanço e sobre como usar o conteúdo de um *Data Warehouse* para suprir suas necessidades. Essencialmente, metadados de negócio asseguram que cada usuário tenha um ambiente de *Data Warehouse* consistente bem entendido, e que possam usar isto para formular um método apropriado para encontrar suas necessidades de negócios. Em conseqüência disso, os metadados de negócio devem responder perguntas como: O rendimento inclui ou exclui taxas de venda ?; Quando alguém de outro departamento fala sobre tipos de produtos, a que ele ou ela está se referindo ? Qual a fórmula usada para encontrar o resultado final do banco ? [Banese, 2000].

A capacidade de entender e analisar deve ser grande para responder estes tipos de perguntas citadas acima. É função dos metadados de negócio controlar e gerar respostas para estas perguntas, e assegurar que os usuários do DW e *Data Marts* estejam trabalhando de acordo com as regras da empresa.

Para o controle de metadados de negócio também é necessário possuir um conjunto mínimo de classes tais como [Harding, 1998]: Apelidos - Nomes secundários para classes e atributos – assegurando que exista um entendimento comum das regras de negócio; Domínios; Volumes; Cardinalidade; Volatilidade; Perspectivas do negócio; Medidas; Atributos sem medidas; Escopos de análise e etc.

2.5 Suporte à Decisão e OLAP

O processamento OLAP provê acesso aos dados corporativos de um *Data Warehouse* com total segurança e controle, e provê aos usuários todas as flexibilidades existentes em programas dedicados à análise de dados.

A tecnologia OLAP dispõe de um conjunto de operações e ferramentas que torna o usuário capaz de lidar com a complexidade das planilhas. É possível analisar tendências, fazer comparações, destacar problemas e manipular as informações em qualquer ordem.

O termo OLAP (*On Line Analytic Processing*) representa o tipo de processamento e ferramentas, baseados em um conjunto de tecnologias projetadas para suportar análise e exploração de dados que dão suporte à decisão.

A principal característica dos sistemas OLAP é permitir uma visão conceitual *multidimensional* dos dados de uma empresa. Os dados são modelados numa estrutura conhecida por cubo (Figura 2.3), onde cada dimensão representa os temas mais importantes da empresa como produto, cliente, funcionário e tempo [Chaudhuri & Dayal, 1997].

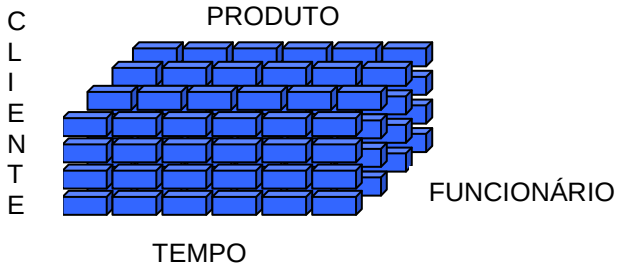

Figura 2.3: Cubo de 4 dimensões

2.5.1 Conjunto de Operações OLAP

Após a montagem do cenário de uma consulta, freqüentemente o analista de negócios deseja mudar o resultado da análise. De nada adiantaria a montagem se não houvesse meios de manipulação. Atualmente as ferramentas OLAP fornecem suporte para funções de derivação de dados complexos que recebe o nome de *Slice and Dice* [Dinter *et al*., 1998].

O suporte *Slice and Dice* é uma das principais características de uma ferramenta OLAP. Ele serve para modificar a ordem das dimensões, alterar linhas por colunas de maneira a facilitar a compreensão dos usuários. Essa característica é de extrema importância, com ela podemos analisar as informações de diferentes prismas, permitindo ao usuário investigar diferentes inter-relacionamentos. O *Slice and Dice* compreende quatro operações: *Ranging*, *Drilling*, *Rotation* e *Ranking*.

- ***Ranging*** **-** É a operação responsável por, a qualquer momento, alterar o resultado das consultas, inserindo novas posições ou removendo as que estão em foco. Para que isto ocorra é preciso que o usuário informe o que está modificando e o que será feito. Por exemplo, a inserção de um novo produto em uma consulta. O resultado desse *Ranging* será considerado para todas as demais operações, ou seja, pode-se encarar o resultado como um novo cubo gerado a partir do cubo original.

- ***Drilling*** **-** Após escolher o que deseja analisar, o analista ainda pode mudar o escopo do que está analisando, porém essas informações podem encontrar-se agregadas em diversos níveis. O *Drilling* permite navegação por entre estes níveis. Existem três operações OLAP que permitem ao analista mudar o escopo dos dados: *Drill Down*, *Drill Up* e *Drill Across*.

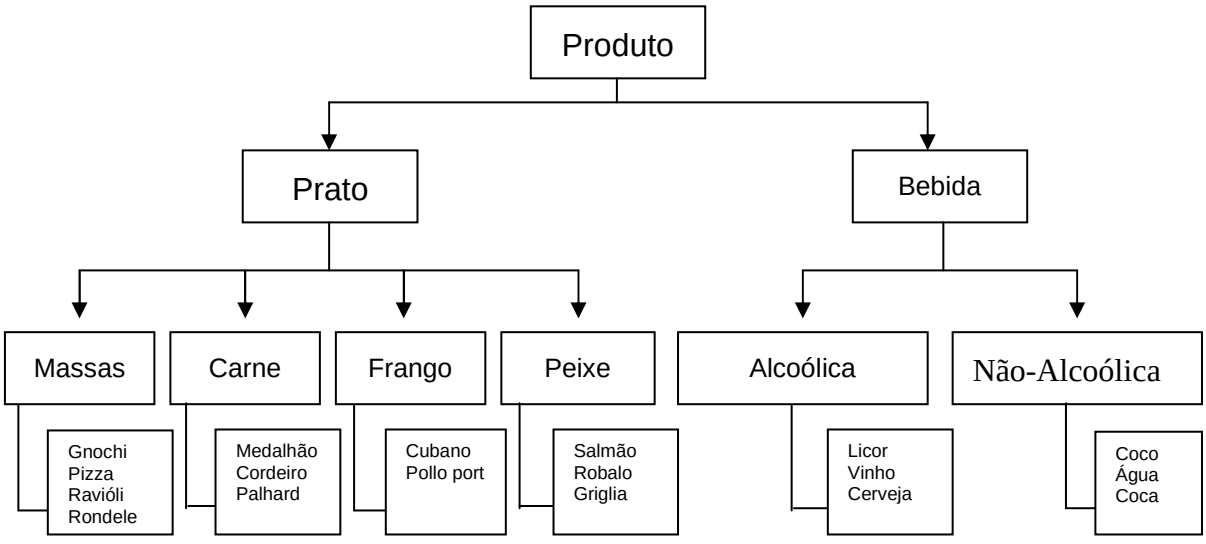

Figura 2.4: Exemplo de hierarquia de uma dimensão Produto para uma organização de restaurantes onde podem ser efetuadas as operações de *Drilling*.

- ***Drill Down*** **-** Esta operação navega verticalmente na hierarquia no sentido em que os dados são mais atômicos. Considerando a Figura 2.4 é possível analisar as vendas de todos os produtos do restaurante.

- ***Drill Across*** **-** Esta operação permite navegar transversalmente no eixo da árvore hierárquica. O *Drill Across* é uma operação de grande utilidade, pois permite inserir e retirar posições do cenário corrente.

- ***Drill Up*** **-** A última operação do *Drilling* realiza a função inversa do *Drill Down*. Ela permite ao usuário uma visão mais agregada das informações. Nesta técnica podemos navegar nos diversos níveis até o mais sumarizado.

- ***Rotation*** **-** Além de mudar as posições em foco, o analista também tem a flexibilidade de alterar a forma de visualização das informações. O *Rotation* permite ao analista mudar a visão que está tendo dos dados. Vale salientar que o *Rotation* não adiciona nem retira posições do cenário. A grande vantagem do *Rotation* é que não há operações de disco no servidor para que os dados sejam vistos de forma diferente. O que muda é apenas como os dados serão dispostos para o usuário.

- ***Ranking*** **-** Com esta operação o analista pode filtrar as informações que ele deseja ver. É possível fazer uma classificação dos dados obtidos e operar diretamente sobre os valores das células. Vale frisar que as operações anteriores atuavam apenas sobre as posições ou dimensões dos dados. Através do *Ranking*, o analista pode executar diversos tipos de filtros, eliminando assim as informações desnecessárias.

2.5.2 OLAP *Multidimensional* (MOLAP)

A primeira categoria de ferramentas chama-se *OLAP Multidimensional* (MOLAP). Essas ferramentas utilizam um modelo *multidimensional* de dados [Jarke *et al*., 1999] [Agrawal, 1999] e permitem a navegação de um nível de detalhamento para outro “mais baixo” ou “mais alto” em tempo real.

Os bancos de dados *multidimensionais* surgiram para melhorar a relação custo/benefício na implantação OLAP. Em resumo o MOLAP é uma classe de sistemas que

permite a execução de análises sofisticadas e exploração de dados usando como gerenciador um banco de dados *multidimensional*.

Em um banco de dados MOLAP os dados são mantidos em arranjos que simulam um cubo com *n* dimensões e indexados de maneira a prover um ótimo desempenho no acesso a qualquer elemento. Combinando as dimensões, o usuário pode ter uma visão geral de todos os dados da empresa num excelente desempenho.

Um *array multidimensional* tem um número fixo de dimensões e os valores são armazenados nas células. Cada dimensão consiste de um número de elementos. Em outras palavras o cubo (Figura 2.5) é, de fato, apenas uma metáfora visual, uma representação intuitiva do evento, pois todas as dimensões coexistem para todo ponto no cubo e são independentes umas das outras.

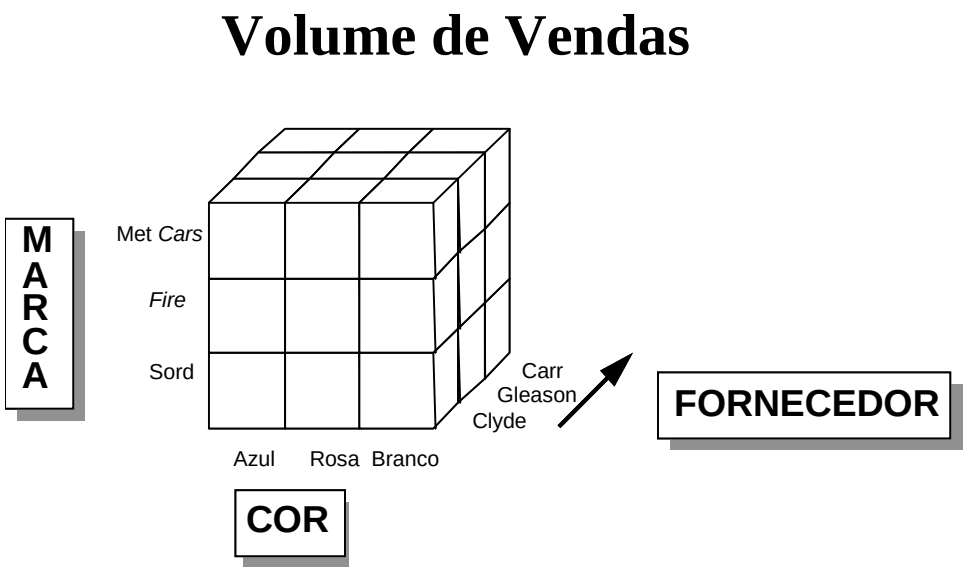

Figura 2.5: Visão *multidimensional* do volume de vendas de uma rede de concessionárias

2.5.2 OLAP Relacional (ROLAP)

Como a maioria das empresas utiliza os Bancos de Dados Relacionais, também surgiu paralelamente ao conceito de MOLAP, o conceito de ROLAP (Relacional OLAP) como uma grande solução para o aproveitamento de uma tecnologia aberta e padronizada que se beneficia de uma grande diversidade de plataformas e paralelismo de hardware.

Os sistemas ROLAP permitem análise multidimensional dos dados que estão armazenados em uma base de dados relacional, podendo ser feito todo o processamento no servidor da base de dados e depois gerados os comandos SQL [3] e as tabelas temporárias. De forma inversa podem ser executados os comandos SQL para recuperação dos dados e posterior processamento no servidor OLAP.

Os servidores ROLAP além de possuírem as características dos sistemas OLAP, utilizam-se de metadados para descreverem o modelo dos dados e auxiliarem a construção de consultas. Também são criados comandos SQL otimizados para bancos de dados, ajudando o analista na utilização desta tecnologia.

Nas ferramentas ROLAP, o uso de uma camada semântica acima do esquema relacional permite que os dados sejam apresentados ao usuário através do modelo multidimensional. Logo, interfaces ou servidores ROLAP são definidos como sistemas OLAP que traduzem operações e consultas realizadas em um esquema multidimensional para um esquema relacional [Dinter *et al*., 1998] [Chaudhuri & Dayal, 1997]. Contudo, algumas ferramentas exigem que os dados estejam estruturados em um esquema relacional particular que facilite a tradução de consultas entre os dois modelos. Este esquema, denominado de esquema estrela, permite a simulação de um Banco de Dados *Multidimensional* numa base relacional [Kimball, 1997] [Kimball *et al*., 1998]. Por considerarmos irrelevante para este trabalho a pormenorização do esquema estrela, não estenderemos a explicação do mesmo.

As ferramentas ROLAP têm vantagem sobre as ferramentas MOLAP pelo fato de que não precisam de um grande armazenamento extra, uma vez que os dados não são copiados fisicamente do *Data Mart*. Isso quer dizer que os metadados dentro do *Data Mart* podem ser acessados e são relevantes ao seu uso eventual. Ferramentas ROLAP, entretanto, tem um menor desempenho em relação às ferramentas MOLAP, sendo um fator crítico em grandes DWs.

2.6 Conclusões

A maioria dos DW possui repositórios de metadados dentro da aplicação da qual o dado é retirado, dentro da ferramenta ETL, e encaixados no *front-end* da ferramenta OLAP.

[3] SQL – *Strutured Query Language* – Linguagem de manipulação e definição de dados de um Banco de Dados Relacional [Elmasri & Navathe, 1994].

Freqüentemente, certas partes da base de metadados são compartilhadas com outras ferramentas, enquanto outras são independentes e isoladas. Teoricamente, existem várias maneiras de assegurar a consistência dos metadados tais como [Marco, 2000] [Banese, 2000]:

- Confiar em um compartilhamento bilateral dos metadados e entre os vários componentes ao longo da cadeia de informações – isso irá prover uma solução parcial, mas provavelmente excluir alguns dos elementos de maior contato com os usuários no controle dos metadados.

- Construir ferramentas automatizadas para checar a consistência de vários repositórios e controlar a arquitetura dos dados. Isso pode se tornar efetivo, mas vai precisar de um profundo conhecimento interno dos repositórios de metadados ao longo da corrente de informação.

- Adotar uma ferramenta universal de controle de metadados que mantém a consistência entre vários repositórios – infelizmente essa ferramenta ainda não existe.

- Definir uma combinação de procedimentos manuais, documentações e facilidades que irão permitir o gerenciamento da consistência. Deve haver direção do processo inteiro pela documentação e interligação de classes de metadados. Pode-se conseguir isto de várias maneiras como:

    - Ferramenta de *Work Flow:* Pode gravar várias entidades de metadados em tipos de documentos separados e utilizar características particulares para manter integridade.

    - *Intranet:* Um bom método de compartilhamento através de um banco de dados acessado através de uma *intranet*, contendo metadados.

    - Ferramenta *CASE* [4]*:* Contém metadados que tendem a ser usados por analistas de sistemas e pessoas com um sólido entendimento em

[4] *CASE - Computer Aided Sotfware Engineering* - Toda ferramenta que ajude no processo de construção de um *software* seja ela lógica ou física, documentação ou teste.

modelagem de dados. Isto pode restringir a proliferação de metadados de negócio.

- Construindo um gerenciador de metadados dedicados: Ferramenta que exige um conhecimento muito vasto de diversas tecnologias e metodologias de desenvolvimento para ser modelada e construída.

O gerenciamento de metadados em um DW sempre será um grande problema de acordo com a tecnologia existente. A preocupação é assegurar que os dados que perfazem todo o ciclo de vida de um DW permaneçam consistentes. Em cada fase, existe a possibilidade de uma nova interpretação das regras de negócio, gerando inconsistências. Estas inconsistências poderão sair nos relatórios gerenciais, causando erros de julgamento, estatísticas e perda de tempo na tentativa de reparar esses conflitos.

Em resumo, neste capítulo, apresentamos o domínio dos Sistemas de Suporte à Decisão baseados em *Data Warehouses* com ênfase na importância do gerenciamento de metadados. Foram introduzidos conceitos necessários ao entendimento dos requisitos para a construção de um gerente de metadados. O próximo capítulo necessitará dos conceitos aqui introduzidos, para a concepção de uma boa arquitetura de gerência e armazenamento de metadados.

Capítulo 3

Elementos de uma Arquitetura de Metadados

Para a definição de uma arquitetura básica de metadados devemos sempre ressaltar a inevitável presença de diversas fontes de metadados em um ambiente de *Data Warehouse.* Em virtude disto, é necessária a presença de uma ferramenta de integração de metadados para unir as várias fontes de metadados (Figura 3.1) [Marco, 2000] [Vaduva, 2000] [Harding, 1998].

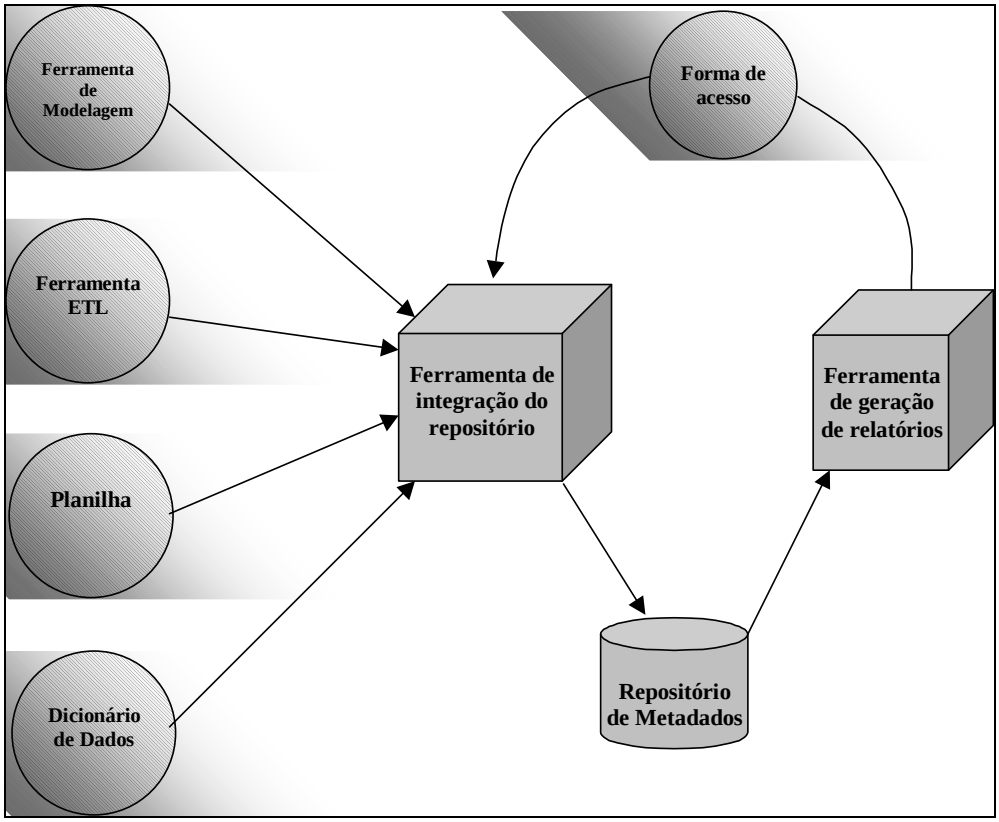

Figura 3.1: Uma arquitetura básica de metadados

Para o uso da ferramenta de integração, precisa-se saber quais os metadados genéricos e se a mesma suportará todos os outros metadados existentes no processo. A identificação das

fontes e de seus requisitos de integração é fundamental para a visualização de possíveis necessidades de mudanças no modelo da base de metadados suportado pela ferramenta de integração de dados.

Deve haver compatibilidade entre o repositório e as ferramentas ETL e de modelagem. Já o processo de integração com um dicionário de dados pode ser feito através da escrita de um programa complexo para moldar o dicionário de dados a um formato que possa ser integrado à ferramenta do repositório ou, idealmente, o modelo de metadados suportado pela ferramenta de integração de metadados deve possuir os campos necessários para suportar esses metadados.

Finalmente, pode ser usada uma ferramenta OLAP para acessar a informação no repositório ou até mesmo ser desenvolvida uma *interface* Web para acesso através de uma Intranet, por exemplo.

3.1 Tipos de Arquitetura de Metadados

Podemos definir 4 tipos de arquitetura de metadados [Marco, 2000]. São elas:

- **Repositório de Metadados Centralizado:** O conceito de arquitetura de metadados centralizada é um modelo uniforme e consistente onde o esquema de definição e organização de vários metadados seja armazenado em um repositório global de metadados.

- **Repositório de Metadados Descentralizado:** O objetivo de uma arquitetura descentralizada é um modelo uniforme e consistente onde o esquema de definição e organização de vários metadados seja armazenado em um repositório global de metadados e em elementos de metadados que apareçam em repositórios locais. Todo metadado que é compartilhado e reutilizado entre os vários repositórios primeiro deve passar pelo repositório central. Porém, o compartilhamento e acesso ao metadado local são independentes do repositório central. O repositório central é o conjunto dos metadados armazenados em repositórios locais. Essa arquitetura é muito desejável para aquelas empresas que têm linhas de negócio diferentes e não-relacionadas, pois há o compartilhamento dos metadados entre múltiplos repositórios e

também cada repositório local é autônomo para com seu conteúdo e requisitos administrativos.

- **Metadado Bidirecional**: Uma arquitetura de metadado bidirecional permite que o metadado seja modificado no repositório para depois ser transportado para seu lugar de origem (Figura 3.2). Por exemplo, se um usuário acessa o repositório e muda o nome de um atributo para um dos *Data Marts*, a mudança é repetida na ferramenta de modelo de dados para atualizar o modelo físico para aquele *Data Mart* específico. Apesar de complexa, esta arquitetura é bem desejada, pois permite que as ferramentas compartilhem os metadados e também permite que as empresas detectem modificações nos metadados, o que é extremamente atrativo para organizações que querem implementar um repositório de metadados em um nível mais elevado.
  Existem 3 (três) desafios para implementação de metadados bidirecionais:
    - O repositório de metadados deve sempre ter a última versão da fonte de metadados que irá ser alimentada;
    - As mudanças precisam ser capturadas e resolvidas, pois um usuário pode estar mudando o metadado no repositório enquanto outro usuário muda o mesmo metadado em sua fonte;
    - Mais interfaces precisam ser modeladas para o envio dos metadados do repositório às suas fontes.

- **Metadado Rotativo**: O metadado rotativo permite que o repositório envie seu metadado de volta ao sistema de informação operacional da empresa (Figura 3.3). Essa arquitetura é parecida com a bidirecional, mas nesse caso o repositório envia seu metadado para o sistema operacional e não para dentro de outras aplicações. Metadados rotativos estão ganhando popularidade entre as organizações que querem implementar um repositório de grande escala, pois permite fazer mudanças globais no repositório e ter essas mudanças detectadas dentro do sistema operacional da empresa.

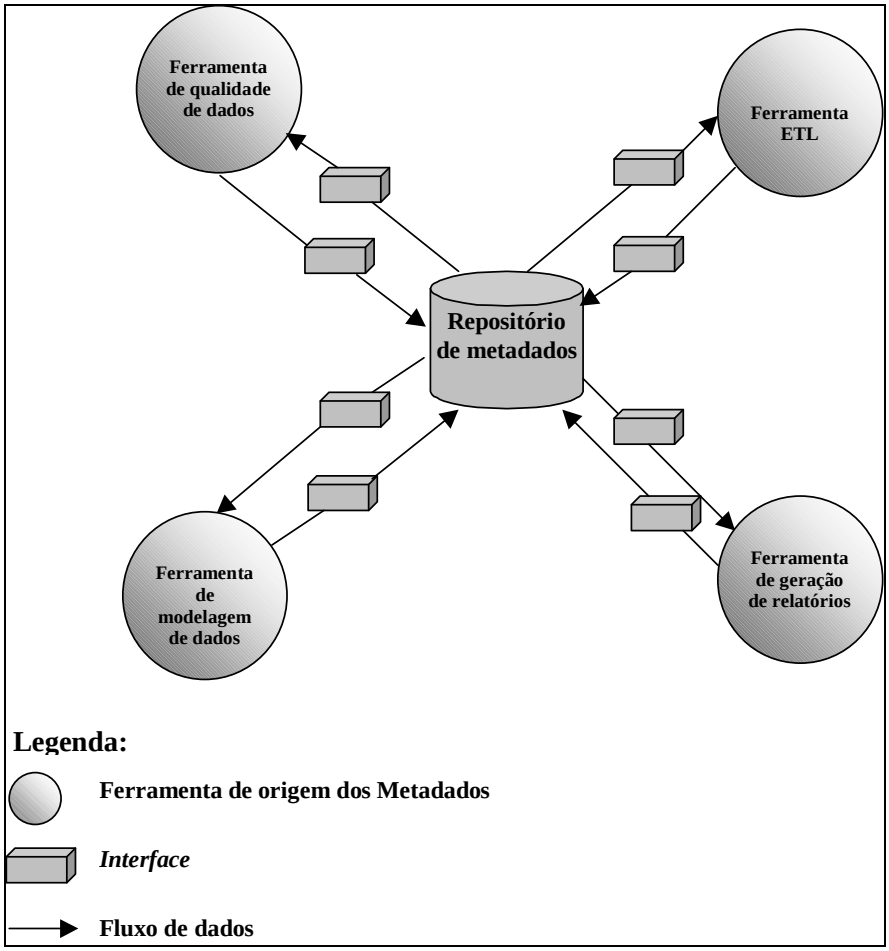

Figura 3.2: Uma arquitetura de metadados bidirecionais

Metadados rotativos possuem as mesmas complexidades dos bidirecionais. O repositório terá que possuir a última versão dos metadados, no caso desse metadado ser enviado do repositório para o sistema operacional. Se o repositório não possuir a última versão, não há garantia de que o usuário atualize a última versão. Além disso, um usuário pode fazer mudanças nos metadados no repositório ao mesmo tempo em que outro está mudando no sistema operacional. Esses conflitos devem ser detectados e *interfaces* construídas para unir os metadados a sistemas operacionais. Apesar de poucas empresas estarem usando uma arquitetura rotativa, é um grande progresso em repositórios de metadados.

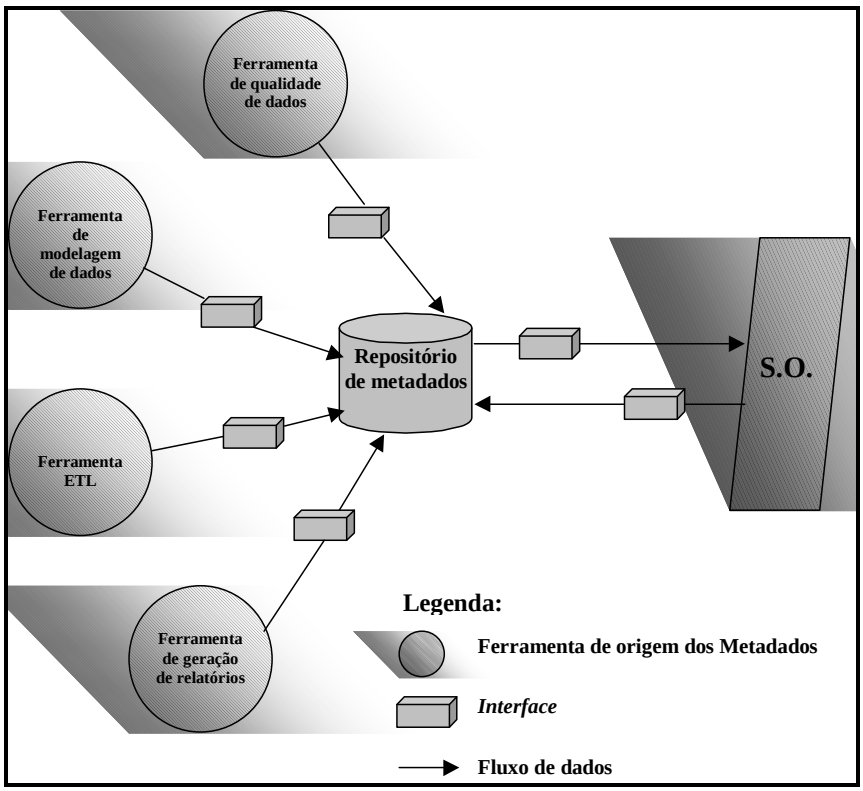

Figura 3.3: Uma arquitetura de metadados rotativos

3.2 Requisitos de uma Arquitetura de Metadados

3.2.1 Integração

O maior desafio em construir um DW é integrar todos os seus dispositivos de dados e transformar esses dados em informações valiosas. O mesmo acontece para um repositório de metadados. Por exemplo, uma organização deseja mostrar a seus usuários a definição de um campo que aparece em um relatório, e digamos que esse campo veio de uma planilha eletrônica. O processo de integração dos metadados deve criar um link entre os metadados do relatório e o campo da planilha. A integração dos dados é a tarefa mais complexa na implementação de um repositório de metadados [Marco, 2000] [Vaduva, 2000].

3.2.2 Extensibilidade

Se a integração é a característica mais difícil a ser atingida, a escalabilidade é a mais importante. Um repositório de metadados que não foi feito para crescer constantemente, breve ficará obsoleto [Marco, 2000] [Vaduva, 2000]. Três fatores estão levando os repositórios a esse crescimento:

- **Crescimento contínuo de sistemas de suporte à decisão:** É normal que o tamanho de um DW e o número de usuários que acessam esse DW dobrem em um só ano. Se esses sistemas de suporte à decisão crescem, o repositório tem que crescer para atender esse crescimento.

- **Reconhecimento do valor de um uso mais amplo dos metadados:** Durante os últimos cinco anos as empresas começaram a notar a importância de ter um repositório de metadados. Essas empresas não estão utilizando o repositório só para suporte à decisão, mas também para outros fins.

- **Maior confiança no gerenciamento do conhecimento:** Gerenciar conhecimento consiste em identificar, capturar, e compartilhar todos os recursos da empresa. Em resumo o conceito de gerenciamento do conhecimento é a captura dos recursos e disponibilização dos mesmos na rede da organização.Nos últimos anos, as organizações começaram a entender que o repositório de metadados é a principal estrutura para implementar o gerenciamento do conhecimento.

A ferramenta que gerencia o repositório também deve ser transportável entre sistemas operacionais, pois a mudança de um sistema operacional na organização não pode devastar um projeto de repositório de metadados.

3.2.3 Robustez

Como qualquer sistema, um repositório de metadados deve ter funcionalidade e desempenho suficiente para atender as necessidades da sua empresa. Deve atender os usuários técnicos como também os usuários de negócios, e mais algumas características como [Marco, 2000] [Vaduva, 2000]:

- Capacidade de importação e exportação de metadados;

- Configuração segura e facilidades de permissão de acesso aos dados;

- Arquivamento e backup simplificados;

- Habilidade de gerar relatórios técnicos e de negócios.

3.2.4 Abertura

A tecnologia usada para a integração de metadados e acessos deve ser aberta e flexível. Por exemplo, os bancos de dados usados para guardar metadados são geralmente relacionais, mas a arquitetura de metadados tem que ser suficientemente flexível a ponto de permitir que a organização mude de um banco de dados relacional para outro tipo de arquitetura. A abertura também diz respeito ao acesso ao repositório, ou seja, as organizações geralmente desejam fornecer seus relatórios aos seus usuários via *Web* direto de qualquer *browser* [Vaduva, 2000]. Um bom gerente de metadados deve permitir isso.

3.2.5 Automatização e Reutilização de Processos

O processo de carregar e manter o repositório de metadados deve ser o mais automatizado possível. Implementações que contêm muitos processos manuais em sua arquitetura diminuem o desempenho do repositório, ou muitas vezes tornam-o inativo. Com uma análise cuidadosa e alguns esforços, esses processos podem ser automatizados. Infelizmente alguns desses processos precisam ser feitos de forma manual para funcionar, e precisam de tempo para ser completados. Nessas situações, é mais fácil desenvolver um *front-end* para os usuários e deixá-los criar e manter seus próprios metadados. Mesmo que esses usuários não se sintam à vontade em criar seus metadados, pelo menos até o repositório estar funcionando, um bom *front-end* e uma boa assistência da equipe de desenvolvedores poderiam encorajá-los nesta tarefa [Marco, 2000].

3.2.6 Padronização do Processo de Integração

O processo de integrar dispositivos de metadados é baseado no mesmo conceito de ETL (Extração, transformação, carregamento) em um DW [Marco, 2000]. O processo ETL divide-se em:

- **Camada de Extração (Aquisição de Dados):** A principal atividade da camada de extração é adquirir dados de várias fontes sem causar impacto algum nessas fontes. Duas transformações ocorrem ao dado nesta camada. O repositório deve decidir se a seleção de gravação deve ser usada nessa camada ou na camada de transformação. Geralmente é bom evitar critério de gravação nesse ponto, ao menos que os dados no metadado sejam bastante volumosos e essa quantidade que será carregada no repositório seja um subconjunto menor.

Logo após, o desenvolvedor pode adicionar campos específicos para serem usados no repositório.

- **Camada de Transformação:** A camada de transformação é a "espinha dorsal" do repositório. A mais importante atividade do repositório de metadados acontece nessa etapa: integração e limpeza da fonte de metadados. Após esta tarefa, os metadados ficam prontos para serem carregados no repositório. A transformação e limpeza devem acontecer na mesma plataforma física do repositório de metadados. Desse modo, enquanto as requisições ao repositório crescem, toda fonte de metadados pode ser integrada na mesma área física, portanto minimizando mudanças na camada de extração e reduzindo os pedidos à fonte de metadados.

  As funções de transformação e limpeza podem ocorrer no mesmo programa ou processo, ou em vários processos, mas os arquivos gerados provenientes dos processos de transformação devem sempre se igualar aos das classes de metadados, de onde ele foi carregado. Qualquer erro que possa ocorrer quando o repositório estiver sendo carregado, acontece nessa camada. Isso é importante para criar procedimentos de recuperação no caso da leitura tiver que parar.

- **Camada de Leitura:** A camada de leitura captura os arquivos que são gerados na camada de transformação e os leva ao repositório. Procedimentos de recuperação também são importantes nessa camada caso haja algum problema no processo de leitura. Geralmente usa-se o mecanismo de leitura de volume padrão e bancos de dados relacionais. Se no futuro houver a necessidade de ligar tabelas relacionais, os processos nas camadas de extração e transformação não serão afetados, mas os processos nesta camada terão que ser modificados. Como poucos processos acorrem nesta camada, as modificações são relativamente fáceis.

Existem 2(duas) grandes vantagens em separar a camada de extração da camada de transformação:

- **Sincronismo** – A camada de extração mantém o metadado no repositório em sincronismo. Digamos que temos 3 classes que precisam de dados da mesma

fonte de metadados. Se existisse um processo para ler cada uma das classes diretamente do mesmo metadado, os dados presentes no metadado poderiam mudar ao longo do processo de montar as classes. Isso pode acontecer se o dispositivo de metadado for dinâmico, e ocorre porque a leitura dos metadados é feita em diferentes instâncias. Assim, a informação no repositório de metadados não estará em sincronização. Criando um arquivo de extração no processo de integração, todas as classes podem ser construídas a partir desse arquivo, que eliminaria qualquer problema de sincronismo;

- ***Backup*** – A criação de um arquivo de extração provê uma cópia da fonte de metadados. Entretanto, se alguma coisa fizer parar o processo de integração de metadados, pode ser feita uma recuperação das mudanças sem causar nenhum dano.

3.2.7 Flexibilidade

Modelos de metadados geralmente usam um esquema orientado a objetos para representação conceitual da organização dos repositórios. O modelo de metadados provê uma estrutura para estabelecer um protocolo para acesso aos metadados, administração e intercâmbio de informações entre os softwares que acessam o repositório. O modelo de metadados deve ser capaz de capturar o que for de interesse das aplicações que fazem uso do repositório. Por exemplo, em um DW ou ambiente de suporte à decisão temos como metadados chave: objetos relacionais e não-relacionais, conceitos multidimensionais, lógicas de transformação, regras de negócio e programas de trabalho. Todos esses metadados devem ser manipulados pelo repositório. A metodologia de modelagem de metadados deve ser capaz de acomodar vários tipos de relacionamentos, como um-pra-um, um-pra-muitos, muitos-pra-muitos, agregação e herança [Marco, 2000] [Vetterli, 2001].

3.2.8 Gerenciamento de Múltiplas Versões de Metadados

No momento que o metadado provê um contexto para interpretação do significado da informação, o repositório tem a função de gerenciar a estrutura dos dados de um DW por um longo espaço de tempo. Por esta razão, é preciso saber o período de tempo em que o metadado é armazenado no repositório. Para realizar isso, as classes do modelo de metadados

devem ter atributos com a data inicial e a data final. Estas datas permitem que o usuário facilmente consulte versões antigas do metadado [Marco, 2000] [Banese, 2000].

3.2.10 Facilidades de Atualização

Inevitavelmente, podem ocorrer mudanças nas aplicações que acessam o repositório a partir do momento que ele for carregado. A equipe que implementou o repositório tem que decidir em que momento atualizar o repositório para cada fonte de metadados. Como regra geral, a maioria dos repositórios é atualizada mensalmente. Na maioria das fontes de metadados, não é necessária uma atualização do repositório para cada mudança que ocorra. Por exemplo, se for adicionado um índice em uma tabela de um sistema fonte, não é necessária a atualização, mas se algo de mais abrangente acontecer como a eliminação de uma entidade de um sistema fonte, o repositório tem que ser atualizado para incorporar os novos modelos de dados, definições de negócios, e assim por diante. Claro que, alguns tipos de metadados, como por exemplo as estatísticas de leitura de um DW e acesso dos usuários, são constantemente atualizados.

3.2.11 Arquitetura Multicamadas

A maioria dos repositórios de metadados é baseada em arquitetura de duas camadas, cliente-servidor, da mesma forma que um servidor de banco de dados funciona, sendo acessado por várias aplicações. Mas uma arquitetura multicamadas provê uma arquitetura mais aberta e extensível para gravação, leitura e modificação dos metadados. Essa arquitetura inclui um servidor de repositório que encapsula um sistema de gerenciamento físico de banco de dados (seja relacional ou orientado a objetos) e provê vários componentes para manipular inúmeras *interfaces* (XML, COM, CORBA ou OLE-DB) [OMG, 2001]. Também deve haver provisão de mecanismos para acesso e gerenciamento de metadados em redes locais e de longa distância para acomodar ambientes de distribuição. Com o rápido surgimento da *internet* e comércio eletrônico, uma arquitetura multicamadas é um importante requisito para um repositório de metadados.

3.2.12 Gerenciamento de segurança

É importante salientar que o metadado é um recurso inestimável em uma organização. O acesso ao metadado deve ser cuidadosamente controlado para projetar recursos intelectuais e assegurar a validação e integridade do metadado para todos os usuários ao mesmo tempo.

O esquema de gerenciamento de segurança para o repositório de metadados é parecido com a maioria dos SGBD´s, mas deve ser projetado para necessidades específicas dos criadores de metadados, usuários e administradores. Além disso, o acesso deve ser restrito ao metadado de acordo com o tipo, ou as operações que os usuários ou administradores dos metadados pretendem fazer. Um esquema de gerenciamento de segurança robusto é um requisito essencial para o repositório de metadados.

3.3 Funcionalidade de um Repositório de Metadados

Para alcançar seus objetivos, o repositório de metadados deve prover certa funcionalidade como: provisão de informação baseada em um metamodelo, acesso automático, administração de versão e configuração, análise de impacto e notificação [Vaduva, 2000] [Banese, 2000] [Vetterli, 2001].

3.3.1 Provisão de Informação

O repositório tem que oferecer mecanismos para exame, filtro e navegação de modo a satisfazer as necessidades de informação dos usuários. O esquema do repositório deve prover consultas sob certas condições como, por exemplo, fazer consultas selecionando apenas o processo de atualização, ou todos os metadados lógicos que se relacionam com objeto de negócio “sócio”.

A estrutura do repositório também deve permitir seleção de metadados de acordo com sua origem, propósito e tempo de produção. Isto exige que os metadados possuam chaves que contêm os valores apropriados (por exemplo, em terminologia relacional atributos para propósito, origem e etc.. devem existir).

Um repositório não só deve prover informações explicitas sobre relacionamentos entre metadados como também implícitas. Em conseqüência disso, uma exigência de funcionalidade essencial é a navegação dentro da coleção de metadados. Começando com um certo elemento, o usuário pode navegar por outros elementos ao longo de relações existentes. Esta navegação é dirigida pelo esquema especificado em um nível conceitual do repositório.

Com relação à filtragem o ideal é que além do exame de atributos fixos seja possível também a procura de palavras-chave dentro de descrições textuais. Desta forma, toda informação relacionada com um tópico pode ser provida.

Com relação à navegação é extremamente necessária uma interface amigável para o usuário interagir com o repositório. Cada usuário deve ter seu ponto de partida de navegação baseado nos seus direitos de visibilidade (visões do usuário).

3.3.2 Metamodelo

Para prover a funcionalidade necessária, um metamodelo apropriado deve ser escolhido. Um metamodelo é o esquema conceitual do repositório de metadados. Especifica elementos de metadados e relacionamentos entre eles.

O metamodelo requer elementos para representar metadados em níveis diferentes de abstração e deve ser extensível para que usuários possam definir aplicações com elementos de metadados específicos.

3.3.3 Acesso ao Repositório

Para ser utilizável o repositório tem que ser atualizado. Só podem ser garantidos o uso próspero e longevidade do repositório se interfaces para interoperabilidade com outros repositórios estejam disponíveis. Para isso é necessário que um padrão de intercâmbio seja adotado e uma API seja fornecida para acesso por outras ferramentas.

3.3.4 Administração de Versão e Configuração

Mudanças importantes de metadados como atualização de esquemas de fontes operacionais requerem criação de versões diferentes e conseqüente armazenamento no repositório. Porém, problemas podem surgir com vínculos incompatíveis, ou descrições no nível conceitual que não são mais válidas. Desta forma o repositório deve prover mecanismos de notificação para descobrir tais erros de estrutura e validade dos metadados.

3.3.5 Análise de Impacto

Esta característica deve permitir que administradores possam avaliar o impacto de mudanças no DW antes que eles sejam feitos. Simulações são feitas para descobrir quais partes do sistema são afetadas quando a aplicação requer mudanças. Por exemplo, mudanças de esquemas fontes podem ter conseqüências para regras de transformação.

3.3.6 Notificação

O repositório deve prover mecanismos de notificação para usuários e/ou ferramentas interessadas em mudanças significantes.

3.4 Metadados Técnicos e Qualidade de Dados em Metadados

Poucos DWs exploram as vantagens de incorporar metadados técnicos especializados em seu modelo de dados de suporte à decisão e processos ETL. Isso sempre leva a uma redução na flexibilidade do DW, causando perda de tempo, dinheiro com manutenção, aquisição de dados e auditoria de informação. Pode também levar os usuários a interpretarem de maneira errada as informações do DW. Nesse tipo de situação, é necessário rever o repositório de metadados e procurar aprimorar a identificação e qualidade dos dados. É importante entender como o uso do metadado controla a qualidade da informação guardada em um DW [Kimball *et al*., 1998] [Marco, 2000].

Sabemos que muitas empresas usam um repositório para acessar informações técnicas e de negócio. Por esse aspecto, o repositório serve como um catálogo de informações para um ambiente de suporte a decisão, sendo como guia para a informação armazenada no DW. Apesar dessa importante característica, o repositório ainda pode ser expandido para uma participação ativa no processamento dos dados.

O repositório de metadados mantém informações do modelo de suporte à decisão, sistemas operacionais fonte, processos ETL, e estatísticas de leitura que abrangem o DW. A integração desses componentes no repositório está em um nível razoavelmente alto. Por exemplo, a informação do repositório pode ser usada para determinar que um sistema operacional de gerenciamento é a fonte que alimenta uma tabela específica em DW relacional. Revendo as estatísticas de leitura do repositório, pode-se determinar como e com que freqüência o DW é atualizado. Essa informação permite aos usuários terem uma boa visão do ambiente de suporte a decisão, mas esse nível de informação é insuficiente quando um usuário técnico ou de negócio precisa ter uma visão mais detalhada dos dados de um DW [Harding, 1998] [Banese, 2000].

Para alcançar uma visão mais detalhada da informação contida no DW, o arquiteto do repositório, o modelador dos dados e os desenvolvedores usam o metadado técnico que foi estendido para forjar um relacionamento mais próximo entre o repositório e banco de dados

de suporte a decisão. Isso é realizado incorporando o metadado técnico no DW. Essa técnica é usada para estender o *design* e a arquitetura de um DW, para aumentar o processo de otimização para aquisição dos dados, atividades de manutenção e medidas para qualidade dos dados.

Para fazer do metadado técnico uma ponte entre o repositório e o modelo de dados de um DW, o projetista deve incluir operadores no modelo de dados físico. O metadado técnico, ao contrário da informação guardada no repositório, é referenciado de maneira livre no DW. Essas marcações de metadados técnicos provêem um bom detalhamento da informação contida no DW. Essa associação direta do metadado para cada informação no DW, é o que distingue o metadado estendido.

Para escolher os operadores, o sistema operacional une os dados ao metadado técnico estendido durante o processo ETL. O metadado técnico se une aos dados no DW provendo um significado semântico mais claro para o dado. Por exemplo, a tabela de um cliente deriva sua informação de duas fontes operacionais. A informação é retirada tanto de uma aplicação de automação de venda quanto de uma aplicação de planejamento de recursos, dependendo da disponibilidade. Sem o metadado técnico estendido na tabela, a informação só pode ser usada do jeito que está, sem o sistema operacional que a provê. O metadado técnico marcado permite determinar quais dados foram derivados das duas fontes.

O projetista do repositório, o modelador dos dados e desenvolvedores precisam calmamente considerar o plano usado pelos metadados técnicos no modelo de suporte a decisão. Usuários de negócio e técnicos do DW devem identificar e concordar com um claro e consistente método de marcação dos dados originários do sistema operacional. Qualquer metadado técnico amarrado aos dados deve ser aplicável por completo, não só a maioria dos atributos da classe.

Prefere-se manter metadados técnicos amarrados a mínimos modelos de dados dimensionais, como aqueles que são só uma ou duas classes, onde uma simples fonte alimenta o DW, ou quando vários sistemas operacionais precisam ser integrados para carregar uma classe de suporte à decisão. Esquemas complicados fazem da saída dos metadados do sistema operacional mais desafiadora.

A equipe do DW é responsável por avaliar o *projeto* e o impacto de manutenção causados pelo uso de metadados amarrados ao repositório, processos ETL, modelo de dados,

tamanho do banco de dados, e o *front-end* para acesso aos dados. Por exemplo, o mesmo *front-end* da ferramenta OLAP requer uma aderência rígida no modelo de suporte à decisão para que funcione completamente a apropriadamente. Isso pode impedir o uso de alguns ou todas as marcações de metadados técnicos em certas dimensões como a dimensão tempo de um DW. Certos tipos de metadados estendidos podem precisar que processos adicionais de ETL ocorram, o que poderia causar lentidão na carga [Banese, 2000] [Marco, 2000].

Uma variedade de marcações do metadado técnico pode ser incorporada ao projeto do modelo de dados de suporte a decisão para melhorar o detalhamento de informações de um DW. Dependendo dos requisitos da aplicação, do número de sistemas fontes operacionais alimentando o DW, ou da complexidade do modelo de suporte à decisão, a inclusão de marcações de metadados técnicos pode ou não fazer sentido. Por exemplo, adicionando uma coluna a uma tabela de dimensão em um DW relacional para identificar o sistema fonte operacional, onde só uma pequena fonte de informação existe para preencher a tabela, pode parecer pouco produtivo ou prover pouco valor. Mas tem que se considerar os possíveis efeitos de não incorporar essa marcação de metadado. Primeiro, o fato de o DW ter uma só fonte hoje, não assegura que amanhã ele vai continuar assim. Segundo, se o DW realmente evoluir e o número de sistemas fontes aumentarem, é difícil mudar a tabela de um DW depois de ela ter sido carregada. E por último, tendo o sistema fonte operacional ligado a todas as tabelas, indiferentemente do número de fontes, estamos provendo consistência para modelos de suporte à decisão e gerando disciplina na implementação da metodologia de desenvolvimento adotada [Banese, 2000].

A incorporação de metadados técnicos em um modelo de DW relacional, por exemplo, acontece durante a transformação do modelo de dados de negócio. Durante essa modelagem, as colunas físicas são adicionadas às tabelas apropriadas como foi determinado pelos requisitos de negócios e avaliação técnica previamente completa. O metadado técnico é incorporado no modelo físico baseado no tipo de tabela que está sendo endereçado.

Dependendo dos requisitos de negócio de um projeto, os seguintes atributos dos metadados técnicos serão muito úteis:

- **Data de Leitura**: Uma das diferenças fundamentais entre um modelo de dados de um sistema operacional normal e modelo de DW é a adição da data em todos os dados. Um dos atributos mais usados dos metadados técnicos é o

atributo “Data de Leitura”. Este atributo mostra quando(data ou hora) uma informação foi carregada pelo DW. Essa data instantânea é usada para manter a integridade temporal do dado em um DW no momento que as informações são inseridas e atualizadas. Esse atributo pode ser referenciado pelo administrador do DW para identificar possíveis dados que precisem ser arquivados ou deletados. Usuários finais só podem usar esse atrinbuto para reconciliar e fazer auditorias no DW. Existem casos em que a data em que o dado foi carregado para o DW tem pouca relevância para o cenário de negócios. A data efetiva dos dados extraídos do sistema operacional pode ser mais importante. É necessário observar estes detalhes quando for feita a determinação de que marcações de metadados técnicos adicionar no modelo de dados.

- **Data de Atualização:** Outro atributo comum dos metadados técnicos é a “Data de atualização”. Esse atributo indica quando um registro foi atualizado pela última vez durante seu ciclo. Esse atributo também é usado para manter a integridade temporal das informações em um DW [Kimball, 1996].

- **Identificador de Ciclo de Leitura:** Esse atributo é um identificador seqüencial nomeado durante cada ciclo de leitura em um DW, não importa a freqüência. Como um identificador seqüencial, pode ser usado facilmente para remover dados de um determinado ciclo caso haja ruptura dos dados ou qualquer outro erro que possa aparecer. O ”Identificador de Ciclo de Leitura” é bem usado em conjunto com uma classe do repositório de metadados que descreva outras estatísticas operacionais sobre o ciclo de leitura. Usando somente o repositório de metadados, pode-se determinar quando e quantos ciclos ocorrerem no DW. Usando o identificador, pode-se determinar até quando e quais registros e foram lidos.

- **Indicador de Versão:** O atributo “Indicador de Versão” identifica a última versão de um registro numa tabela de um DW relacional por exemplo [Kimball, 1996]. Além de ser importante para manter histórico, esse atributo também é muito útil para esquemas que não sejam tipo estrela [Kimball, 1996], onde o modelo de dados é como um DW atômico e as estruturas tendem para a terceira forma normal. Em vez de pesquisar na tabela

pelo campo de data mais antiga, o processo ETL nomeia “N” para o campo mais antigo e “S” para o mais atual. Isso permite que os usuários consultem dados históricos e atuais do DW [Banese, 2000].

- **Identificador do Sistema Fonte Operacional:** Uma das mais úteis técnicas em termos de campos de metadados tanto para administradores, quanto para usuários finais, é o uso do “Identificador do Sistema Fonte Operacional”. Esse atributo é usado para identificar a fonte original dos dados e permite identificar individualmente, para cada linha da tabela de um DW relacional, por exemplo, quais fontes foram usadas na sua construção. Os usuários de negócio poderão questionar a qualidade ou a validade dos dados em um DW para devolver a informação ao sistema de informação operacional de origem que forneceu a informação [Banese, 2000]. Em alguns casos, os administradores podem usar esse atributo para identificar e remover dados corrompidos de um ou mais sistemas fontes operacionais.

- **Identificador de Chaves:** Esse identificador indica se as chaves em um DW ainda continuam ativas em seus sistemas operacionais de origem. O “Identificador de Chaves” provê uma análise alternativa para pesquisas no DW. Esse atributo pode ser usado efetivamente em uma variedade de atividades de análise para identificar quais dados deveriam estar em relatórios. Por exemplo, o “Identificador de Chaves” pode ser usado para identificar e filtrar as contas contábeis que estão inativas em um certo banco [Banese, 2000].

- **Indicador de Nível de Secreto:** Uma das técnicas mais controversas é o atributo “Indicador de Nível Secreto”. Esse atributo é usado para indicar que regras de negócio ou suposições foram aplicadas em um dado em particular durante o processo ETL. Esse atributo possibilita ao usuário medir o nível de credibilidade de um dado baseado em um processo de transformação que foi executado. O “Indicador de Nível Secreto” é freqüentemente usado para identificar problemas de qualidade de dados do sistema fonte operacional e facilitar a correção desses problemas. O “Indicador de Nível Secreto” também é usado para identificar dados que tiveram que ser estimados, previstos ou tendenciados. Desta forma, um DW pode ter, por exemplo, níveis de

estabilidade: um determinado nível representa dados considerados mais voláteis, fáceis de limpar, ou relativamente moderados para definir; outro nível pode representar dados que foram considerados mais problemáticos para definir; um terceiro nível pode consistir de dados que não vieram de nenhum dos sistemas fontes operacionais mas sim de alguma planilha e um quarto nível para marcar os dados de fontes externas como novos serviços ou fontes comerciais.

Vale salientar que incorporar marcação de metadados técnicos faz possível uma variedade de otimizações na aquisição dos dados, atividades de manutenção e informações de auditoria dos usuários finais no DW. Como exemplo podemos citar a possibilidade de eliminação de uma carga corrompida ou suspeita de um DW com maior facilidade, pois antes dos atributos de metadados técnicos serem incorporados dentro do modelo de dados, os desenvolvedores tinham opções limitadas de isolamento e remoção de informações corrompidas. A marcação de metadados técnicos permite aos desenvolvedores serem seletivos em seus métodos de remoção de dados do banco [Banese, 2000].

Em resumo, a incorporação de atributos de metadados técnicos em um DW permite usuários de negócios e administradores resolverem problemas administrativos e de qualidade de dados. Os principais objetivos são: Medidas de qualidade de dados; Amostragem de resultados do ciclo de leitura do processo ETL; Administração e manutenção.

3.5 Exportação e Importação de Metadados

Como o servidor Edwards [Jorge, 2001] também pode ser alimentado através de XML no que diz respeito a metadados, a funcionalidade de exportação de metadados através de XML foi colocada como requisito e implementada no Metha Manager. Nos capítulos 5 e 6 do presente trabalho podemos verificar a análise e implementação desta referida funcionalidade. Além disso, há uma notória convergência do mercado para o uso de XML para troca de metadados. Por estes motivos, apresentaremos a seguir uma introdução à linguagem XML, bem como seu importante papel na troca de metadados.

3.5.1 XML

Em poucos anos a *Word Wide Web* evoluiu de uma novidade para a biblioteca mundial. Na essência desse recurso universal de informações está a linguagem HTML.

Contudo, a HTML ganhou uma linguagem concorrente, a *Extensible Markup Language* – XML. A XML está a caminho de transformar a Web no núcleo comercial e financeiro em todo o mundo.

A Internet está amadurecendo a ponto de ter uma estrutura próxima ao controle de Metadados. Com a definição do padrão emergente para a XML, este controle torna-se ainda mais real.

Como a HTML, XML está coincidente com a definição e estrutura das páginas Web, no entanto, a chave diferencial entre as duas sintaxes situa-se nos focos que XML tem no significado ou contexto de elementos dentro de suas páginas, enquanto a HTML preocupa-se com o *layout* e aparência das páginas Web.

Como um ponto de referência, a XML está submetida a um padrão de *Markup Language* (SGML – ISO 8879). A SGML (*Standard Generalized Markup Language*) é designada como padrão universal de *Markup Language*. Confirmando a definição de documentos padrão, XML tem sido definida como subconjunto simplificado da SGML, evitando a complexidade da generalidade total e apresentando um ambiente flexível e enriquecido.

Por outro lado, SGML é uma linguagem genérica para definição de estrutura de documentos. Uma determinada aplicação SGML, de simples definição com estrutura de documento hierárquico e *link* de *hipertexto*, é definida como HTML. XML é definida e limitada sob a SGML, ativando aplicações práticas de protocolo genérico sem exagerada complexidade.

A XML é definida com uma estrutura de página que utiliza "*tags*" para separar os elementos. Algumas destas *tags* tomam formas de metadados e controlam o subconjunto de elementos (dados localizados no centro das *tags*) de página relativo a outras páginas, endereços de *e-mail*, e etc. O número de possíveis tipos de *tags* dentro do protocolo XML é, teoricamente, infinito.

Em resumo, a expansão da XML no contexto de página básica *markup* se dá devido a dois fatores importantes:

- **Enfoque em conteúdo de página em lugar de interface gráfica:** Considerando que as *tags* de HTML se preocupam com distinção entre o cabeçalho e o corpo de texto, XML pode ser usada para definir partes da página como nome de companhia, fórmula química, marcas registradas, pessoas, etc. É este diferencial que coloca a XML como uma verdadeira ferramenta de metadados, desde que provenha uma base estruturada para o conteúdo da informação em lugar de sua aparência somente.

- **Definições gerais dos dados:** Claramente, se a XML será usada como ferramenta universal de metadados precisa conter definições gerais dos dados. Sabemos que na internet há uma enorme massa de dados heterogênea. Para abranger isto, a XML leva um nível abstrato e define formas de mecanismos pelos quais serão definidos metadados específicos. Se uma aplicação precisa identificar tipos de frutas, marcas de carros, dentro do contexto Web, esta será feita por um *framework* que deve ser adotado para tal especificação. A XML está se fixando entre padrões de dados, sendo utilizada por grandes empresas privadas e governamentais, por padrões globais ISO e etc.

3.5.1.1 Histórico

A XML se origina do padrão SGML (*Standard Generalized Markup Language*) – ratificada pela ISO (*International Standard Organizational*) em 1986, em que se baseia a HTML (*Extensible Markup Language*), criada em 1990. Embora a SGML seja um padrão amplamente usado em documentos e a HTML consideravelmente encontrada como base de milhares de páginas da Web, a XML está obtendo uma rápida aceitação. Desde a sua publicação, em fevereiro de 1998, a XML é vista como o padrão de linguagem de troca de dados.

A SGML é muito poderosa, porém complexa. A XML é um subconjunto da SGML, que tinha como objetivo torná-la leve o suficiente para uso na internet. Todos os documentos XML são documentos SGML válidos, contudo nem todos os documentos SGML são documentos XML válidos (Figura 3.4).

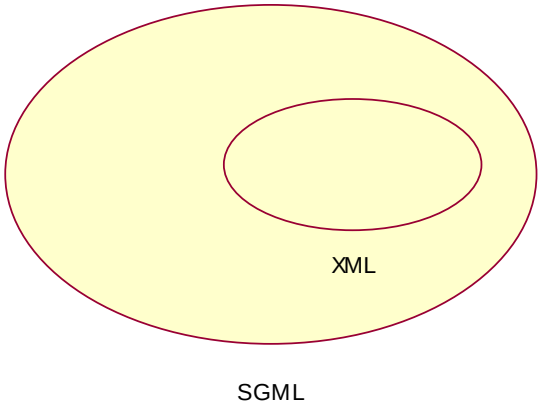

Figura 3.4: Relação entre SGML e XML

A complexidade de implementação da SGML limitou seu uso a grandes empresas que realmente necessitavam de todos os seus recursos. Organizações com dezenas de milhares de páginas de informações são típicas de usuárias da SGML. Com o surgimento da XML, uma SGML simplificada que retém a maior parte dos recursos, podemos pensar em organizações que necessitam de abrangência menor em suas informações.

A linguagem XML foi proposta pelo W3C (*Word Wide Web Consortium*) como um padrão para representação e troca de dados na Web. Uma das aplicações beneficiadas com a utilização de XML é a integração de dados, pois XML fornece uma plataforma para compartilhamento de dados através de uma sintaxe comum [Bray *et al.*, 2001].

3.5.1.2 Conceitos Básicos e Terminologia da XML

A HTML e XML são linguagens de marcação *(Markup Language*), cujos dados são delimitados pelas *tag*s. Na HTML as *tag*s são pré-definidas e necessariamente conhecidas. Ao contrário da HTML, que foi projetada exclusivamente para descrever a apresentação dos dados, XML foi projetada para descrever o conteúdo dos dados. Abaixo seguem algumas características [Chang *et al.*, 2001]:

- XML é uma linguagem extensível;
- A estrutura dos documentos pode ser aninhada em qualquer nível;
- Um documento XML pode conter uma descrição opcional de sua gramática, a qual pode ser usada pelas aplicações para validações na estrutura dos documentos;

- XML é uma linguagem flexível, pois permite a apresentação de um mesmo conteúdo em diferentes formatos.

A XML permite que os usuários definam marcadores de acordo com o domínio que está sendo modelado. Um arquivo XML tem uma simetria e cada dado tem um contexto que o delimita na forma <contexto>...</contexto>. Os colchetes e o texto contido dentro destes são chamados de *tags* e cada conjunto de *tags* e dados que elas delimitam são chamados de elemento. É possível pensar nesse relacionamento como tabelas um banco de dados relacional, sendo as *tags* as colunas e o texto delimitado por elas, as linhas dessas tabelas.

As *tags* podem conter outras *tags* ao invés de dados. Este é um recurso importante da XML, que permite o aninhamento dos dados para definir melhor os relacionamentos. Ainda em relação ao banco de dados, *tags* como <nome> e <tipo> são chamadas, por exemplo, de *tags* filhas da *tag* pai <atributo>. O documento é finalizado com as *tags* finais, essas *tags* são definidas como raiz e somente uma deve existir.

Os documentos XML têm estruturas lógicas e físicas. A estrutura física do documento XML refere-se simplesmente ao arquivo XML e aos outros arquivos para os quais ela pode ser importada, enquanto a estrutura lógica se refere ao corpo do documento.

```
<HTML>
<BODY>
<TABLE>
   <TR>
        <TD>Metha Manager</TD>
        <TD>Universidade Federal da Paraíba</TD>
        <TD>2002<BR>UFPB<BR>Campina
Grande<BR>Primeira Edição</TD>
    </TR>
  </TABLE>
  <BODY>
  <HTML>
```

```
   <?xml version  "1.0"?>
   <Livro>
   <NOME>Metha Manager</NOME>
   <Author>
    <Name>Methanias</Name>
   </Author>
   <Publicação>UFPB</Publicação>
<Cidade>Campina Grande</Cidade>
   <Edição>Primeira</Edição>
</Livro>
```

Figura 3.5: Relação entre HTML e XML

3.5.2 XML para Troca de Metadados

Um desejo atual é que as ferramentas usadas por um DW sejam capazes de integrar todas as fontes de metadados dentro de um repositório. Infelizmente, essa utopia não existe “ainda” no escopo global. Muitas companhias compram uma ferramenta para desenvolver seu projeto de apoio à decisão, mas para um bom projeto há a necessidade de um conjunto de ferramentas integradas.

Uma aliança dos fabricantes de ferramentas é necessariamente importante, mas é difícil prover este fato, pois é difícil construir metadados que compartilhem resultados entre as ferramentas, ou seja, o significado de uma classe de metadado para uma ferramenta pode não ser o mesmo para outra [Marco, 2000] [Banese, 2000]. Se houvesse um padrão de meta modelo estabelecido e amplamente adotado, o número de interfaces para compartilhar os dados seria reduzido. Um modelo padrão de metadados também permitiria a existência de ferramentas para apoiar metadados bidirecionais, uma tarefa muito desafiadora. No capítulo 5 discutiremos os problemas referentes à busca de um padrão de meta modelo pelas empresas que fornecem ferramentas de DW.

Metadados compartilhados entre vários repositórios e ferramentas de *software* são particularmente desejáveis para empreendimentos globais com equipes dispersadas, que usam computação integrada para resolver problemas de análise de dados.

Há uma convergência para que ferramentas de DW usem XML como *interface* para troca de metadados por diversos motivos tais como: [OMG, 2001]

- Codificada corretamente, a XML é fácil de ler e compreender. As *tags* associadas aos dados fornecem um significado de forma a facilitar seu entendimento. Um ótimo resultado deste recurso é que as pesquisas na Internet podem ser mais eficientes se as páginas da Web estiverem escritas em XML;

- A XML se baseia nos padrões abertos definidos pelo W3C. Portanto, nenhuma empresa pode impor seu domínio sobre as outras. Se outras empresas quiserem outras funcionalidades, estas devem ter o conhecimento e aprovação do W3C. Caso essas empresas queiram agir independentemente, correm o risco de não verem seu produto adotado pelo consumidor;

- Facilidades de padronização.

As van*tag*ens da XML são inúmeras, abaixo verificamos algumas:

- XML já é estabelecido como um padrão aberto, plataforma-independente através de organização internacional;

- XML apóia o caráter internacional, fixou padrão ISO;

- O custo para XML é baixo. Podem ser criados documentos de XML à mão até mesmo usando qualquer editor de texto. Principais fabricantes de computadores, inclusive a IBM, oferecem atualmente pacotes de soluções XML escritos em linguagem Java [Flanagan, 1997] e uma variedade de ferramentas de apoio à decisão, inclusive API´s XML, estão disponíveis na Internet;

- As *tags* XML padrão e sintaxe textual são fáceis de ser lidas e são claramente superiores a HTML por carregar padrão de informação.

3.6 Persistência de Metadados

Como visto anteriormente, os bancos de dados usados para guardar metadados são geralmente relacionais, mas a arquitetura de metadados tem que ser suficientemente flexível a ponto de permitir que a organização mude de um banco de dados relacional para outro tipo de arquitetura [Vaduva, 2000]. Partindo deste princípio e também com o intuito de testar a persistência dos metadados gerenciados pelo Metha Manager com o uso de outra tecnologia, também foi implementada no presente trabalho uma base de dados objeto-relacional. Em virtude disto, introduziremos a seguir o modelo usado como alternativa de persistência de metadados. No capítulo 7 analisaremos a viabilidade de adoção do modelo objeto-relacional.

3.6.1 Modelo Objeto-Relacional

O modelo objeto-relacional é baseado na idéia de que o modelo relacional deve ser estendido para contemplar conceitos de orientação a objetos. Para isso são acrescentadas estruturas a linguagens de consulta relacionais, como SQL, para tratar os tipos de dados acrescentados. Tais extensões tentam preservar os fundamentos relacionais, em particular o acesso declaratório ao dado, enquanto que estende o poder da modelagem.

Sua principal estrutura ainda é composta por tabelas sendo que com muito mais recursos do que as tabelas puramente relacionais. Entre esses recursos podemos destacar: definição de tipos pelo usuário, registros e vetores, métodos e funções, referências, herança e polimorfismo [Klein, 1999].

Os SGBDOR's são uma inovação de seus antecessores SGBDR's, e, ao contrário da conversão para sistemas de banco de dados orientados a objetos, a migração objeto/relacional não necessariamente envolverá uma mudança completa.

Mostraremos em seguida alguns conceitos que definem um modelo objeto-relacional:

- **Relações Aninhadas:** A primeira forma normal (1FN) exige que todos os atributos tenham domínios atômicos, ou seja, os elementos de seu domínio devem ser unidades indivisíveis [Elmasri & Navathe, 1994]. Entretanto, nem todas as aplicações são compatíveis com esta forma normal. Em vez de enxergar o banco de dados como um conjunto de arquivos, os usuários de certas aplicações enxergam o banco de dados como um conjunto de objetos. Esses objetos podem exigir vários arquivos para sua representação. Pode-se notar que uma interface simples e fácil de usar exige uma correspondência um para um entre a noção intuitiva de um objeto para o usuário e a noção de um item de dado para o banco de dados. O modelo relacional aninhado é uma extensão do modelo relacional em que os domínios podem ser definidos como atômicos ou como relações. Então, o valor de uma tupla sobre um atributo pode ser uma relação, e relações podem ser armazenadas dentro de relações. Então, um objeto complexo pode ser representado por uma ou mais tuplas de uma relação aninhada.

- **Tipos Complexos e Orientação a Objetos:** As relações aninhadas são apenas um exemplo de extensões ao modelo relacional básico. Outros tipos de dados não atômicos, como registros aninhados, também têm se mostrado úteis. O modelo de dados orientado a objeto tem provocado uma necessidade por características como herança e referências a objetos. Sistemas com tipos complexos e orientação a objeto permitem que os conceitos do modelo entidade-relacionamento, como identidade de entidades, atributos

multivalorados, generalizações e especializações, sejam representados diretamente sem uma tradução complexa para o modelo relacional.

- **Herança:** Da mesma forma que as linguagens de programação, o modelo objeto-relacional também suporta o conceito de herança de tipos de dados estruturados definidos pelo usuário, permitindo criar um subtipo de um ou mais tipos existentes. Assim, um subtipo herda especificações de atributos e métodos de todos os supertipos associados. A herança pode ser no nível de tipos ou de tabelas. Será considerado inicialmente o primeiro tipo. Especificada por uma cláusula como *under*, a herança agrupa os tipos em uma hierarquia, permitindo modularidade e reuso na definição dos tipos.

  Temos que considerar ainda que uma hierarquia de tabelas não precisa estar associada a uma hierarquia de tipos. Uma tabela pode herdar atributos diretamente de outras tabelas, sem a necessidade de criação de tipos específicos. A herança torna a definição de esquema mais natural. Sem a herança de tabelas, precisaremos ligar, explicitamente, as tabelas correspondentes às subtabelas, com as tabelas correspondendo às supertabelas via chaves primárias, ou através de tipos referência, e então definir restrições entre elas para assegurar restrições referenciais e de cardinalidade.

- **Tipos Referência:** Assim como as linguagens orientadas a objeto, o modelo objeto-relacional também fornece a habilidade de se referir a objetos. O modelo objeto-relacional permite que o domínio de um atributo seja uma referência para outro de um tipo especificado. Em geral, se uma tabela tem uma coluna do tipo referência, então cada valor da coluna pode referenciar qualquer objeto do tipo referenciado, ou seja, pode-se referenciar objetos em diferentes tabelas. Entretanto, existem recursos que restringem o escopo de referências para uma única tabela.

- **Funções:** Sistemas objeto-relacionais permitem que funções sejam definidas pelos usuários. Essas funções podem ser definidas em uma linguagem de programação como C/C++, Java ou em uma linguagem de manipulação de dados como a SQL. As funções definidas em linguagens de programação podem ser mais eficientes do que funções definidas usando SQL. Além de que

as consultas que não podem ser executadas em SQL podem ser executadas por essas funções. Um exemplo de uso de tais funções seria executar uma complicada operação aritmética sobre o dado em uma tupla.

3.7 Conclusões

Neste capítulo discutimos uma classificação prévia para ser usada como base para a construção de um sistema de administração de metadados. Enfocamos funcionalidade e exigências de estrutura para um repositório, bem como aspectos relacionados à sua arquitetura.

Em se tratando de arquitetura, idealmente, uma empresa deveria ter um único repositório para administrar seus metadados, porém a centralização não condiz com a realidade por diversas razões, como evolução de diversos departamentos que implica desenvolvimento assíncrono de repositório. Em outras palavras e fazendo uma anologia com a construção de um DW, um repositório de metadados é construído idealmente numa metodologia *bottom-up*. Ou seja, o repositório evolui gradativamente até abranger os metadados de toda corporação de uma forma centralizada, assim como os *Data Marts* formam o DW corporativo.

Analisando a questão da extensão dos metadados técnicos, podemos concluir que atributos de metadados técnicos não devem sempre ser incorporados nas classes de um DW. O projetista do repositório, o modelador e o desenvolvedor precisam trabalhar juntos para determinar o que será preciso incorporar em uma classe ou projeto. Adicionar esses atributos no modelo de dados pode afetar o processo ETL, o repositório e as ferramentas de *front-end*.

Por fim, é importante salientar que este capítulo serviu de base para o levantamento de todos os requisitos do Metha Manager apresentados no capítulo 5, bem como serviu para a identificação de diversos e possíveis trabalhos futuros.

Capítulo 4

Abordagem de *Framework* para Desenvolvimento de Software

4.1 Reutilização de Software

À medida que as aplicações de software se tornam mais complexas, tecnologias e ferramentas de apoio que propiciem a reutilização destas aplicações se tornam, progressivamente mais necessárias. Nesse processo de reutilização podemos basicamente nos referir à reutilização de software e reutilização de projeto.

Para entender reutilização de software é necessário diferenciar, em primeiro lugar, o que significa reutilizar um dado artefato em contraste com o que significa utilizar este artefato. De maneira geral, quando se fala em utilização, está implícito o fato que o artefato é utilizado sem realizar nenhuma alteração tanto na estrutura como na forma e contexto no qual ele é utilizado. Quando um artefato é utilizado depois de se fazer alguma mudança, ou é utilizado em uma tarefa ou função diferente daquela para o qual foi originalmente projetado, então este artefato está sendo efetivamente reutilizado. Esta definição coloca em evidência a característica intrínseca da reutilização de software que é a necessidade de adaptação dos artefatos para adequá-los aos requisitos da nova aplicação sendo desenvolvida. Para a reutilização de software existem diversas tecnologias disponíveis e algumas dessas mais populares são as bibliotecas de componentes, orientação a objetos e geradores de aplicações.

Com relação à reutilização de projetos, é importante mencionar que o projeto de um sistema é uma das tarefas mais importantes e complexas dentro do ciclo de desenvolvimento de software. O projeto de um sistema define, essencialmente, a sua divisão em partes, as interfaces entre elas e como estas partes cooperam para implementar a funcionalidade

requisitada. O projeto envolve aspectos de organização de um sistema, como também os aspectos cognitivos das interfaces com o usuário, características específicas de plataformas de hardware que suportarão a implementação, bem como o conhecimento específico sobre o domínio da aplicação.

Considerando esta diversidade de fatores, é razoável afirmar que desenvolver um projeto que forneça um equilíbrio adequado entre todos estes aspectos, não é uma tarefa que possa ser facilmente realizada por projetistas inexperientes. Portanto, a reutilização de projetos existentes se torna muito importante.

Com o advento da orientação a objetos em análise e projeto de sistema no começo da década passada, surgiu um novo paradigma de desenvolvimento no qual a reutilização é parte integral do paradigma. Os mecanismos de herança, polimorfismo e acoplamento dinâmico, habilitaram esta tecnologia para a construção de software por especialização, com base na extensão de funcionalidades de componentes existentes. Através destes mecanismos, é possível definir componentes abstratos, ou semi-acabados, cujo comportamento específico pode ser complementado por subclasses. Deste modo, na criação de componentes específicos, é possível reutilizar tanto o projeto definido pelos componentes quanto o código que implementa este comportamento.

A capacidade de definição de comportamentos abstratos pode ser generalizada para codificar o comportamento genérico de uma família de aplicações. Cada aplicação da família pode ser construída através da implementação de porções de comportamento específicos para ela. Assim, é possível reutilizar, além das classes separadas, classes projetadas em conjunto para resolver a funcionalidade genérica de um domínio de aplicações. Isto significa reutilizar projeto de aplicações.

Um conjunto de classes que fornecem uma solução genérica para um determinado domínio de aplicação é chamado de *framework* orientado a objetos. Um *framework* é composto de um conjunto de classes que definem um projeto abstrato para soluções de problemas de uma família de aplicações dentro de um domínio. Um *framework* implementa, em termos de classes, o comportamento genérico de um domínio de aplicação, deixando a implementação dos aspectos específicos de cada aplicação para serem complementados com subclasses. Deste modo, um *framework* funciona como um molde para a construção

de aplicações ou subsistemas, dentro de um domínio de aplicação. Analisaremos o conceito de *framework* mais profundamente a seguir.

4.2 *Framework* Orientado a Objetos

4.2.1 Conceito

Framework pode ser definido sob dois aspectos [Roberts, 2000]: estrutura e função. Quanto à estrutura, um *framework* é a reutilização do projeto de parte ou totalidade de um sistema representado por classes abstratas e pela forma como instâncias destas classes interagem.

Em outras palavras, a estrutura de um *framework* consiste de um diagrama de classes e um diagrama de colaboração. Quanto à função, um *framework* é o esqueleto de uma aplicação que pode ser personalizado por um desenvolvedor de aplicação. A função do *framework* é extrair e disponibilizar a essência de uma determinada família de aplicações, de modo que o desenvolvedor possa partir de um patamar superior de implementação da arquitetura. Neste sentido um *framework* tem a mesma função que uma API (*Application Programming Interface*).

4.2.2 Classificação

Os *frameworks* podem ser classificados em caixa-branca e caixa-preta. Os *frameworks* caixa-branca baseiam-se nos mecanismos de herança e ligação dinâmica (*dynamic binding*) presentes em orientação a objetos. Os recursos existentes em um *framework* caixa-branca são reutilizados e estendidos a partir de: herança de classes do *framework* e sobrecarga (*overriding*) de métodos pré-definidos. Os métodos são definidos em interfaces ou classes abstratas e devem necessariamente ser implementados por uma aplicação.

Frameworks caixa-preta são baseados em componentes de software. A extensão da arquitetura é feita a partir de interfaces definidas para componentes. Os recursos existentes são reutilizados e estendidos por meio de definição de um componente adequado a uma interface específica e integração de componentes em um *framework* que utiliza padrões de projeto que serão abordados a seguir [Gamma et al., 1994].

4.2.3 *Frameworks* Versus Biblioteca de Classes

Na reutilização de componentes de biblioteca (funções, procedimentos ou classes) um programador deve desenvolver a estrutura de controle que invoca estes componentes; utilizando um *framework*, ao contrário, o programador deve desenvolver código que será invocado pelo *framework*. A estrutura de controle da aplicação é codificada pelas classes do *framework*, as quais invocam métodos que devem ser implementados para cada aplicação em particular. Assim, aplicações específicas são construídas especializando-se as classes do *framework* para fornecer a implementação de alguns métodos, embora a maior parte da funcionalidade da aplicação seja herdada.

Um *framework* ainda pode ser visto como uma espécie de biblioteca, só que o controle do fluxo de chamadas é *bi-direcional*, ou seja, tanto uma aplicação pode chamar métodos do *framework* como o *framework* pode invocar métodos da aplicação. Isto é diferente de uma biblioteca de classes *stricto sensu*, em que o fluxo de chamadas é uni-direcional, da aplicação para a biblioteca. A possibilidade de o *framework* chamar métodos da aplicação se dá através de chamadas "dynamic binding", implementadas nas subclasses da extensão do *framework* para a aplicação.

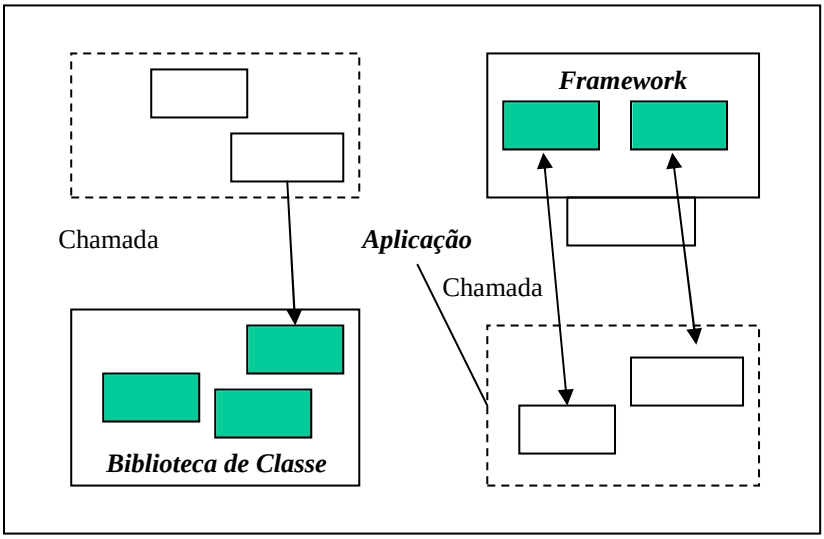

Figura 4.1: Diferença no fluxo de controle entre *Framework* e bibliotecas de classes [Landin, 1995].

Na figura 4.1 podemos observar a ilustração de uma aplicação possuidora de classes que fazem chamadas às classes pertencentes a uma biblioteca de classes e de outra aplicação que utiliza classes de um *framework* que fazem as chamadas às classes da mesma.

4.3 Padrões de Projeto

O desenvolvimento de software ainda hoje é acompanhado de uma grande escassez na documentação dos problemas e nas soluções encontradas para solucioná-los. Com isso, problemas que muitas vezes se repetem, geram esforços adicionais para a implementação de suas soluções. As tentativas de reuso das boas soluções e do acúmulo de experiências sobre determinados problemas são, na maioria das vezes, iniciativas isoladas de alguns desenvolvedores.

O uso de padrões de projeto vem preencher esta lacuna que existe, tornando-se um mecanismo eficiente no compartilhamento de conhecimento entre os desenvolvedores. A meta é a criação de uma linguagem comum, que permita uma comunicação efetiva no que se refere à troca de experiências sobre problemas e suas soluções. Desta forma, soluções que se aplicaram a situações particulares, podem ser novamente aplicadas em situações semelhantes por outros desenvolvedores. Codificando estas soluções, estamos também capturando o conhecimento do sistema, indo ao encontro das necessidades dos usuários. A criação de um repositório de padrões de projeto ajuda tanto ao desenvolvedor experiente quanto ao novato, no reconhecimento de situações nas quais o reuso poderia ocorrer. O foco principal não é nem tanto tecnológico, mas sim o de criar uma cultura de catalogação de experiências e soluções para apoiar o desenvolvimento de software.

4.3.1 Definição

Um padrão de projeto pode ser visto como uma solução para um problema em um determinado contexto [Gamma et al., 1994]. A solução tenta capturar a essência do problema, a fim de que outros desenvolvedores a entendam, e possam fazer uso dela em situações similares.

Em termos de orientação a objetos, padrões de projeto identificam classes, instâncias, seus papéis, colaborações e a distribuição de responsabilidades. Seriam, então, descrições de classes e objetos que se comunicam, que são implementados a fim de solucionar um problema comum em um contexto específico.

O padrão de projeto faz mais do que identificar a solução, ele explica porque a solução é necessária. Além disso, padrões de projeto têm vários usos no processo de desenvolvimento de software orientado a objetos:

- Formam um vocabulário comum que permite uma melhor comunicação entre os desenvolvedores, uma documentação mais completa e uma melhor exploração das alternativas de projeto. Reduz a complexidade do sistema através da definição de abstrações que estão acima das classes e instâncias. Um bom conjunto de padrões aumenta muito a qualidade do programa;

- Constituem uma base de experiências reutilizáveis para a construção de software. Funcionam como peças na construção de projetos de software mais complexos; podem ser considerados como micro-arquiteturas que contribuem para arquitetura geral do sistema;

- Reduzem o tempo de aprendizado de uma determinada biblioteca de classes. Isto é fundamental para o aprendizado dos desenvolvedores novatos;

- Quanto mais cedo são usados, menor será o retrabalho em etapas mais avançadas do projeto.

4.3.1 Requisitos

Na maioria dos casos um padrão deve ser aplicado em três situações comprovadas para receber certificação. Nem toda solução, algoritmo ou prática constitui um padrão. Embora possa parecer que possua todos os requisitos básicos para ser um padrão, ele não pode ser considerado como tal até que o fenômeno da repetição seja verificado.

Devemos usar padrões de projeto em soluções que se repetem com variações. Não faz sentido o reuso, se o problema só aparece em um determinado contexto. Em geral, os padrões de projeto representam soluções que requerem vários passos. Se o número de passos for pequeno, não há a necessidade de usarmos padrões.

Padrões de projeto se aplicam muito bem em situações em que o desenvolvedor está mais interessado na existência de uma solução do que na sua total implementação. Isto ocorre, na maior parte das vezes, quando o domínio do problema é o foco da questão.

Padrão de projeto pode ser visto como extensão da definição de classes. Em orientação a objetos, as classes têm dois aspectos básicos, análogos aos padrões de projeto:

- Externo, a visão do problema : descrições das propriedades, responsabilidades, capacidades e serviços prestados ao software ou ao ambiente externo;

- Interno, a visão da solução: descrições estáticas e dinâmicas, limitações, e relacionamentos entre os componentes, colaboradores, ajudantes, cada um apenas com uma visão externa incompleta.

Padrões de projeto aumentam a expressividade e o nível de descrição suportados pela orientação a objetos. Inversamente, os conceitos de OO podem ser aplicados aos padrões:

- Encapsulamento e Abstração: Cada padrão engloba um problema em um determinado contexto e a sua solução, com limites bem definidos entre o espaço do problema e da solução. Também corresponde a uma abstração que abrange conhecimento e experiências sobre o domínio da aplicação.

- Extensibilidade e Adaptabilidade: Cada padrão deve ser extensível para que outros padrões possam moldá-lo a fim de resolver um problema mais amplo. Deve ser também adaptável para que sua solução possa servir em uma vasta gama de implementações.

- Geração e Composição: Cada padrão, uma vez aplicado, gera como consequência um contexto inicial de um ou mais padrões. Estes poderão ser aplicados em conjunto, com o objetivo de alcançar uma solução global e completa.

- Equilíbrio: Os padrões devem procurar atingir um equilíbrio entre suas forças e restrições. Deve-se minimizar os conflitos dentro do espaço de solução do problema,e dar um porquê para cada passo ou regra dentro do padrão

Podemos considerar como requisitos básicos de um padrão as seguintes características:

- Resolve o problema, e não apenas indica princípios e estratégias;

- É um conceito provado, não constituindo apenas uma hipótese ou uma especulação;

- A solução não é óbvia;
- Não apenas descreve os módulos, mas como eles se relacionam, através de estruturas e mecanismos do sistema;
- Tem um significante componente humano: atinge a qualidade quer seja na clareza, na utilidade ou na documentação.

4.3.1 *Framework* e Padrões de Projeto

Os conceitos de framework e padrões de projeto são estreitamente relacionados, sendo ambos viabilizados pela orientação a objetos.

Recapitulando, sabemos que um *framework* difere das bibliotecas de programação porque possui um fluxo inverso de controle, isto é, é ele que determina quais objetos e métodos devem ser usados quando da ocorrência de eventos.

Ao contrário dos *frameworks,* os padrões de projeto permitem o reuso de micro-arquiteturas abstratas sem uma implementação concreta. Os *frameworks* são implementados em uma linguagem de programação, enquanto que os padrões constituem formas de usá-las. Em geral, os padrões de projeto são menos especializados que os *frameworks*, podendo ser aplicados em um número maior de aplicações. Eles ajudam na definicão e na documentação de um *framework*. Normalmente, *frameworks* maduros englobam dezenas de padrões de projeto, e estes podem participar de diferentes *frameworks.*

Os padrões de projeto mais comuns utilizados em projetos de *framework* são: “Façade”, “Command”, “Template Method”, e “Abstract Factory” [Gamma et al., 1994].

Dentre os padrões citados, “Template Method” é a base para a construção de um *framework*.. O “Template Method” define a estrutura de um algoritmo de uma operação, transferindo alguns passos desta operação para as subclasses. Permite que uma subclasse redefina certos passos de um algoritmo sem alterar a estrutura do mesmo.

4.4 Processo de Desenvolvimento de um *Framework*

O processo de desenvolvimento apresentado nesta seção é o mesmo apresentado e utilizado com sucesso em [Jorge, 2001] e fundamentado em [Landin, 1995]. Este trabalho

também segue a linha do processo unificado interativo e incremental [Jacobson *et al*., 1998], com algumas mudanças importantes para contemplar as diferenças em desenvolver um *framework* em relação a uma aplicação convencional.

Durante todo o processo de desenvolvimento do *framework* o objetivo maior é o de migrar o máximo de comportamento comum das aplicações para o *framework.*

Antes de começar o projeto de um *framework*, recomenda-se implementar de forma convencional no mínimo duas aplicações relacionadas com o domínio da aplicação. As aplicações a serem desenvolvidas devem possuir funcionalidades distintas. O motivo para o desenvolvimento das aplicações está na obtenção de conhecimento sobre o comportamento genérico (*framework*) e do conhecimento do comportamento específico das aplicações. Para desenvolvedores com experiência no domínio das aplicações a serem mapeadas pelo *framework* esta fase pode ser eliminada.

A figura 4.2 apresenta graficamente as etapas do processo de construção do *Framework.*

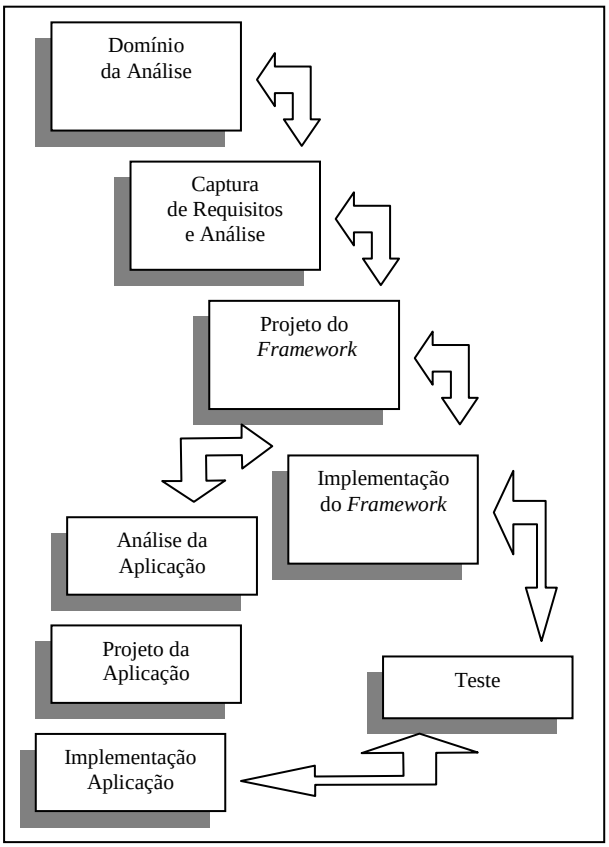

Figura 4.2: Etapas do processo de desenvolvimento do *Framework* [Landin, 1995].

As próximas subseções resumem cada etapa do processo.

4.4.1. Domínio da Análise

Semelhante ao desenvolvimento de uma aplicação convencional, o Domínio da Análise consiste na primeira fase do desenvolvimento do *framework*. É ela que provê informações conceituais sobre o *framework*, que serão usadas nas etapas subseqüentes. O objetivo é determinar exatamente o propósito do projeto, explicitando todos os seus limites. Referencia-se o que será contemplado e o que não será contemplado. É relevante também citar quais os sistemas farão interface com o projeto em questão.

As informações colhidas nesta fase devem estar focadas sobre o domínio do problema. Não deve haver preocupação com detalhes de mais baixo nível ou de implementação, ou seja, devemos pensar no *quê* é necessário e não *como* implementar.

Os documentos gerados nesta etapa servem como forma de comunicação entre o cliente e os desenvolvedores do sistema.

Nesta etapa são gerados pelo menos dois documentos, o *escopo* e um *modelo estático* contendo os objetos e classes relacionadas com o domínio da aplicação.

4.4.2 Captura de Requisitos e Análise

Nesta fase são discriminados todos os requisitos do sistema. Os requisitos especificam as regras de negócio e os serviços que o sistema deve prover. Para identificar os requisitos de um *framework* deve-se listar, preliminarmente, todos os requisitos de, no mínimo, duas aplicações dentro do escopo do *framework* [Landin, 1995].

O processo de listar os requisitos segue a premissa de achar as *generalizações*. E conduzido a partir da seguinte estratégia:

- Achar todos os requisitos específicos e isolá-los;
- Agrupar todos os requisitos comuns e relacioná-los com o *framework*.

Segue-se o *detalhamento dos requisitos*. No processo de detalhamento dos requisitos, deve-se diferenciá-los em *funcionais* ou *não-funcionais*. Requisitos funcionais estão diretamente relacionados com as funcionalidades da aplicação. Já os requisitos não-

funcionais estão associados com características como: desempenho, segurança, integração, etc.

Os requisitos não-funcionais de um *framework* podem ajudar na sua extensão. Alguns padrões são criados com o intuito de ajudar o usuário implementador na tarefa de construir aplicações com base no *framework*. Por exemplo, pode-se levantar um requisito não-funcional onde o *framework* siga padrões de nomes de classes e métodos.

Portanto, os requisitos comuns são encapsulados no *framework*, ficando para as aplicações os requisitos específicos. A figura 4.3 o ilustra.

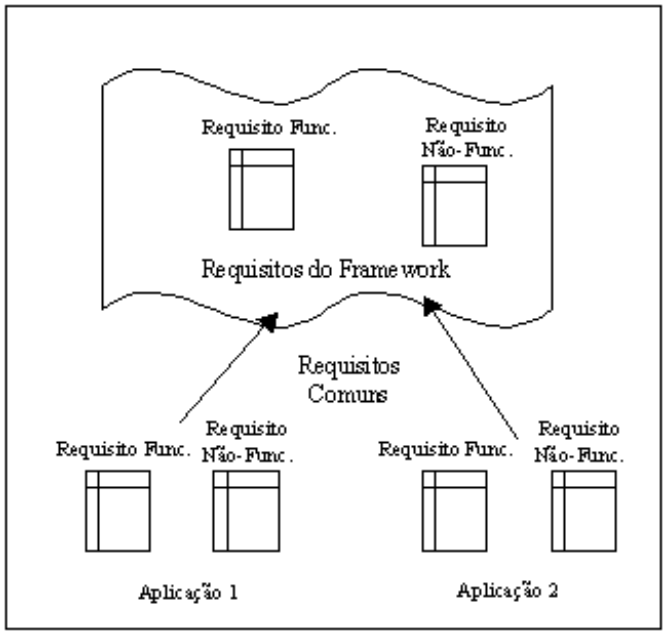

Figura 4.3: Separação dos requisitos das aplicações e dos requisitos pertencentes ao *Framework* [Landin, 1995].

Depois de listar e detalhar os requisitos, é ainda realizada a atividade de construção do *Diagrama de Casos de Uso*. Os requisitos funcionais, identificados e detalhados na etapa anterior, são a base para a construção dos Casos de Uso.

Neste tipo de diagrama, os Atores são entidades externas ao sistema com interesse em interagir com o sistema para obter algum resultado. Atores têm instâncias chamadas de usuários. Por exemplo, o ator Cliente tem como instâncias todos os clientes que estão interessados em interagir com um Sistema de Pedido. Para elucidar, pode-se fazer uma correlação entre classes e seus objetos e Atores e seus usuários (instâncias do Ator).

Descrevemos as características do diagrama de Caso de Uso que ajudam na modelagem de um projeto de *framework*. Temos nesse diagrama relacionamentos que proporcionam criar modelos com alto grau de reutilização. Existem dois tipos de relacionamentos entre Casos de Uso: “Includes” e “Extends”.

O relacionamento ”Includes” é utilizado quando o caminho de um Caso de Uso é incluído em outro. O relacionamento “Extends” é utilizado quando um Caso de Uso especializa um outro Caso de Uso de uma forma mais controlada, ou seja, o Caso de Uso base declara um número de pontos de extensão.

Os relacionamentos entre Casos de Uso no projeto de um *framework* permitem projetar funcionalidades genéricas com escopo de aplicação que poderão ser reutilizadas.

Os Casos de Uso num projeto de *framework* podem ser concretos ou abstratos. Os concretos são usados diretamente pelos os Atores. Já os abstratos nunca serão usados diretamente pelos Atores e para cumprirem suas funcionalidades necessitam que sejam estendidos (“extends”) por outros Casos de Uso.

Os Casos de Uso abstratos definidos num *Framework* são estendidos pelos Casos de Usos das aplicações (que estendem o *Framework*).

4.4.3 Projeto do *Framework*

Esta fase tem como objetivo realizar os requisitos identificados nas fases anteriores. Na fase de análise, o escopo foi o trabalho de levantamento e compreensão dos conceitos e dos requisitos relacionadas com o *framework*, e das aplicações construídas com base nele. Na fase de projeto, o foco foi desenvolver modelos que ajudassem a construir uma solução computacional que atenda aos requisitos modelados.

Os artefatos da fase de projeto são: projeto arquitetural, diagrama de classes, diagrama de estados e diagrama de colaboração.

Projeto Arquitetural

O projeto arquitetural visa decompor o sistema em subsistemas menores para reduzir a complexidade do problema original. São identificadas e definidas camadas (subsistemas), módulos (partes de subsistemas) e interdependências.

Uma estrutura em camadas é algo desejável, pois aumenta a escalabilidade e reduz o escopo do problema. Uma camada é um subsistema que interage somente com uma camada vizinha. Dentro das camadas temos os módulos (pacotes). Os módulos agrupam unidades funcionais (classes) que têm correlações.

No projeto arquitetural, define-se modelos gráficos com a disposição das camadas em sistemas computacionais. Para as camadas que estão em sistemas computacionais distintos deve-se elucidar como é feita a comunicação entre elas (RMI, ODBC, HTTP, etc).

O projeto arquitetural é modelado para atender aos requisitos não-funcionais identificados na fase de análise, observando critérios como escalabilidade/performance.

O critério de "particionamento" dos subsistemas em módulos é de acordo com o nível de acoplamento das funcionalidades. Portanto, dentro dos módulos as classes têm um maior nível de acoplamento. Já o acoplamento entre as classes dos módulos externos é mais fraco, isolando assim a complexidade. O objetivo é criar uma arquitetura em que o projeto a ser desenvolvido esteja preparado para sofrer manutenções evolutivas de forma que uma alteração não cause problemas não esperados.

Outra questão importante no projeto arquitetural é que nesta etapa pode-se identificar a existência de mais de um *framework* no projeto. Esta situação é muito comum em grandes projetos.

Projeto de Baixo Nível

O objetivo é a definição de modelos que permitam ajudar na implementação do *framework*. Nesta fase são criados os diagramas de classe, de estado e de colaboração. Para a construção dos diagramas são utilizados os objetos e classes de negócio identificados na análise.

Listamos as principais atividades:

- Novos objetos e classes são identificados;
- Associações entre classes são transformadas em referências, agregações e heranças;
- Definição de métodos e atributos;

- Estudo do ciclo de vida das classes com comportamentos mais relevantes (diagramas de estado);

- Aplicação de padrões de projetos.

O diagrama de classes do *framework* deve prover a base para a criação de uma implementação genérica, que será usada por várias aplicações similares [Landin, 1995]. Assim, provê-se classes que `encapsulam´ comportamento genérico das futuras aplicações.

4.4.4 Implementação e Teste do *Framework*

Nesta fase, é feita a codificação das classes projetadas na fase anterior numa linguagem de programação. Projeta-se também nesta fase algumas extensões para teste do *framework*. Visto que *framework* é um pedaço de código que não pode ser instanciado sem que uma extensão (aplicação) esteja a ele acoplada. O desenvolvimento do projeto das extensões contempla as fases de Análise, Projeto e Implementação como está ilustrado na figura 4.2. A implementação das extensões permite finalmente instanciar o *framework*.

O processo de implementação do *framework* e das extensões é interativo. Ou seja, pode-se mudar funcionalidades que estão encapsuladas nas aplicações para o *framework* e vice-versa. Essas mudanças são determinadas pelos testes que validam a solução.

Ressalta-se que ao longo da codificação do *framework* deve-se documentar todos os atributos e métodos das classes.

4.5 Conclusões

Neste capítulo abordamos em nível de engenharia de software toda fundamentação que serviu de base para a definição e construção do Metha Manager, objetivo principal deste trabalho.

Para a construção do Metha Manager foram realizadas as etapas de análise, projeto, implementação e testes. Na construção do projeto, utilizamos a técnica de projeto de software, *Framework*, associada com padrões de projeto e orientação a objetos como os principais paradigmas para a definição e construção do Gerente.

Como processo de desenvolvimento, adotamos o Processo Unificado Iterativo e Incremental descrito em [Jacobson *et al*., 1998].

Os próximos dois capítulos tratam de apresentar em detalhes o Metha Manager.

Capítulo 5

Requisitos e Análise do *Framework* Metha Manager

Neste capítulo descreveremos a análise e o projeto conceitual de um Gerente de Metadados Extensível – Metha Manager – objetivo principal deste trabalho. Utilizaremos a linguagem UML (*Unified Modeling Language*) [Booch *et al*., 1999] [Fowler & Scott, 1999] como mecanismo de representação para especificar e documentar as diversas fases do desenvolvimento desse projeto. Adotamos uma abordagem iterativa e incremental. Basicamente cada camada do Metha Manager apresentada no decorrer deste capítulo representa um novo incremento do desenvolvimento do projeto. A linguagem de programação Java [Flanagan, 1997] foi escolhida para a fase de implementação em função de sua simplicidade e crescente aceitação, características que tornam o código fonte em mais um artefato de documentação.

5.1 *Business Case* – Contexto do Metha Manager

O gerenciamento de metadados é um mercado atraente para repositórios centrais, diversas empresas como Rochade [5] e Platinum [6] possuem soluções com metamodelos "genéricos" construídos para a perspectiva deles de um processo de *Data Warehousing*. Na maioria dos casos, este modelo "genérico" não possui documentação disponível, complicando assim uma comparação detalhada e em particular uma possível adoção destas propostas por outras ferramentas.

Mesmo com a existência de ferramentas administrando metadados em locais especializados ainda permanece obscuro se a meta de um repositório central acessado por qualquer aplicação e usuário será alcançada. É praticamente impossível possuir uma

[5] http://www.viasoft.com

[6] http://www.platinum.com

representação geral de metadados empresariais de uma companhia. Integrações virtuais baseadas em um metamodelo de integração que mapeia metadados de ferramentas para um metamodelo padrão é uma meta muito ambiciosa, procurada, por exemplo, pela Ardent [7] com sua tecnologia chamada de *metabroker*. Infelizmente, o modelo de integração usado para *metabrokers* em desenvolvimento só permanece implícito e não é administrado por um componente de controle independente.

Em conseqüência, investir em desenvolvimento próprio de um gerente de metadados continua sendo uma decisão altamente recomendável. Sendo assim, o Metha Manager se propõe em fornecer um modelo de metadados aberto e extensível que poderá representar metadados genéricos de negócio, bem como ser facilmente adaptado para prover metadados às diversas ferramentas que venham a ser utilizadas em um processo de *Data Warehousing.* Isto é possível pelo fato do Metha Manager ter sido construído segundo a tecnologia de *framework* discutida no capítulo 4.

Listamos as principais vantagens na adoção da técnica de projeto de software, *framework*, na construção do Metha Manager:

- Em âmbito geral, o *framework* Metha Manager possibilita criar uma estrutura genérica para a parte comum das aplicações de metadados. Na sua essência, o Metha Manager é constituído de *classes abstratas*, que são estendidas com o comportamento especifico das aplicações, quaisquer que sejam os modelos conceituais específicos utilizados, bem como a tecnologia de armazenamento ─ ROLAP, OROLAP, MOLAP, etc.

- O *Framework* Metha Manager é capaz de prover metadados para aplicações OLAP que executam em qualquer SGBD, seja ele relacional, ou objeto-relacional, ou puramente orientado a objeto, ou ainda um SGBD baseado em arrays multidimensionais (tecnologia MOLAP).

- Para a provisão de metadados a uma nova aplicação OLAP, reutiliza-se a análise, o projeto, o código e os testes do *framework* Metha Manager, alcançando-se assim uma alta produtividade de desenvolvimento.

[7] http://www.ardentsoftware.com

5.1.1 Trabalhos Relacionados

A globalização das corporações em um ambiente de negócios altamente competitivo exige um fluxo de informações rápido e eficiente. Para isso, é necessário a integração de informações e seu conseqüente gerenciamento como forma de diminuir o custo e tempo despendido na implementação de novas soluções. Contudo, esse gerenciamento integrado encontra obstáculos na proliferação de ferramentas de gerenciamento e manipulação de dados que, em sua maioria, processam metadados de maneira proprietária. A solução para esse problema apresenta-se na forma de padrões para definição e troca de metadados, o que permite a reutilização de dados entre diferentes aplicações. Podemos citar dois principais padrões que possuem esse objetivo: MDIS (*Metadata Interchange Specification*) [Coalition, 1998] e OIM (*Open Information Model*) [Coalition, 1999] ambos pertencentes à Metadata Coalition, um consórcio não-lucrativo de vendedores e usuários finais que tem como objetivo prover uma especificação não proprietária dos metadados corporativos. O MDIS provê conceitos para suportar a troca de metadados relacionados com diferentes tipos de repositórios de dados: relacionais, multidimensionais, em rede, hierárquicos e baseados em arquivos. Contudo, sua especificação restringe-se aos relacionamentos existentes em um esquema de dados. Em contrapartida, o OIM apresenta modelos de informações que englobam todas as fases do desenvolvimento de sistemas de informação. Um desses modelos é o modelo de banco de dados, que fornece conceitos relativos a provedores de dados relacionais.

Mesmo assim a padronização de metadados ainda não está estabilizada, pois padrões para aplicações especificas requerem um alto grau de detalhe e regulamentação mais precisa que padrões de propósito geral. Por exemplo: A Microsoft [8] está oferecendo seu metamodelo baseado no OIM, porém sabemos que o OIM só cobre um DW relacional e não é uma especificação de um repositório ou implementação – o foco é na descrição da informação e não no acesso e gerenciamento dos dados.

Seguindo na busca por padronização diversas empresas como a Oracle e Unisys deixaram a Metadata Coalition e estão propondo o CWM (*Common Warehouse Metadata*) [Giovinazzo, 2000] [OMG, 2001]. O mecanismo é baseado no Intercâmbio XML de Metadados (XMI) especificamente. XMI usa os padrões da XML. Em outras palavras, todo

[8] http://www.msdn.microsoft.com/repository/

atributo de um meta-modelo é representado na Definição de Tipo de Documento (DTD) por um elemento de XML cujo nome é o atributo. Cada associação entre duas meta-classes, são representadas por dois elementos de XML que representam os papéis na associação. As multiplicidades da associação estão em multiplicidades de XML que são válidas para especificar os modelos de elementos de XML.

Como está em fase de proposição, ainda não se sabe qual nível de detalhe e cobertura pode ser esperado pelo mesmo, ou seja, até agora não está claro se o mesmo será um padrão estrutural que define um metamodelo para *Data Warehouse* ou apenas um padrão de troca. O CWM não é a solução para a integração funcional dos repositórios de dados, mas pode vir a facilitar esta tarefa por fornecer metadados padronizados [Iyengar, 2000].Comparando o CWM com o OIM temos como principal diferença a amarração do pacote OLAP do OIM ao modelo relacional [Vetterli, 2000]. As diferenças são interessantes de serem vistas, porém já é sabido que a Metadata Coalition forneceu suas especificações para o CWM e parece ter desistido de continuar com a tarefa árdua de construir um "padrão".

Todas as pesquisas com relação a metadados ao invés de procurar o estabelecimento de um esquema global de metadados estão dirigidas para aspectos específicos tais como: representação multidimensional, aspectos de qualidade ou negócio e relatórios que na verdade são lacunas deixadas pelo CWM e OIM.

Os vendedores de repositório fizeram propostas próprias para esquemas de metadados para *Data Warehouse e* a maioria dos vendedores de ferramentas seguem soluções descentralizadas e proprietárias. A idéia de um esquema unificado suportado por todas as partes envolvidas não possui aceitação total.

Em relação a tipos de metadados, diversos pontos ainda não foram considerados por nenhuma proposta de padrão:

- Um modelo seguro para representação de sistemas fonte e respectivo DW de destino;
- Conteúdo de aplicações cliente e tempo de execução;
- Suporte para procura “on-line”, explicações e suporte interativo para formulação de consultas;

- Observações sobre transformações efetuadas nas fontes de dados.

Além disso, um Dw não é apenas a especificação de processos complicados para serem executados em dados, mas um processo cuja criação, manutenção e evolução podem ser descritas através de metadados. A manipulação destes aspectos e o acoplamento a um esquema global de metadados ainda não foram tratados.

Em outras palavras, apesar de algumas tentativas de criar padrões de metadados para DWing ainda não existe um padrão de fato aceito [Vetterli, 2000]. As ferramentas de DWing disponíveis no mercado suportam a gerência parcial de metadados, algumas tendo seu foco na consulta a dados, outras na extração de dados. Inexiste pois um padrão de metadados que atenda a todos os elementos da arquitetura.

Em [Stöhr *et al*., 1999], é apresentado um modelo integrado e uniforme para gerenciar metadados em ambiente de DWing. Metadados são divididos em metadados semânticos (ou metadados conceituais) e metadados técnicos (lógicos, físicos e para descrever a carga do DW). As dependências entre metadados semânticos e técnicos também são tratadas. Infelizmente, os metadados lógicos são exclusivamente relacionais, e o autor não discute sobre a extensibilidade do seu modelo para outras tecnologias (ou como estender o seu modelo).

5.1.1.2 O metamodelo CWM

O principal objetivo do CWM é facilitar a troca de metadados de *warehouse* e *business intelligence* entre ferramentas, plataformas e repositórios em um ambiente distribuído. Dessa forma, o CWM pode ser classificado como um padrão para representação e troca de metadados. Nele é especificado um metamodelo que pode ser visto como um esquema conceitual para representação de metadados.

O padrão CWM é uma proposta do OMG que se baseia em outros três padrões do mesmo grupo.

O Object Management Group (OMG) é uma organização internacional, fundada em 1989, que hoje é formada por mais de 800 membros, dentre eles fabricantes de software e usuários. O objetivo maior do OMG é promover a interoperação de sistemas heterogêneos

distribuídos utilizando tecnologia orientada a objetos. Seu principal produto consiste em especificações voltadas para a indústria de software e independentes de fabricante.

Dentre as várias especificações (e padrões) desenvolvidas/adotadas pelo OMG estão o MOF (Meta Object Facility), a UML (Unified Modeling Language) e o XMI (XML Metadata Interchange) que, juntos, formam o núcleo da arquitetura de metamodelagem do OMG, na qual o CWM está inserido, sendo um metamodelo de domínio específico.

O CWM é constituído de vários sub-metamodelos, que representam metadados comuns às principais áreas da tecnologia de *data warehousing.* Essas áreas funcionais foram classificadas em:

- *Resources* – suporta a definição de vários tipos de dados fontes e alvos, é composta por metamodelos que representam fontes relacionais, orientadas a objetos, registros, multidimensionais e XML;

- *Analysis* – define as transformações e processamentos analíticos que ocorrem nas fontes de dados. Abrange os metamodelos que representam transformações de dados, OLAP (O*n-Line Analytical Processing*), mineração de dados, visualização de informações e nomenclatura de negócios;

- *Management* – utilizado quando na definição de metadados para registrar informações de *scheduling* e detalhes operacionais tais como o resultado de transformações. Consiste dos metamodelos *Warehouse process* e *Warehouse operation*;

- *Foundation* – provê os tipos de metadados para representação de conceitos e estruturas que são compartilhados por outros pacotes do CWM;

- *Object Model* – neste pacote estão definidos os construtores básicos para a criação e descrição de todas as classes do metamodelo em todos os outros pacotes. O *ObjectModel* é um subconjunto da UML que inclui apenas as características necessárias para criar e descrever o CWM.

Apesar da existência do CWM, o mesmo não foi adotado neste trabalho por razões de reuso dos trabalhos envolvidos na construção do *Framework* e pela não completude do mesmo. No CWM ainda inexiste uma linguagem de consulta a metadados. Contudo, as

classes do nosso modelo são um subconjunto das classes do CWM, facilitando assim uma futura adoção.

5.1.2 Objetivos do Metha Manager

O Metha Manager é uma ferramenta de gerência de metadados para ambiente de DWing que, em sua versão atual, contempla todas as camadas de um ambiente de Dwing. As funcionalidades disponíveis no Metha Manager para a gerência de metadados visam atender a seus diferentes tipos de usuário: usuários de negócio e usuários técnicos em geral(projetistas de DW, programadores de aplicação e administradores de banco de dados).

O repositório de metadados contém informações sobre estruturas, conteúdos e interdependências entre os componentes do sistema. Um importante requisito do projeto do repositório é a sua extensibilidade, no sentido de tornar o Metha Manager facilmente "customizado" para tecnologias ROLAP ou MOLAP de construção de DW, ou para futuras tecnologias de DW. Além disso, é possível extender o Metha Manager para o armazenamento de modelos conceituais diferentes suportados pelas diversas ferramentas de um ambiente de DWing.

A facilidade da “customização” é possível porque o Metha Manager é um software reutilizável, desde o seu projeto conceitual até o nível de código. Trata-se, portanto, de uma ferramenta para aumentar a produtividade na construção de aplicações para gerência de metadados.

O Metha Manager propõe um modelo de representação de metadados que:

- Integre as diferentes camadas de metadados de um Servidor de DW (Capítulo 1);
- Seja um *framework* caixa-branca desenvolvido com a técnica detalhada no Capítulo 4 para prover extensão de metadados.

Também são objetivos do Metha Manager:

- Implementação de classes persistentes do repositório de metadados, usando o SGBDOR –Oracle8i. As classes persistentes poderão ser armazenadas em uma base relacional, bem como numa base objeto-relacional;

- Implementação de interfaces web amigáveis e personalizadas para os diferentes usuários do gerente de metadados;

- Possuir tratamento de níveis de segurança para acesso ao repositório;

- Alcance gradativo de níveis de integração com o dicionário de dados do SGBD Oracle 8i;

- Tratamento da consistência dos metadados de negócio através de um controle dos responsáveis pela inclusão de valores de atributo. Em diversos casos, a interpretação dos dados é feita de maneira diferente em cada grupo, pois cada departamento tem suas próprias regras e conceitos;

- Exportar metadados através de XML.

Na seqüência descreveremos a arquitetura do Metha Manager, bem como seus requisitos funcionais e não funcionais.

5.1.3 Arquitetura do Metha Manager

Com o intuito do gerente, através de extensões, poder representar qualquer modelo conceitual para *Data Warehouses,* foi utilizada uma "arquitetura de quatro níveis" para metamodelagem [Rational, 1999], ilustrada na figura 5.1, comumente usada para descrever os diferentes níveis de informação dentro do domínio de modelagem.

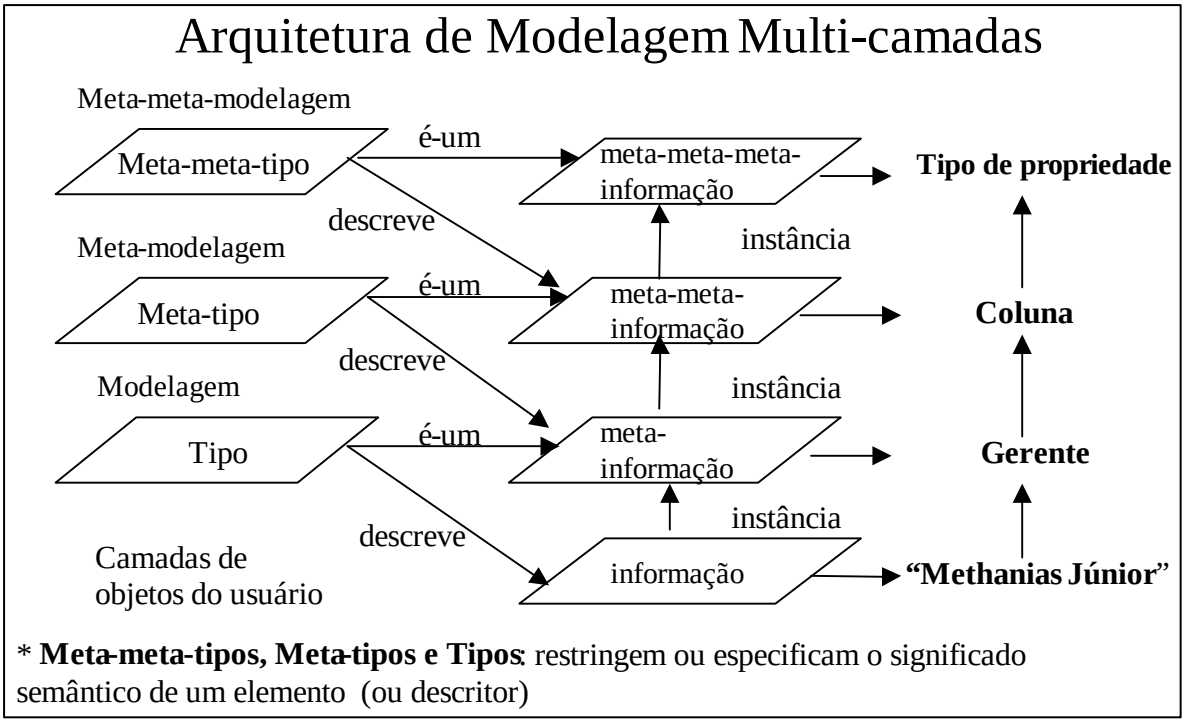

Figura 5.1: Arquitetura de Modelagem Multi-camadas

Na figura 5.2 podemos observar a materialização no Metha Manager de uma arquitetura de modelagem multi-camadas proposta por nossa solução.

A camada "Dados do Usuário" refere-se a um domínio de informação específico. Dentro do escopo do Metha Manager, podemos considerar como instâncias desta camada os dados de DWs armazenados em Bancos de Dados.

A camada técnica(Esquemas) contém os esquemas lógicos resultantes da aplicação de um modelo conceitual, como o de Agrawal [Agrawal, 1999] utilizado em [Jorge, 2001], sobre um determinado domínio de informação. Esta camada engloba as camadas de implementação e operacional citadas em [Jorge, 2001] (Capítulo 1). Os dados operacionais(fontes de dados operacionais que alimentam os diversos DWs e os serviços de extração, validação e carga) presentes nesta camada não foram considerados em [Jorge, 2001] e [Nascimento, 2001].

A camada semântica(Meta-esquemas) contém os esquemas conceituais [Jorge, 2001] [Nascimento, 2001] para representação de Bancos de Dados Multidimensionais. A camada semântica pode ser considerada como sendo o "Nível dos Modelos".

A camada de negócio(Metameta-esquema) define uma linguagem para especificar meta-esquemas. Ou seja, através de um determinado metameta-esquema deve ser possível

criar instâncias de meta-esquemas, como também, por exemplo, o meta-esquema do Modelo Conceitual de Agrawal [Agrawal, 1999]. Foi considerada para essa camada uma forma de representação “genérica” para um negócio a ser analisado através de um DW. Através desta camada é possível o mapeamento para modelos conceituais e paralelamente manter informações sobre definições do negócio na visão de diversos setores da corporação, uma das lacunas deixadas pelo CWM e OIM. Além disso, esta solução proposta por nós, pode capturar o significado comum, a nível de negócio, entre diferentes modelos conceituais. Esta camada implementada neste trabalho é colocada como necessária em [Jorge, 2001] e [Nascimento, 2001] e faz parte de suas indicações de trabalhos futuros.

Seguindo a hierarquia da arquitetura de quatro níveis, podemos considerar "Dados do usuário" como sendo uma instância de "Esquemas", "Esquemas" como sendo uma instância de "Meta-esquemas", e, finalmente, "Meta-esquemas" como sendo uma instância de "Metameta-esquema".

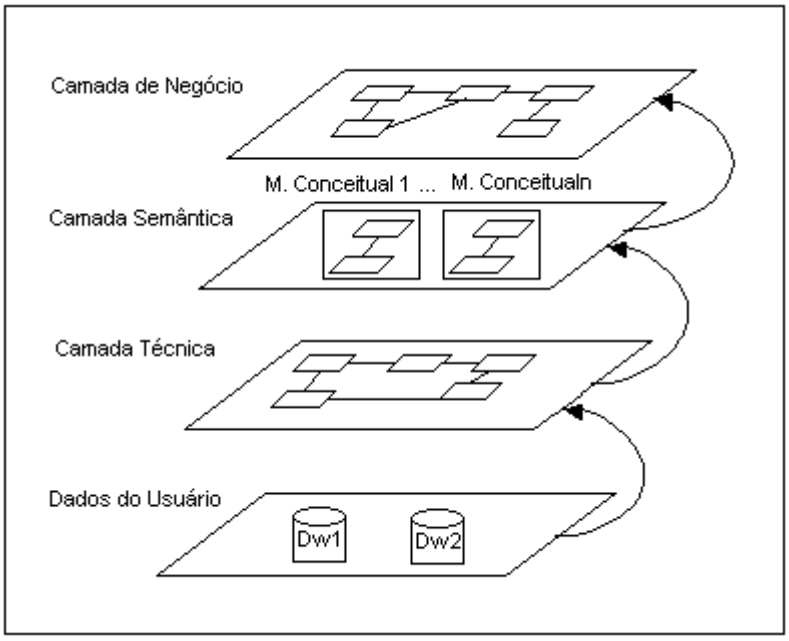

Figura 5.2 Arquitetura Multicamadas do Metha Manager

5.2 Modelo Conceitual

Nesta seção apresentaremos o modelo conceitual do Metha Manager nas suas diversas camadas. Os diagramas serão acompanhados de um glossário para o entendimento da

semântica dos atributos e classes apresentados. Cada camada será apresentada em separado, a solução de mapeamento entre as diversas camadas do modelo será apresentada no capítulo 6.

5.2.1 Camada de Negócio

Na figura 5.3 podemos observar o diagrama de classes(fase de análise) da camada de negócio. Abaixo temos o glossário para as principais classes e atributos desta camada:

N.1 Aditivos

Indicadores que podem ser adicionados ao longo de quaisquer perpectivas do negócio. O objetivo é poder prover a capacidade de informar se um indicador pode ser totalizado em uma consulta.

N.2 Características

Itens que descrevem perspectivas.

N.3 Consulta_N.

Consulta realizada por um ou mais usuários de negócio. A experiência mostra [Banese, 2000] que os usuários de um DW sentem a necessidade de armazenar uma consulta considerada estrategicamente importante, bem como os fatos que influenciaram a mesma.

N.4 Definição_Itens

Definição de item de negócio para um determinado setor. Como foi abordado na seção de objetivos, faz-se necessário um controle dos responsáveis pela inclusão de valores de atributo. Em diversos casos a interpretação dos dados é feita de maneira diferente em cada grupo, pois cada departamento tem suas próprias regras e conceitos [Banese, 2000] [Marco, 2000].

N.5 Definição_Perpectiva

Definição de perspectiva para um determinado setor.

N.6 Descrição_Função

Descrição da função do usuário

N.7 DI, DF e *Status*

Atributos temporais necessários para o controle de versionamento dos metadados. São respectivamente: Data de início de validade do metadado, data de fim de validade do metadado e situação do metadado (válido ou inválido atualmente). Desta forma é possível manter um histórico da evolução da semântica dos dados, um requisito indispensável aos analistas de negócio.

N.8 Escopos de Análise

Diversas abrangências de uma análise gerencial.

N.9 Fatos

Fatos ou causas que influenciaram uma consulta.

N.10 Gestor

Usuário de negócio.

N.11 Indicadores

Itens que medem desempenho do negócio.

N.12 Itens_do_Negócio

Diversos itens que caracterizam um negócio.

N.13 Não_Aditivos

Indicadores que não podem ser adicionados ao longo das perpectivas do negócio.

N.14 Negócio

Conjunto de regras e ações que caracterizam um dado setor em conjunto com seus usuários.

N.15 Nível_Caract

Determina o nível da característica em um escopo de análise. Isto é necessário, pois uma característica pode assumir diferentes níveis ao longo de escopos de análise diferentes.

N.16 Nome_Popular

Nome da perpectiva ou item como são chamados pelos usuários.

N.17 Perpectivas_do_Negócio

Diversas visões para análise de um negócio.

N.18 Semi_Aditivos

Indicadores que só podem ser adicionados ao longo de algumas perpectivas do negócio.

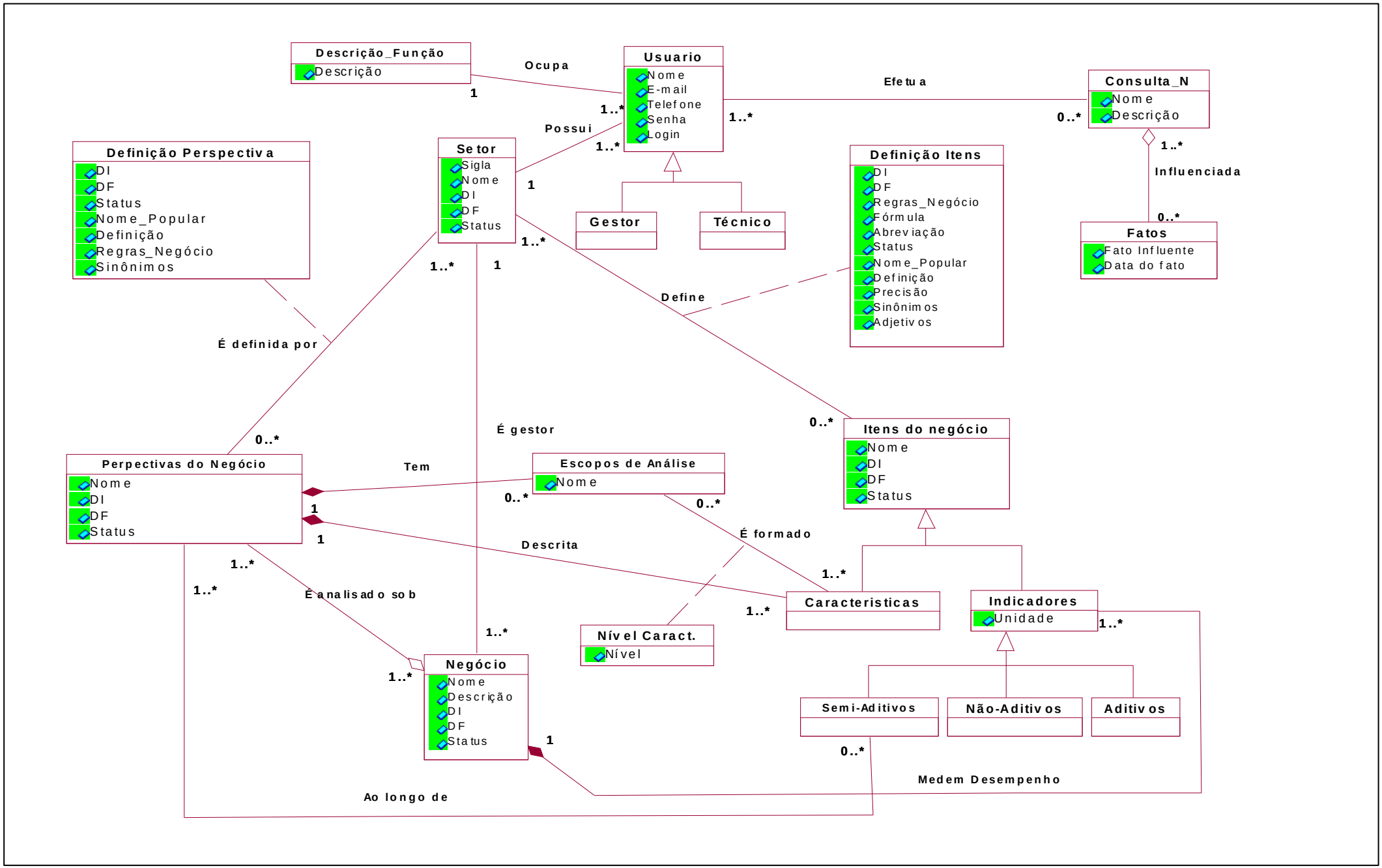

Figura 5.3 Modelo Conceitual – Camada de Negócio

5.2.2 Camada Semântica

Na figura 5.4 apresentamos as classes concretas, bem como as abstratas(itálico) que compõem o *framework* Metha Manager na sua camada semântica. As classes abstratas serão importantes para o mapeamento da camada de negócio para a semântica e para a possibilidade de extensão do modelo no intuito de acoplar diversos modelos conceituais.

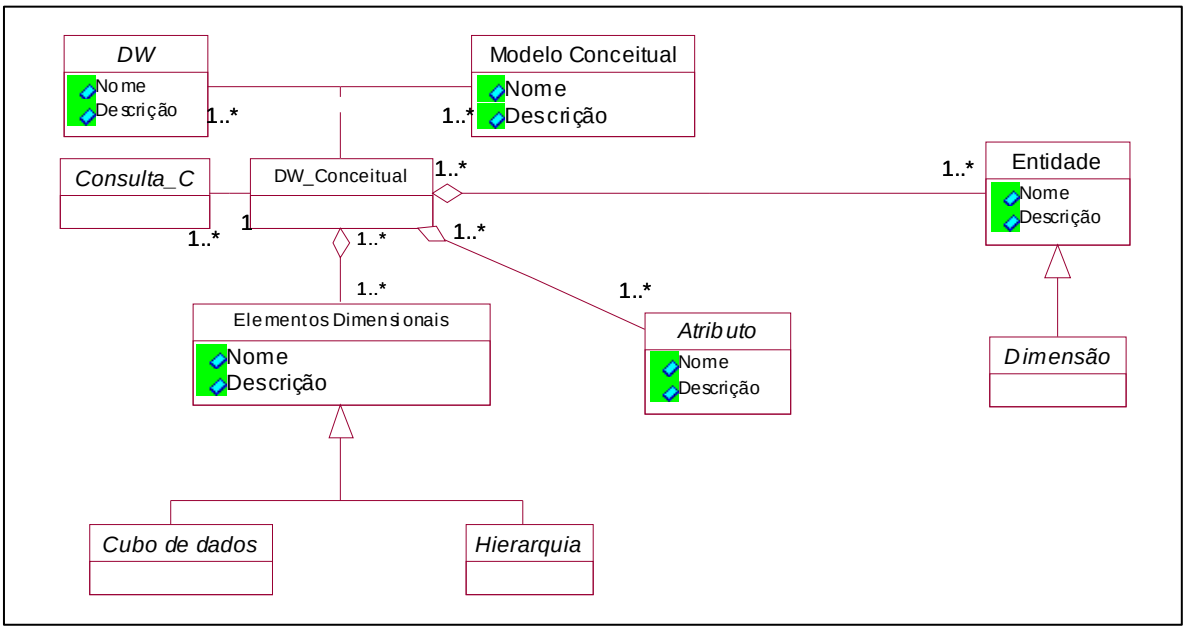

Figura 5.4 Modelo Conceitual – Camada Semântica

Para esta camada temos as classes abstratas *DW*, *Dimensão*, *Atributo*, *Cubo de dados*, *Hierarquia e Consulta_C* que poderão ser estendidas para acoplar diversos modelos conceituais. Estas classes foram selecionadas depois de um vasto estudo feito em diversos modelos conceituais utilizados por ferramentas de um processo de *Data Warehousing*. Todos os modelos analisados possuem uma classe hierarquia, por exemplo. A classe *Consulta_C* é útil para o mapeamento de uma consulta armazenada na camada de negócio e possível navegação até a camada técnica que possuirá metadados técnicos relativos à mesma considerada importante e influente por um usuário de negócio.

A seguir temos um glossário com as principais classes:

S.1 Elementos Dimensionais

Esta classe descreve as entidades específicas de um modelo conceitual multidimensional que dependem de outras entidades para existir. Um bom exemplo é a especialização *Hierarquia*, pois a mesma não existe sem a presença de uma Dimensão.

S.2 Cubo de Dados

Descreverá os diversos cubos de um modelo multidimensional.

S.3 Hierarquia

Representa as hierarquias que podem estar presentes em uma dimensão.

S.4 Atributo

Descreve os diversos atributos de um modelo conceitual, sejam eles descritores de uma dimensão ou medidas de um cubo de dados.

S.5 Entidade

Descreve as entidades independentes de um modelo conceitual multidimensional, ou seja, entidades mestres que podem descrever ou agregar outras entidades. A especialização *Dimensão* é um bom exemplo, pois instâncias de classes de diversos modelos analisados tais como *Cubo de Dados e Hierarquia* dependem da mesma para existirem.

S.6 Dimensão

Descreve dimensões de um modelo multidimensional.

5.2.3 Camada Técnica

A camada técnica abrange metadados lógicos, operacionais e físicos, ou como o próprio nome sugere, todos os metadados importantes para a equipe técnica do projeto de DW. Não devemos confundir camada com esquema, pois a nossa camada técnica abrange os esquemas lógico, físico e operacional, assim como a camada semântica abrange os esquemas conceituais acoplados ao Metha Manager. A figura 5.5 ilustra esta camada e a seguir também temos um glossário com as principais classes e atributos:

T.1 Atributo_L

Informações dos atributos lógicos do DW de acordo com a tecnologia de armazenamento utilizada.

T.2 Atributo_Origem

Atributos dos diversos sistemas de origem que abastecem um DW.

T.3 Cód_Retorno e Msg_Retorno

Código e mensagem de retorno que indicam sucesso ou falha no processo ETL.

T.4 Consulta_L

Esta classe será usada para armazenar estatísticas de processamento das consultas do ambiente DW. Ela também faz parte do mapeamento dos metadados de uma consulta a nível de negócio até o nível técnico.

T.5 Descrição_Tecnologia

Descrição da tecnologia de armazenamento (Relacional, Objeto-Relacional, etc.).

T.6 Domínio_Destino e Domínio_Origem

São usadas para listar os valores permitidos para cada atributo, bem como a descrição sobre esse valor.

T.7 Estatísticas_Extração

A classe Estatísticas_Extração é usada para armazenar estatísticas de processamento de um processo ETL.

T.8 Flag_Integração

Flag que indica se mais de um objeto de origem alimentam o *Objeto_L* destino.

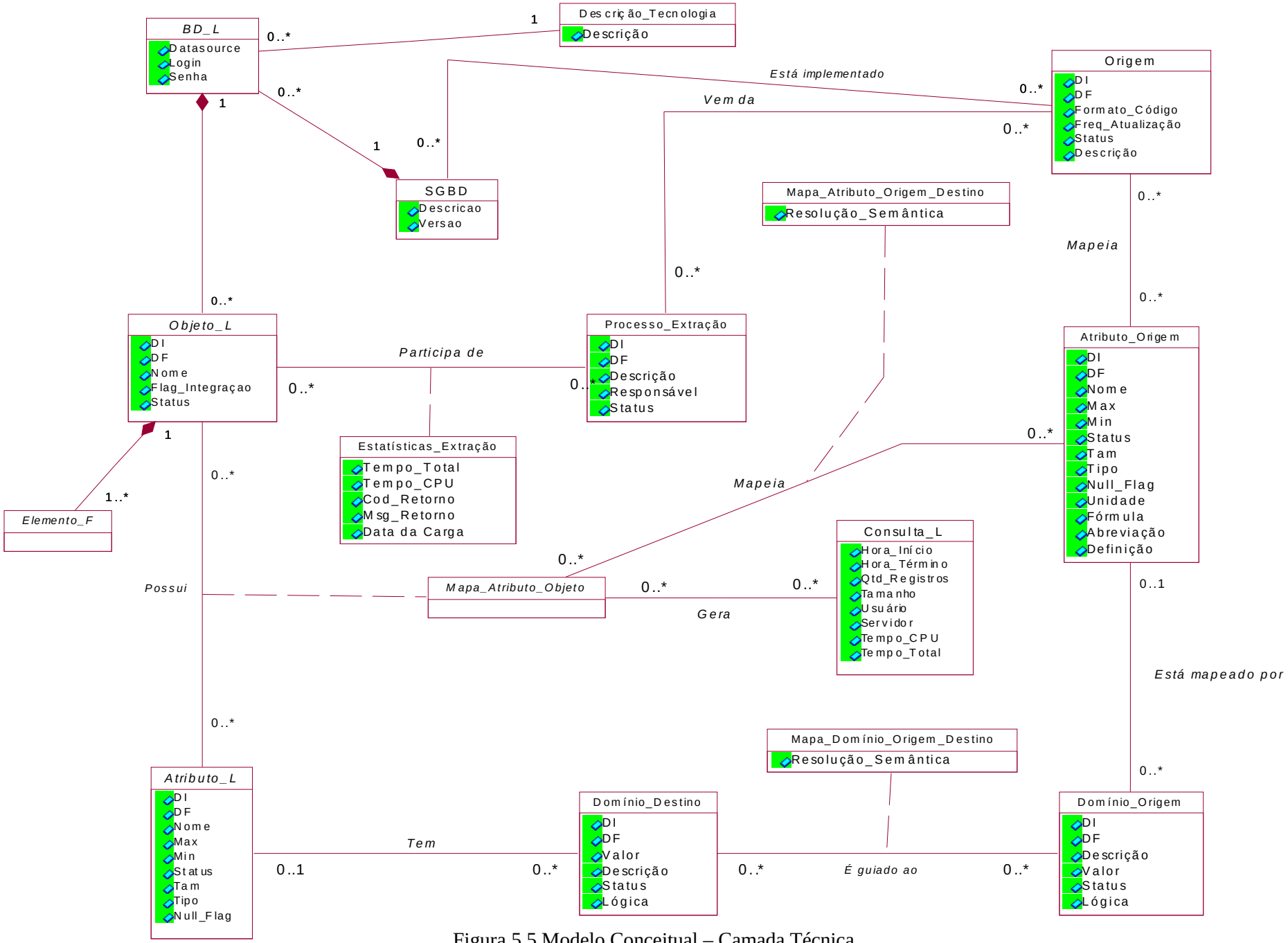

Figura 5.5 Modelo Conceitual – Camada Técnica

T.9 Formato_Código

Formato do código-fonte do sistema origem.

T.10 Fórmula

Cálculo ou fórmula usada para criar o valor do atributo.

T.11 Freq_Atualização

Freqüência em que os dados são atualizados no sistema origem.

T.12 Lógica

Lógica utilizada para definição do domínio.

T.13 Max e Min

Valor máximo e mínimo permitidos para o atributo.

T.14 Null_Flag

Flag que indica se é permitido valores nulos para o atributo.

T.15 Objeto_L

Esta classe armazena informações sobre os objetos lógicos do ambiente DW. Se o DW está armazenado em um banco de dados relacional esta classe armazenará informações sobre uma tabela, por exemplo.

T.16 Origem

A classe *Origem* descreve os sistemas usados para alimentar o ambiente DW.

T.17 Processo_Extraçào

Esta classe descreve os procedimentos de extração, transformação e carga dos dados que provém dos sistemas origem até serem inseridos no DW.

T.18 Responsável

Pessoa ou departamento responsável pelo processo de extração.

T.19 Tam

Tamanho máximo permitido para o atributo.

T.20 Tipo

Tipo do atributo (caracter, inteiro, etc.).

T.21 Unidade

Unidade de medida do atributo.

As classes abstratas(itálico) desta camada também servirão para mapeamento, bem como para possibilitar extensão de acordo com a tecnologia de armazenamento utilizada pelo DW. Analisaremos mapeamento e extensão no próximo capítulo.

5.3 Requisitos Funcionais e Não-Funcionais

Podemos identificar 5 principais atores com relação ao sistema: a) o usuário de negócio, b) o usuário técnico, c) as ferramentas de um processo de *Data Warehousing*, d) o SGBD, e) programador (implementador). O último não interage diretamente com o sistema, mas é responsável por estender a funcionalidade da aplicação. Cada um desses atores possui cenários particulares de interação que serão detalhados a seguir.

A.1 O Usuário de Negócio

Usuários que interagem com ferramentas de DW para efetuar consultas OLAP através de esquemas conceituais (Tomadores de Decisão). Estes usuários também podem interagir diretamente com o Metha Manager para efetuar consultas sobre metadados de negócio.

U.1 Consulta Metadados de Negócio

A principal atividade de interação realizada pelo Usuário de Negócio é a exploração do esquema de negócio de sua aplicação. Com o auxílio de algum mecanismo, seja ele gráfico ou textual, ao usuário devem ser apresentados os principais conceitos relativos ao negócio a

ser analisado através de uma ferramenta OLAP, por exemplo. Dessa forma, o mesmo estará apto a utilizar as diversas variáveis como forma de seleção ou restrição em suas consultas. Vale ressaltar que a consulta poderá ser feita com dados atuais e históricos,

U.2 Insere Metadados de Negócio

O Usuário de Negócio tambem pode com auxílio de algum mecanismo inserir informações sobre os diversos itens e perpectivas de negócio existentes para análise. É oportuno enfatizar que o Metha Manager foi projetado para fazer distinção entre interpretações diferentes de diversos setores de uma organização, bem como manter histórico dos metadados.

U.3 Mantem Versionamento de Metadados

Em qualquer inserção de metadados, se for o caso, as versões antigas serão mantidas junto com suas rescpectivas datas (atributos temporais). Ver Capítulo 3, seção 3.2.9.

U.4 Mantem Propriedade da Informação

Na inserção de metadados de negócio, o gerente terá que identificar o setor que está inserindo a informação. Desta forma é possível manter a interpretação de diversos setores.

A.2 Usuário Técnico

O Usuário Técnico será qualquer integrante da equipe técnica do projeto de DW excetuando o implementador, entre eles:

- Projetistas de DW - Usuários que fazem uso de modelos conceituais para definir esquemas conceituais de DW.
- Administradores de Banco de Dados - Usuários responsáveis pela administração dos servidores de banco de dados (OR-Oracle, R-Oracle, R-SQLServer, MOLAP-Essbase, etc), incluindo o mapeamento entre esquemas lógicos e físicos.

U.5 Insere Metadados Conceituais

O Usuário Técnico pode criar um esquema multidimensional de um DW, com elementos: Cubo, Dimensão, Hierarquia de Agregação, Célula do Cubo e Atributo de Dimensão.

U.6 Insere Metadados Técnicos

Como o esquema conceitual em questão deve ser vinculado a um esquema lógico o Usuário Técnico também pode inserir metadados lógicos e/ou físicos e operacionais.

U.7 Consulta Metadados Conceituais

Permite ao Usuário Técnico consultar o esquema conceitual definido. Exemplo: “Quais são as dimensões de um certo cubo ?“.

U.8 Consulta Metadados Técnicos

Permite consultar o esquema lógico que foi acoplado e vinculado com um esquema conceitual. Exemplo: “Quais tabelas de um banco de dados relacional simulam um cubo? “

Este caso de uso utiliza o caso de uso [U7] Consulta Metadados Conceituais, para obter informações do esquema lógico. Todas as informações do esquema lógico estão relacionadas com o esquema conceitual. Também é possível consultar metadados operacionais, bem como consultar informações internas de como os objetos multidimensionais estão fisicamente no Gerenciador de Dados. Para a consulta sobre informações físicas o usuário navega desde o esquema conceitual, passando pelo esquema lógico, até chegar no esquema físico. Exemplo: *“Seleciona um Cubo : Vendas no esquema Conceitual. O Cubo Vendas no esquema lógico referencia uma tabela Vendas_Diarias; Por fim, no esquema Interno o usuário obterá informações sobre: Índices, Tablespace, Espaço que esta ocupando, etc;”.*

A.3 Programador (Implementador DW)

São os usuários que fazem o mapeamento de esquemas conceituais de DW para esquemas lógicos de DW. Os esquemas lógicos podem ser de diferentes tecnologias (OR-Oracle, R-Oracle, R-SQLServer, MOLAP-Essbase, etc). Este usuário também faz o mapeamento de um modelo de negócio para esquemas conceituais acoplados ao Metha Manager.

U.9 Acopla Esquema Conceitual

O Implementador pode acoplar modelos conceituais ao Metha Manager.

U.10 Acopla Esquema Lógico

O Implementador pode acoplar um determinado esquema lógico DW para implementar um determinado esquema conceitual. Por exemplo, um esquema em estrela relacional, sintaxe Microsoft SQLServer, implementando um esquema conceitual multidimensional.

U.11 "Customiza" *Framework*

Permite ao Implementador "customizar" o Metha Manager para prover metadados para uma nova tecnologia de implementação de DW. A "customização" consiste na extensão do Metha Manager, criando um repositório apto a prover metadados para uma determinada tecnologia de armazenamento (ROLAP, MOLAP, etc). Também é possível extender o Metha Manager para que sejão acoplados novos modelos conceituais.

A.4 SGBD

Interage com Metha Manager ao fornecer metadados técnicos pré-armazenados no SGBD utilizado em questão.

U.12 Utiliza Informações do Dicionário de Dados

O objetivo é permitir o aproveitamento de metadados técnicos armazenados no dicionário de dados do SGBD.

A.5 Ferramentas de DW

A consulta de metadados através do Metha Manager também pode ser feita por uma Ferramenta de DW, como o Servidor Edwards [Jorge, 2001], através da API (*Application Programming Interface*) fornecida pelo Metha Manager.

U.13 Gera XML

Uma ferramenta de DW pode carregar metadados também através de XML [Jorge, 2001]. Este caso de uso permite a exportação de metadados coneituais e lógicos através de XML.

Nas figuras 5.6, 5.7 e 5.8, as interações dos atores são descritas por meio de três *diagramas de casos de uso*. Nos cenários das figuras são apresentados os diagramas de casos de uso do Metha Manager, em que o Metha Manager é estendido para para prover metadados de um DW implementado segundo duas tecnologias, R-SQLServer e OR-Oracle e segundo dois modelos conceituais diferentes [Jorge, 2001] [Nascimento, 2001].

Por questões de visualização, dividimos o diagrama de casos de uso em três.

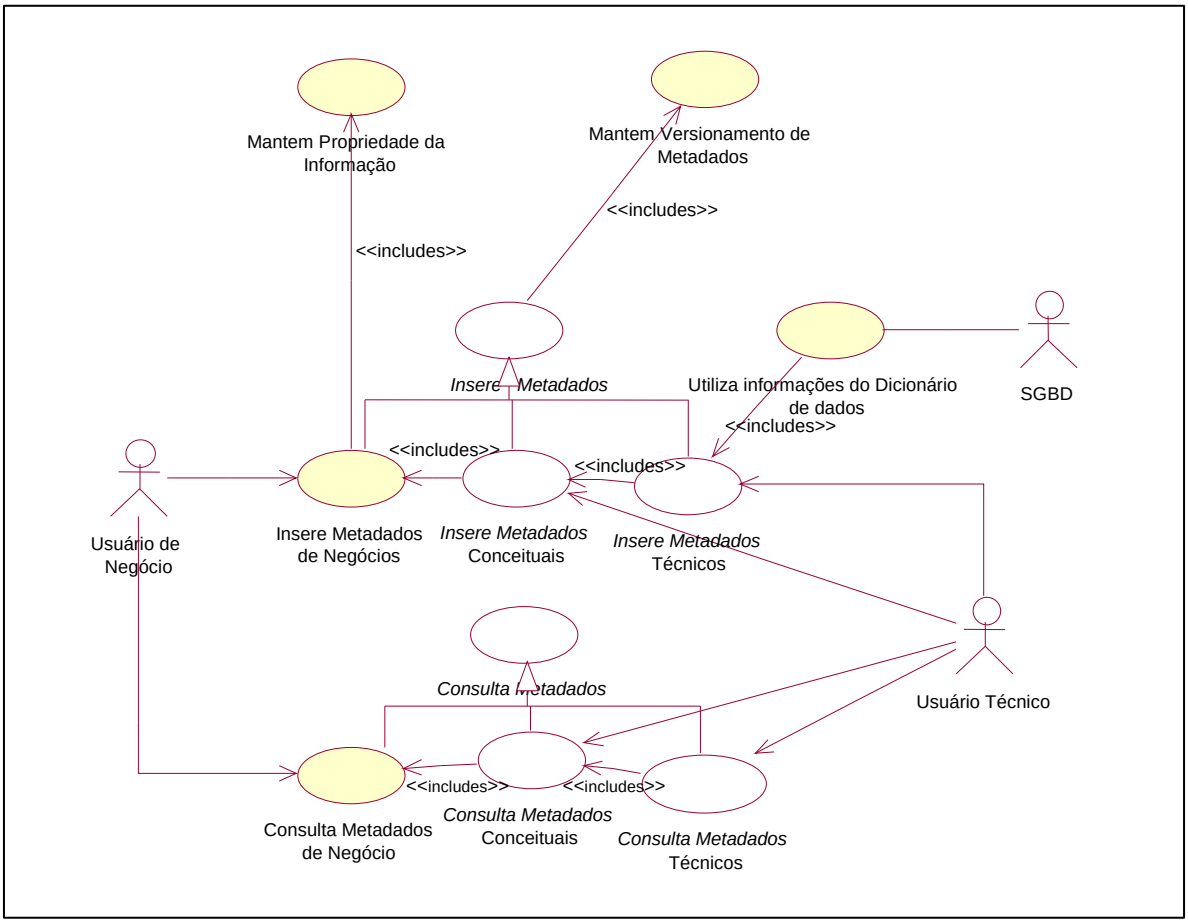

Figura 5.6 Diagrama de Caso de Uso para o cenário de consulta e inserção de metadados

O primeiro diagrama (Figura 5.6) descreve o cenário para as consultas e inserções realizadas pelos usuários técnicos e de negócio, bem como a interação do Metha Manager com o SGBD.

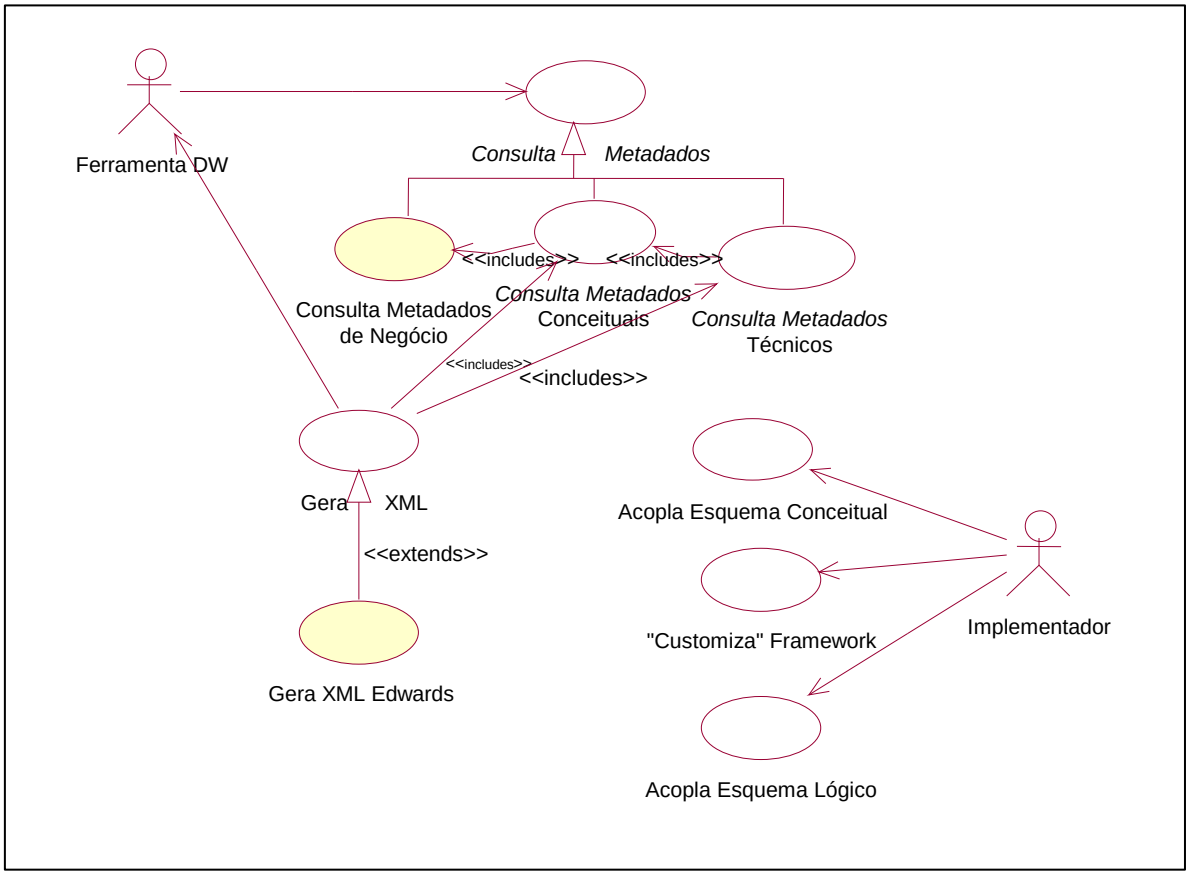

Figura 5.7 Diagrama de Caso de Uso para o cenário de “customização”, geração de XML e consulta de metadados por uma ferramenta de DW.

O segundo diagrama (Figura 5.7) descreve o cenário para os usuários que estão interessados em “customizar” o gerente, ou seja, habilitá-lo a prover metadados de uma determinada tecnologia de armazenamento, bem como utilizar diferentes modelos conceituais. Este cenário descreve ainda a interação de Ferramentas DW com a funcionalidade de consulta de metados e geração de XML.

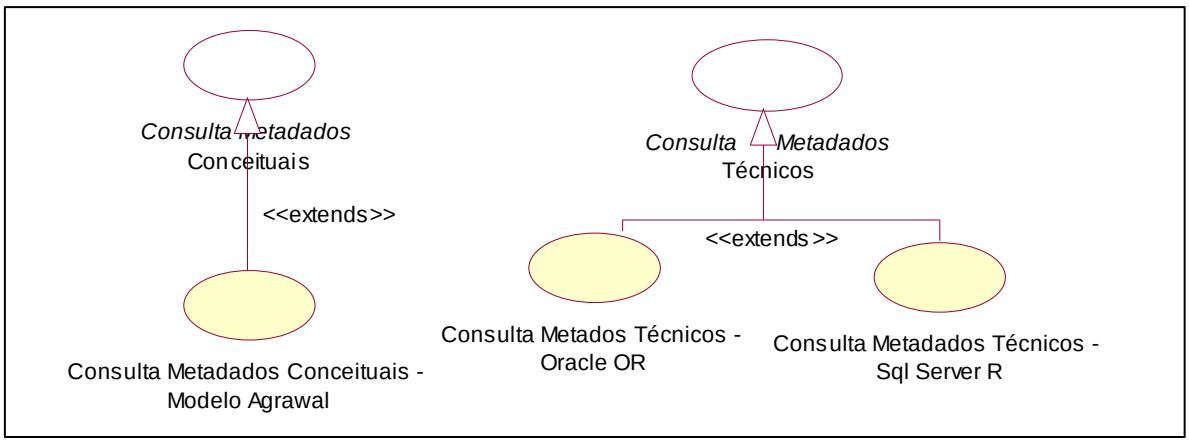

Figura 5.8 Diagrama de Caso de Uso para o cenário de extensão do Metha Manager.

O terceiro diagrama (Figura 5.8) ilustra extensões que podem ser feitas no Metha Manager.

No diagrama de caso de uso ilustrado nas figuras 5.6, 5.7 e 5.8 as funcionalidades do Metha Manager são representadas por elipses brancas. Os casos de usos de aplicações que estendem o Metha Manager são representados por elipses de cor amarela.

Os casos de uso brancos, pertencentes ao gerente, podem ser concretos ou abstratos. Os casos de uso abstratos para cumprirem suas funcionalidades necessitam ser estendidos (“extends”) por outros casos de uso.

5.3.1 Requisitos Não-Funcionais

NF.1 Tratamento de níveis de segurança para acesso.

Com o objetivo de garantir a funcionalidade *Manter Propriedade da Informação* (U.4), o Metha Manager possui um esquema de segurança que não só restringe o acesso como identifica os responsáveis pela inserção de metadados. Ver Capítulo 3, seção 3.2.12.

NF.2 Atendimento das requisições concorrentes dos usuários de forma eficiente.

Para atender as requisições concorrentes dos usuários de forma eficiente, o Metha Manager possui uma arquitetura escalável. Ela é composta de três camadas (Apresentação, Aplicação e Dados), com cada camada podendo estar distribuída em máquinas distintas. Na camada de dados, o Metha Manager faz uso das otimizações específicas do Servidor de Banco de Dados, obtendo assim, o máximo de desempenho do Servidor.

NF.3 Transportabilidade entre ambientes operacionais.

O Metha Manager é facilmente transportável entre ambientes operacionais, pois foi todo desenvolvido em Java.

NF.4 API para que ferramentas de DW possam interagir com o Metha Manager.

Qualquer ferramenta de DW pode fazer consultas a metadados. Para isso, a ferramenta deve usar uma API (*Aplication Programming Interface*) fornecida pelo Metha Manager.

NF.5 Ambiente Web de consultas

O Metha Manager disponibiliza um ambiente onde os usuários podem fazer consultas a metadados. Para o usuário utilizar o ambiente, ele só necessita de uma máquina em rede que execute um "Browser" HTML.

NF.6 Independência de tecnologias de Servidores de Banco de Dados.

O Metha Manager está preparado para adaptar-se a qualquer tecnologia de servidores de banco de dados no que diz respeito a provisão de metadados. Desenvolvido com base na tecnologia de *framework* (ver o capítulo 4), o Metha Manager possibilita que ele seja estendido para que lhe seja acoplado metadados para qualquer Sistema de Gerência de Banco de Dados, com o mínimo de impacto.

NF.7 Reutilização da análise, do projeto e do código do Metha Manager no desenvolvimento de extensões do Metha Manager.

Outro benefício da tecnologia de *framework*, utilizada na construção do Metha Manager, é o de permitir ao desenvolvedor que vai implementar o código para "customizar" o gerente a utilizar novas tecnologias e novos modelos conceituais, reutilizar a análise, o projeto, a implementação e os testes do projeto do *Framework* Metha Manager.

5.5 Estudo de Caso

Nesta seção, ilustraremos um pequeno exemplo de uso do Metha Manager através de um estudo de caso, um DW Vendas, e da *interface* web desenvolvida para consulta a metadados. Todo processo será tratado em alto nível. Vale salientar que o Metha Manager também foi testado provendo metadados para o Servidor Edwards [Jorge, 2001] e para uma Interface para Consultas a DW's [Nascimento, 2001].

A interface Web (NF.5) foi construida de maneira básica para, inicialmente, apenas validar a API (NF.4) do Metha Manager, que foi, posteriormente, utilizada pelo Servidor Edwads, por exemplo. Na figura 5.9 podemos observar um dos requisitos do gerente (NF.1), ou seja, tratamento de níveis de segurança para acesso.

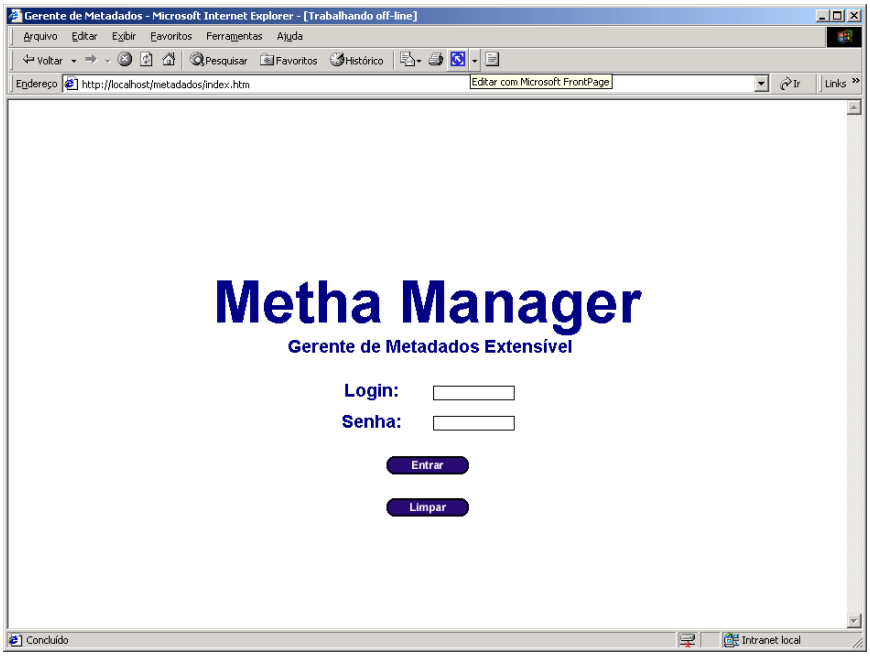

Figura 5.9 Interface Web do Metha Manager

Ao acessar o gerente, o mesmo identifica qual o tipo de usuário e, consequentemente, aplica as restrições definidas. Um usuário de negócio (Gestor), por exemplo, pode navegar nos diversos metadados de negócio existentes (Figuras 5.10 e 5.11). Também é possível, através da API, consultar todos os metadados lógicos relacionados com um item, perpectiva ou característica do negócio a ser analisado (Ver Capítulo 3, seção 3.3.1) (Figura 5.12)

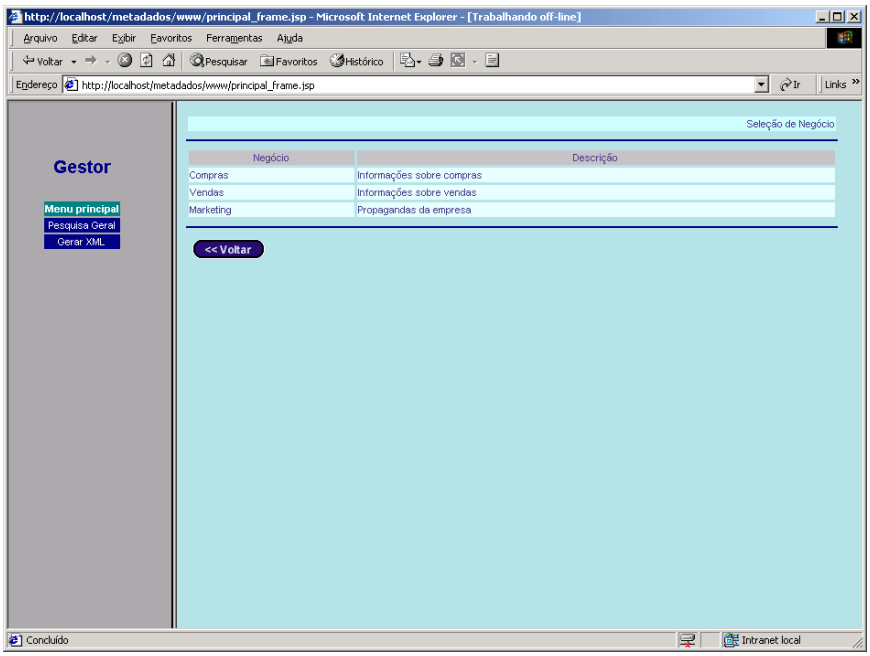

Figura 5.10 Interface Web para Navegação por Metadados de Negócio

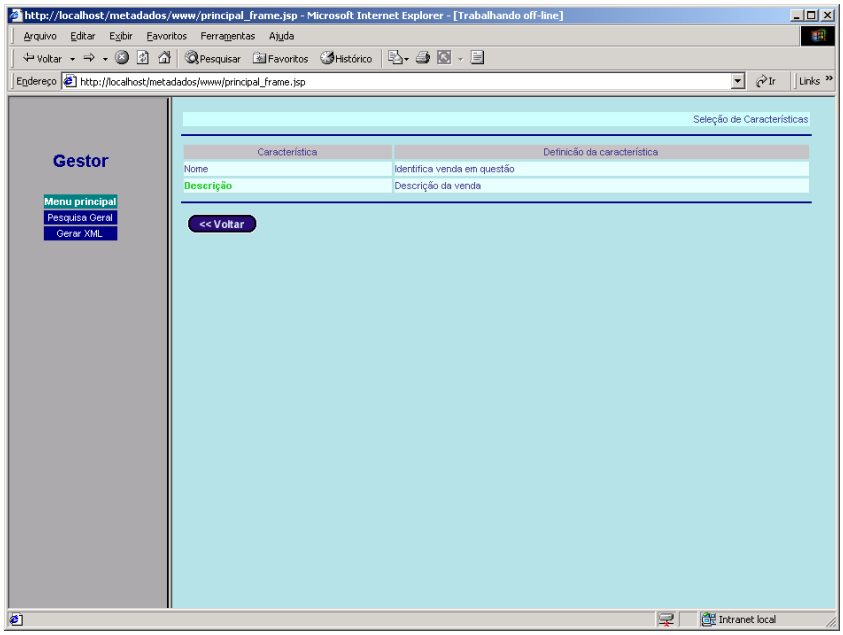

Figura 5.11 Interface Web para Navegação por Metadados de Negócio

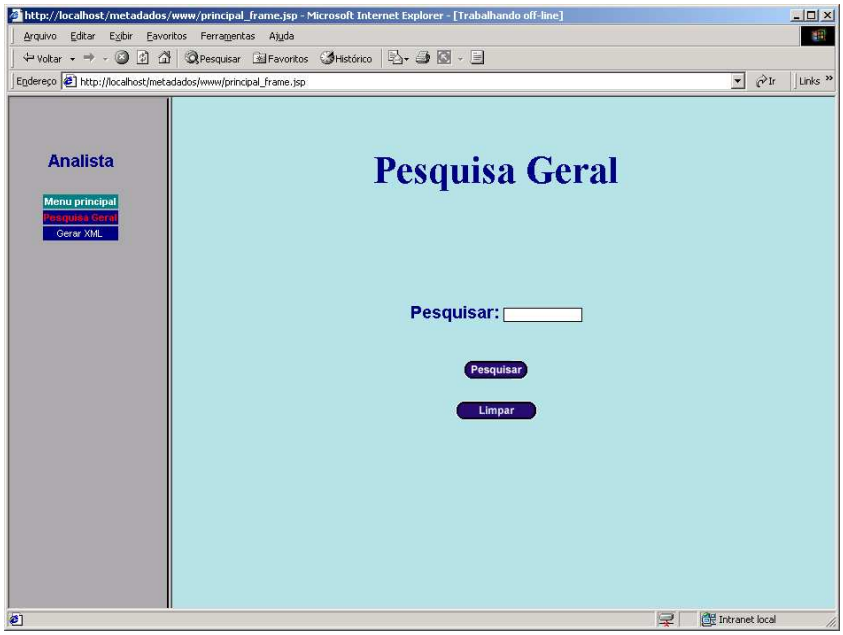

Figura 5.12 Interface Web para “Provisão de Informações”

Também foi criada uma interface para validar a geração de XML pelo Metha Manager para o Servidor Edwards (Figura 5.13). É possível, a partir da escolha de um DW, gerar o XML com o esquema conceitual e lógico do mesmo. Um exemplo do XML gerado encontra-se nos anexos.

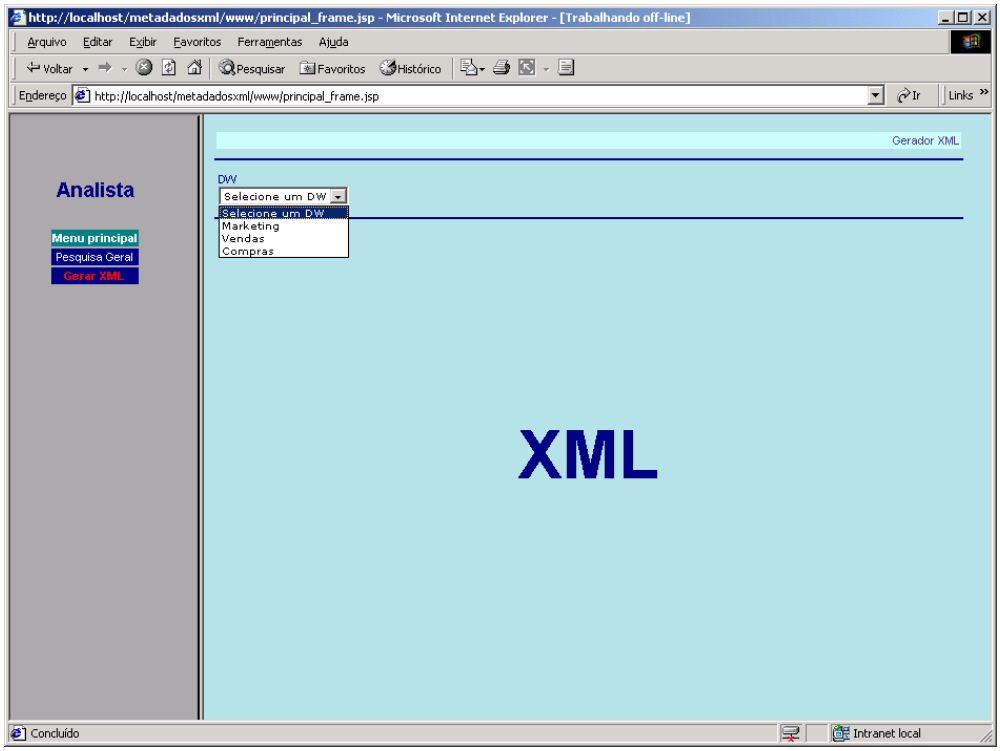

Figura 5.13 Geração de XML pelo Metha Manager

Capítulo 6

Projeto e Implementação do *Framework* Metha Manager

Neste capítulo apresentaremos o projeto arquitetural com o intuito de demonstrar as decisões tomadas em alto nível tais como: modularização, estrutura de comunicação e controle, estratégia de persistência e paradigmas de banco de dados usados. Seguimos pormenorizando o projeto de baixo nível. Por fim, é apresentado um roteiro com os passos para a construção de novas extensões (novos modelos conceituais e novas tecnologias de gerencias de dados acopladas no Metha Manager).

6.1 Projeto Arquitetural

A construção do projeto arquitetural foi oportuna para a elaboração de uma estrutura elegante usando camadas e partições. É importante frisar que o objetivo principal foi manter os subsistemas coesos e com fraco acoplamento.

A arquitetura do Metha Manager — figura 6.1 — é implementada em três camadas (Interface, Aplicação e Dados) para atender aos requisitos não-funcionais enumerados e à arquitetura genérica definida no Capítulo 5 seção 5.1.2.

A *camada de interface* é a camada que o usuário final usa para acessar e manipular o Metha Manager. Ela foi projetada para ser totalmente desacoplada da camada de aplicação, como rege o requisito *"NF.2 Atendimento das requisições concorrentes dos usuários de forma eficiente."*. Trata-se do conjunto de todas as aplicações (Web, Windows, etc) que interagem com a camada de aplicação remotamente (máquinas distintas), através de métodos de acesso remotos ─ chamados através de RMI ("Remote Method Invocation") ─ disponíveis na API.

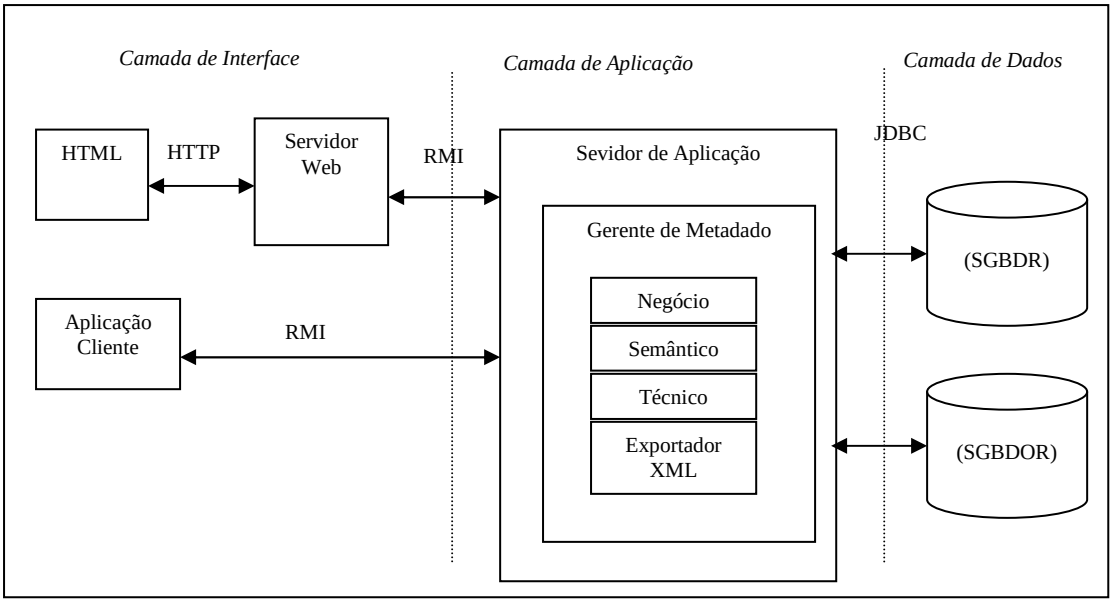

Figura 6.1: Projeto Arquitetural do Metha Manager.

Já a *camada de aplicação* contém todos os objetos com as regras de negócio e as operações para o funcionamento do Metha Manager. É nela que é feita a gerência do repositório de metadados. Devido à complexidade desta camada, esta foi fragmentada em quatro módulos, *Negócio, Semântico, Técnico e Exportador XML*. Segue-se a descrição dos módulos.

M.1 Negócio.

Módulo que mantém as estruturas relativas aos metadados no nível de negócio.

M.2 Semântico

Módulo que mantém as estruturas relativas aos metadados no nível conceitual. Ex: DWs, Cubos, Dimensões, Elementos de Cubo e Elementos de Dimensão.

M.3 Técnico.

Módulo que mantém as estruturas relativas aos metadados nos níveis lógico (Conexões com SGBDs, Esquemas de BD) e físico (Índices, “Tablespace”, Estatísticas sobre tabelas, etc).

M.4 Exportador XML.

Módulo que gera arquivo XML com especificação do modelo conceitual e lógico de um DW.

As dependências entre os níveis são tratadas distribuidamente nos módulos. A título de ilustração, uma dimensão (nível conceitual) corresponde a uma determinada tabela de um banco de dados (nível lógico) com índices que a indexam (nível físico).

Por fim, na *camada de dados* estão os SGBDs onde foi implementada a persistência do Metha Manager. No nosso trabalho, com o objetivo de avaliar o paradigma objeto-relacional, implementamos duas bases de dados: uma relacional e outra objeto-relacional. A comunicação da camada de aplicação a esta se dá através de chamadas JDBC ("Java Database Connectiviy"). Assim, a *camada de dados* pode estar distribuída em um sistema computacional diferente da *camada de aplicação*, algo desejável por questões de escalabilidade e desempenho.

6.1.1 Projeto Arquitetural Detalhado

Nesta seção apresentaremos o projeto arquitetural com um nível de detalhe maior. Estão dispostos, além dos módulos já detalhados anteriormente, os pontos de extensão do *framework* Metha Manager.

A ilustração da figura 6.2 apresenta a relação existente entre os módulos do Metha Manager, bem como os *Pacotes* para extensão do *framework.* Estes *Pacotes* estão representados por retângulos pontilhados. Os retângulos verdes são os blocos para encaixe ("plugs") das extensões. Podemos dividir o *framework* do Metha Manager em três *frameworks* menores: Semântico, Técnico e Exportador XML.

Os *Pacotes* que estendem o *framework* são pacotes com componentes de software (classes) com o código específico para a aplicação que se deseja executar no Metha Manager. Nos *Pacotes* estão um conjunto de classes que derivam classes e implementam funcionalidades de classes predefinidas do *framework* Metha Manager, estendendo assim, suas funcionalidades. Este processo visa acoplar diversos modelos conceituais, bem como metadados lógicos e físicos de uma determinada tecnologia de armazenamento.

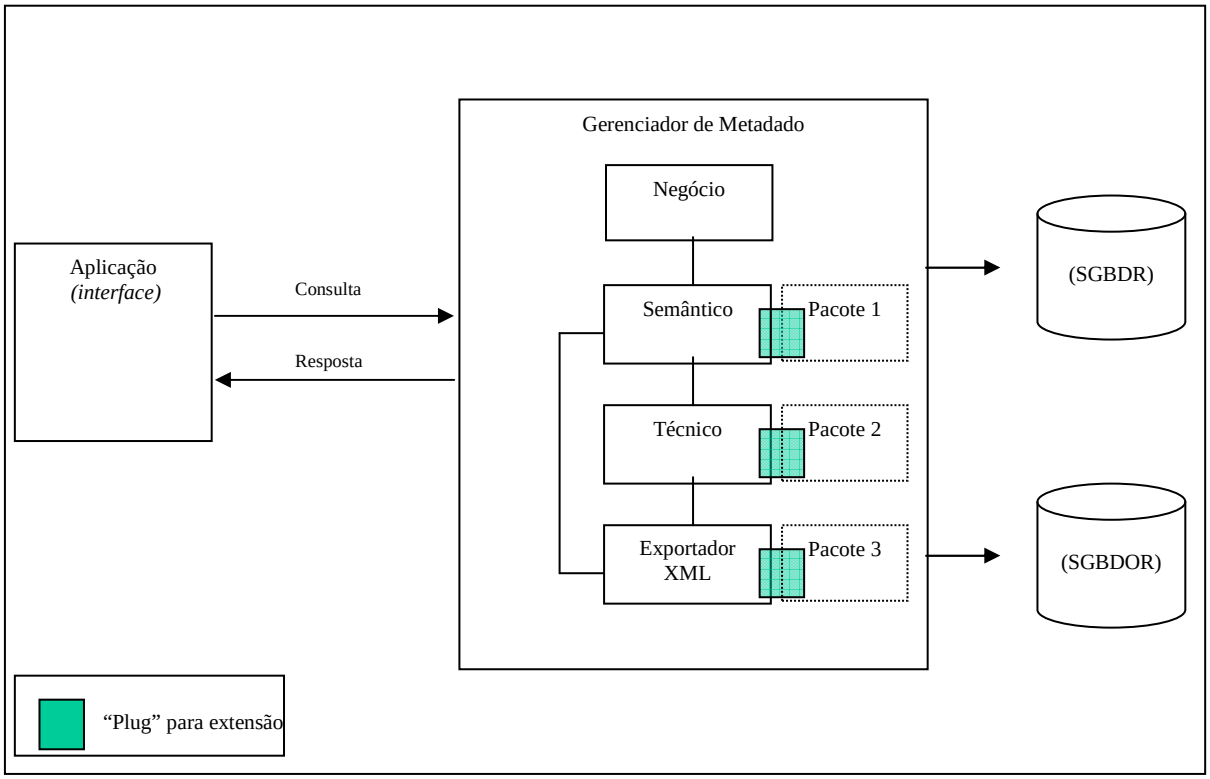

Figura 6.2: Projeto Arquitetural detalhado do Metha Manager.

Listamos os *Pacotes* disponíveis na versão atual para extensão do *framework*:

Extensão Relacional

Pacote 2 – Metadado Lógico e Físico para Aplicações Relacionais.

Extensão OR-Oracle

Pacote 2 – Metadado Lógico e Físico para Aplicações Objeto-relacionais Oracle.

Extensão Modelo Conceitual Agrawal [Agrawal, 1999]

Pacote 1 – Metadado Conceitual para Servidor Edwards [Jorge, 2001].

Extensão Modelo Conceitual Interface

Pacote 1 – Metadado Conceitual para Interface para Consultas Baseada em Linguagem Natural [Nascimento, 2001].

Extensão XML Edwards

Pacote 3 – Geração de XML para o Servidor Edwards.

Na próxima seção, descrevemos em detalhes o projeto do Metha Manager, particularmente o mapeamento entre as camadas do modelo.

6.2 Projeto de Baixo Nível

Por questões didáticas, a apresentação do projeto de baixo nível do Metha Manager é apenas parcial.

O foco são os módulos Semântico e Técnico. Iniciamos com a apresentação dos diagramas de classes dos dois módulos. No caso do módulo Semântico apresentaremos os diagramas de classes dos dois modelos conceituais implementados (extensões) e a solução de mapeamento adotada entre a Camada de Negócio e a Camada Semântica. Por fim, mais uma extensão do Metha Manager é descrita, correspondendo à provisão de metadados para um DW implementado no SqlServer 7 e no Oracle OR 8i.

6.2.1 Modelos Conceituais Implementados

6.2.1.1 Modelo Conceitual de Agrawal

O diagrama de classes da figura 6.3 descreve o modelo de metadados conceitual de Agrawal utilizado em [Jorge, 2001]. Descrevemos os principais elementos do modelo.

A classe *DW_C* – esquema DW – é composta por uma coleção de dimensões da classe *Dimensao_C* e uma coleção de cubos da classe *Cubo_C*. Os objetos da classe *Dimensao* servem para a montagem dos cubos. Já um objeto do tipo *Cubo_C* é o *cubo de dados*, definido em [Agrawal, 1999].

A classe *Cubo_C* agrega uma coleção de elementos da classe *Elemento_Cubo_C* e uma coleção de dimensões compartilhadas da classe *Dimensao_Cubo_C*. Assim, define-se a estrutura de um *cubo de dados.*

A classe *Hierarquia_C* contempla a estrutura da hierarquia de dimensão para mudanças de “granularidade” dos *cubos de dados* de um esquema.

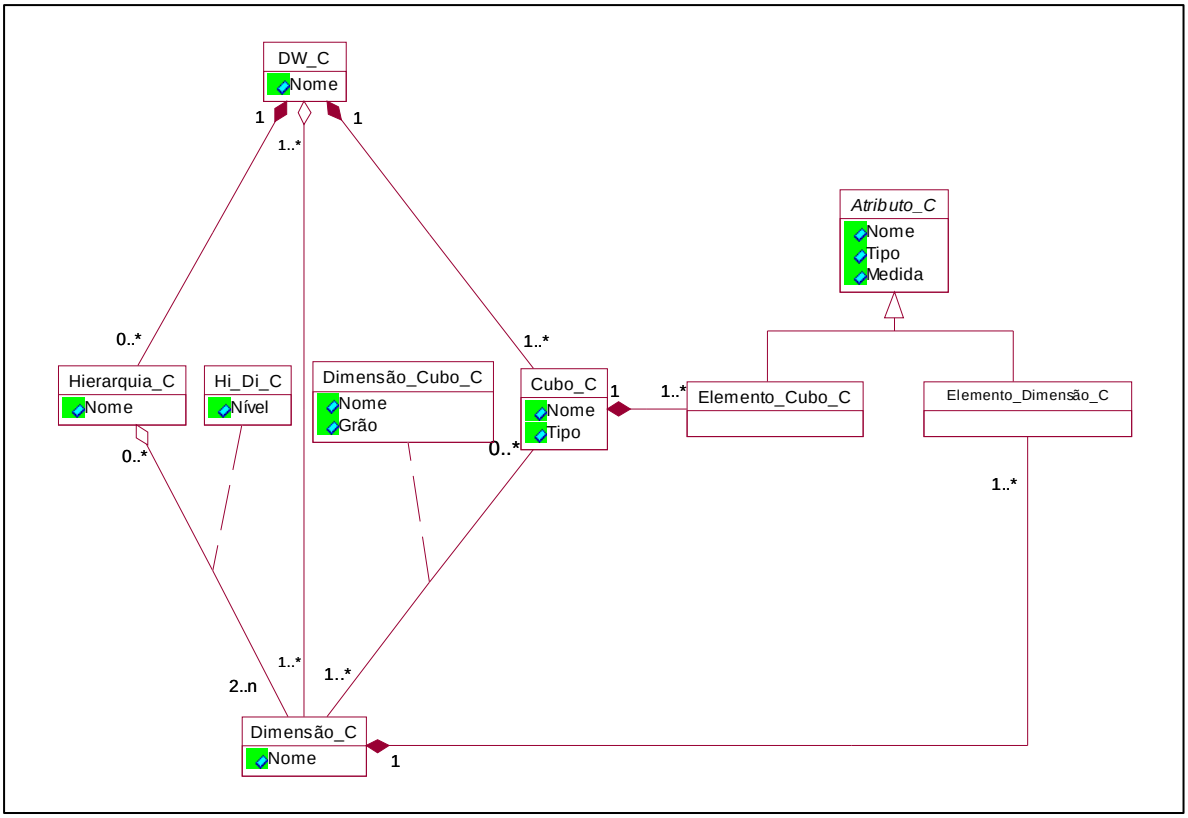

Figura 6.3: Diagrama de classes UML do modelo de metadados conceitual Agrawal.

6.2.1.2 Modelo Conceitual Utilizado Pela Interface Baseada em Linguagem Natural

O diagrama de classes da figura 6.4 descreve o modelo de metadados conceitual utilizado em [Nascimento, 2001]. Podemos fazer a mesma crítica descrita no Capítulo 5 seção 5.1.1 a este modelo, ou seja, o mesmo é totalmente voltado para o modelo relacional. Descrevemos os principais elementos do modelo.

O ponto de acesso para o conjunto de metadados semânticos é a classe *DataCube*, cujas instâncias são responsáveis pela manipulação direta e indireta dos outros objetos pertencentes ao modelo. Uma instância do tipo *DataCube* é descrita por dois principais elementos: *FactTable e Dimension*, descritores para uma tabela de fatos e tabelas de dimensões, respectivamente. Essas duas classes são especializações de uma classe superior, *SemanticEntity*. De modo semelhante, as classes *FactAtribute e DimensionObject*, são especializações da classe *SemanticAtribute*.

Instâncias da classe *FactTable* descrevem as tabelas de fatos em um esquema dimensional. Sua principal responsabilidade é manipular instâncias do tipo *FactAtribute* que descrevem as medidas encontradas em uma tabela de fatos.

Dimensões de um esquema dimensional são descritas por instâncias da classe *Dimension*, a qual representa uma agregação de *DimensionObjects*, descritores para os atributos de uma tabela de dimensão.

Instâncias da classe *DimHierarchy*, definidas por instâncias da classe *DimLevel*, representam as hierarquias que podem estar presentes em uma tabela de dimensão de um esquema dimensional. Hierarquias são compostas de níveis, representados no modelo por instâncias da classe *DimLevel*. Cada instância da classe DimLevel pode conter qualquer número de atributos de uma dimensão, *DimensionObject*, que representam informações a respeito do nível que os contêm.

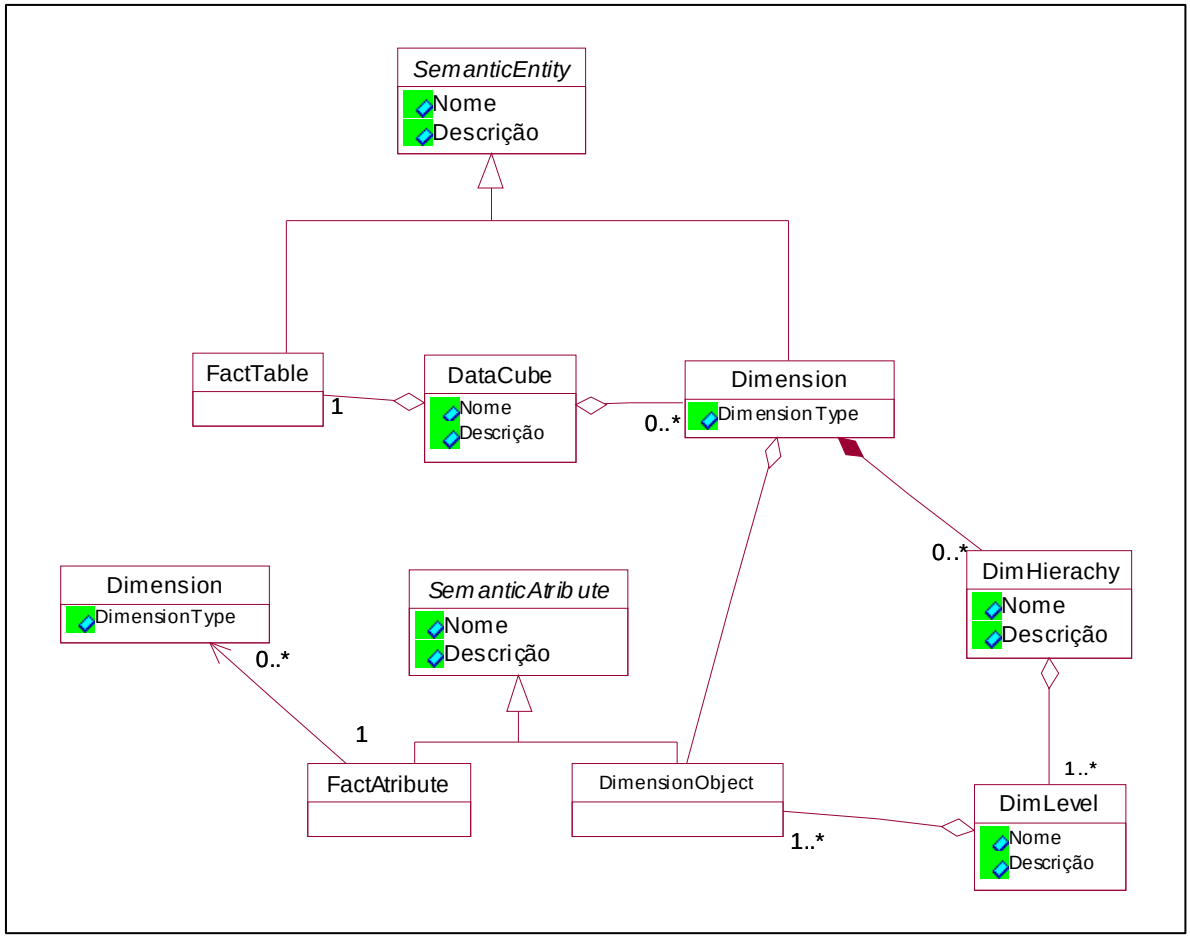

Figura 6.4 Diagrama de classes UML do modelo de metadados conceitual utilizado em [Nascimento, 2001].

6.2.2 Solução de Mapeamento

Nesta seção apresentamos a solução para o mapeamento entre a camada de negócio e os diversos modelos conceituais que poderão estar presentes na camada semântica.

Iniciaremos com alguns conceitos utilizados na solução e depois apresentaremos o padrão de projeto adotado.

6.2.2.1 Composição versus Herança

O objetivo principal do Metha Manager é ser um *framework* caixa-branca. Como vimos no Capítulo 4 um *framework* caixa-branca baseia-se no uso herança para a obtenção de reuso e extensibilidade. Esta seção tem o objetivo de mostrar que a herança pode ser substituída por composição em diversos casos e fornecer uma solução melhor.

Composição e herança são dois mecanismos para reutilizar funcionalidade, porém a maioria dos programadores só considera a herança como ferramenta básica de extensão e reutilização de funcionalidade. A diferença entre composição e herança reside no fato da composição estender uma classe pela delegação de trabalho para outro objeto e a herança estender atributos e métodos de uma classe. Hoje, considera-se que a composição é muito superior à herança na maioria dos casos. A herança deve ser utilizada em alguns contextos que geralmente são relativamente poucos.

6.2.2.1.1 Exemplo de Composição

A composição deve ser usada para estender as responsabilidades pela delegação de trabalho a outros objetos. Vejamos um exemplo no domínio de endereços. Uma empresa tem um endereço, digamos só um. Podemos deixar o objeto empresa responsável pelo objeto endereço e temos agregação composta, ou seja, composição (Figura 6.5).

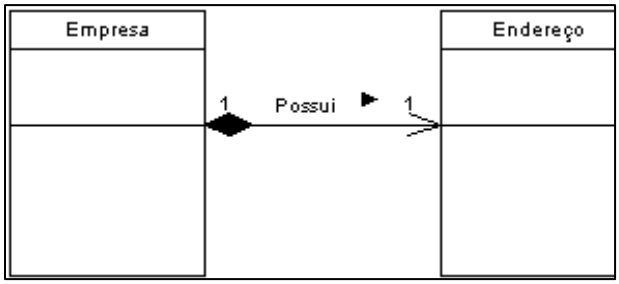

Figura 6.5 Exemplo de Composição.

6.2.2.1.2 Exemplo de Herança

Na herança atributos, conexões a objetos e métodos comuns são definidos na superclasse (classe de generalização) e adicionamos características específicas nas subclasses (classes de especialização).

Vejamos o exemplo de uma transação no domínio de reserva e compra de passagens de avião (Figura 6.6). Uma transação é um momento notável ou intervalo de tempo.

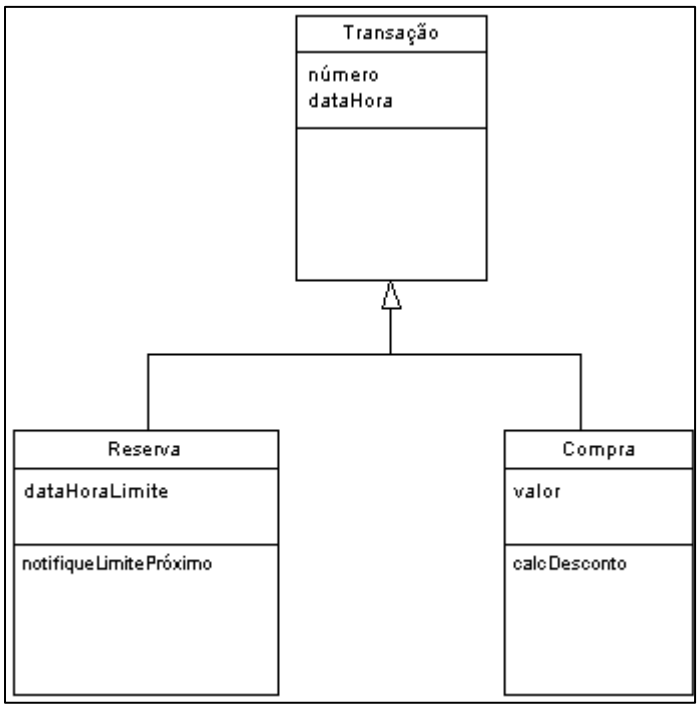

Figura 6.6 Exemplo de Herança.

6.2.2.1.3 Benefícios e Problemas de Herança e Composição

A herança traz os seguintes benefícios:

- Captura o que é comum e o isola daquilo que é diferente;
- A herança é vista diretamente no código.

Problemas da herança:

- O encapsulamento entre classes e subclasses é fraco (acoplamento é forte);
- Mudar uma superclasse pode afetar todas as subclasses;

- Isso viola um dos princípios básicos de projeto O-O (fraco acoplamento);

- Às vezes um objeto precisa ser de uma classe diferente em momentos diferentes.

Com herança, a estrutura fica presa ao código e não pode haver alterações facilmente em tempo de execução, ou seja, a herança é um relacionamento estático que não muda com o tempo. Vejamos mais um exemplo de pessoas envolvidas na aviação para ilustrar a afirmação acima (Figura 6.7).

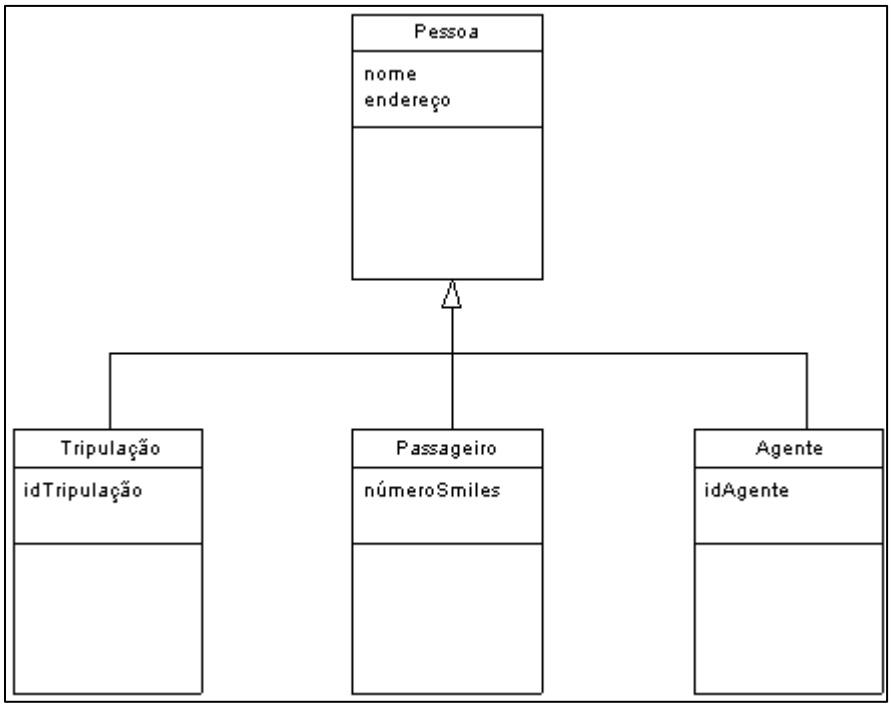

Figura 6.7 Situação de impedimento do uso de herança.

Para a situação da Figura 6.7 temos o problema de uma pessoa poder mudar de papel e assumir combinações de papéis. Fazer papéis múltiplos requer 7 combinações (subclasses).

Podemos solucionar o problema com composição e desta forma uma pessoa pode ter vários papéis possíveis (Figura 6.8). É importante observar que também podemos inverter a composição (uma pessoa tem um ou mais papéis).

A implicação para *interface* de "pessoa" seria o uso de delegação, ou seja, dois objetos estão envolvidos em atender um pedido, por exemplo *setNome.*

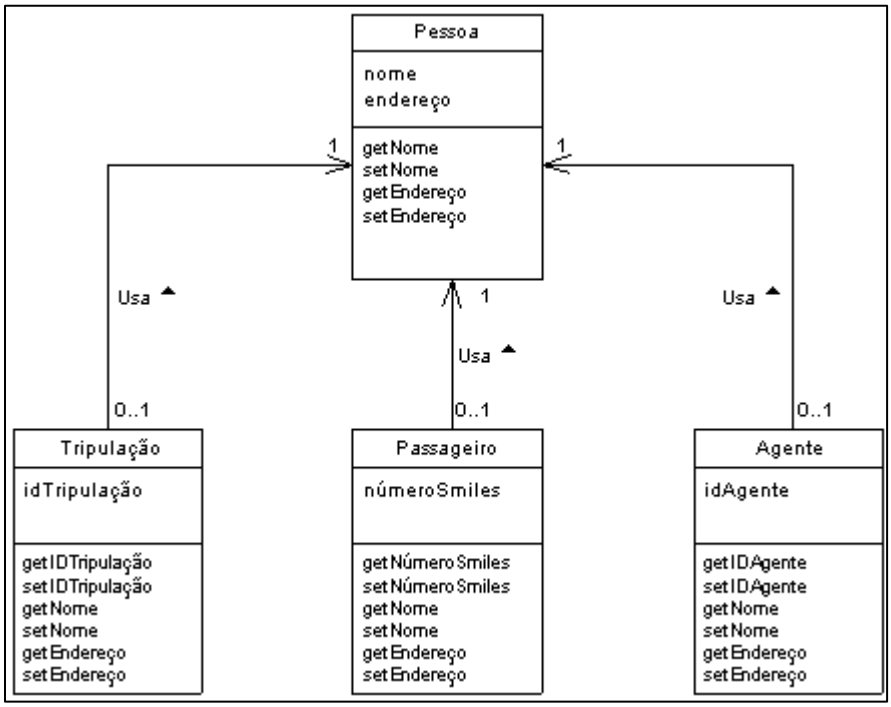

Figura 6.8 Situação propícia ao uso de composição

O objeto *tripulação* delega *setNome* para o objeto *pessoa* que ele tem por composição, isto é semelhante a uma subclasse delegar uma operação para a superclasse (herdando a operação). O fato é que delegação sempre pode ser usada para substituir a herança, neste caso tripulação não é uma pessoa, ela tem uma pessoa. A grande vantagem da delegação é que o comportamento pode ser escolhido em tempo de execução e não está preso ao tempo de compilação. A desvantagem é que um software muito dinâmico e parametrizado é mais difícil de entender do que um software mais estático.

Com o uso da composição não codificamos um comportamento estaticamente, definimos pequenos comportamentos padrão e usamos composição para definir comportamentos mais complexos.

Geralmente usamos herança quando:

- O objeto "é um tipo especial de" e não "um papel assumido por";
- O objeto nunca tem que mudar para outra classe;
- A subclasse estende a superclasse, mas não faz reescrita ou anulação de variáveis e/ou métodos;

- Quando a subclasse expressa tipos especiais de papéis, transações ou dispositivos.

6.2.2.2 Padrão *Role Object*

Existem diversas soluções já apresentadas para a representação dos diversos pápeis que um objeto pode representar em um detrminado cenário de aplicação. Em nossa pesquisa analisamos o padrão de projeto *Role Object* apresentado em [Fowler, 1999] [9]. Não apresentaremos aqui a sugestão de implementação do padrão, pois a mesma está muito bem descrita em [Fowler, 1999].

Resumidamente o padrão *Role Object* sugere que sejam colocadas características comuns em um chamado *Host Object* e objetos separados sejam criados para cada papel. Os clientes pedem ao *Host Object* o papel apropriado para usar as características de um papel. Vejamos o exemplo ilustrado na Figura 6.9. Temos a situação de uma pessoa que pode assumir diversos pápeis numa empresa. A pessoa pode ser um *Engenheiro,* um V*endedor* ou um *Gerente.* Usando *Role Object* o objeto *Pessoa* tem um relacionamento "1 para muitos" com o objeto *Papel de Pessoa* onde *Papel de Pessoa* é um supertipo de *Engenheiro, Vendedor e Gerente.*

A grande vantagem do *Role Object* reside no fato de ao adicionar um novo papel não é necessário mudar a *interface* do *Host Object.*

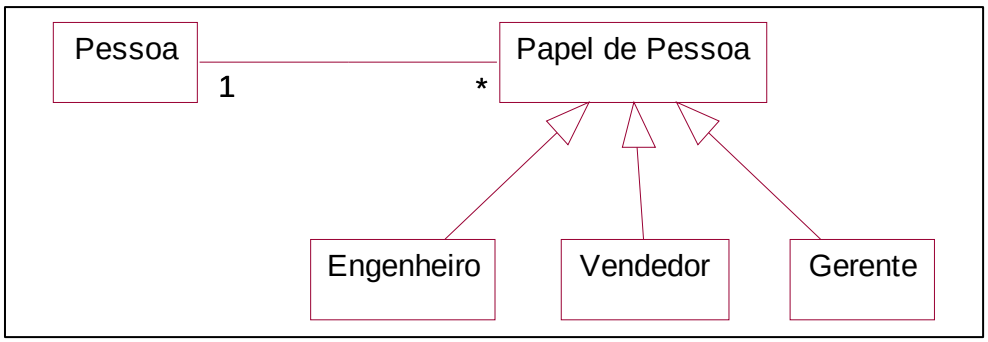

Figura 6.9 Exemplo de aplicação do padrão *Role Object*

[9] Durante a pesquisa o estimado autor repondeu sempre satisfatoriamente aos nossos questionamentos.

6.2.2.3 A Solução

Para fazer o mapeamento da camada de negócio para a camada semântica consideramos que uma classe da camada de negócio pode assumir vários pápeis na camada semântica, ou seja, como poderemos ter diferentes modelos conceituais, um objeto de negócio pode ser conceitualmente representado por diversos objetos.

Partindo desta premissa e analisando os conceitos discutidos nas seções 6.2.2.1 e 6.2.2.2 adotamos uma mescla do padrão *Role Object* e da técnica de composição. Em outras palavras utilizamos o *Role Object* e fizemos extensão através de composição.

Em primeiro lugar, vejamos o mapeamento genérico entre as diversas classes de negócio e as classes conceituais que as mesmas descrevem (Figura 6.10).

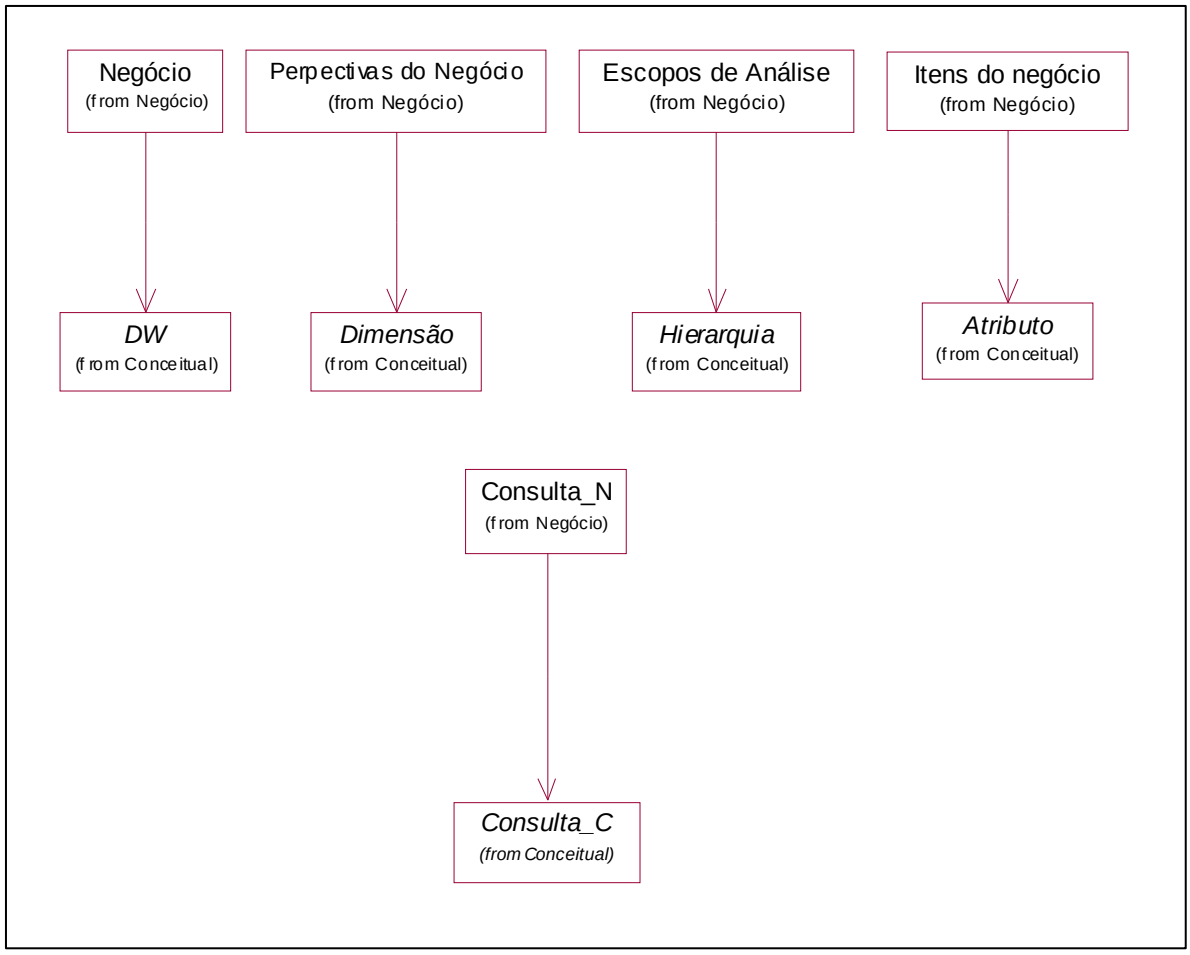

Figura 6.10 Mapeamento genérico entre camada de negócio e camada semântica

Para ilustrar a solução geral tomaremos como exemplo a classe de negócio *Perpectivas de Negócio.* Esta classe pode descrever diversas classes semânticas *Dimensão.* Nos estudos de caso apresentados temos a classe *Dimensão_C* do modelo conceitual de [Jorge, 2001] e a classe *Dimension* do modelo conceitual de [Nascimento, 2001] (Ver seções 6.2.1.1 e 6.2.1.2). Chamaremos estas classes aqui respectivamente de *Dimensão_Edwards* e *Dimensão_Interface.*

Desta forma, temos o relacionamento "1 para muitos" entre *Perpectivas de Negócio* e a classe *Dimensão.* Fazendo uma analogia com o *Role Object* a classe *Dimensão* poderia ser chamada de "Papel das Perpectivas de Negócio". Para concluir, aplicamos composição entre *Dimensão, Dimensão_Edwars* e *Dimensão_Interface* (Figura 6.11).

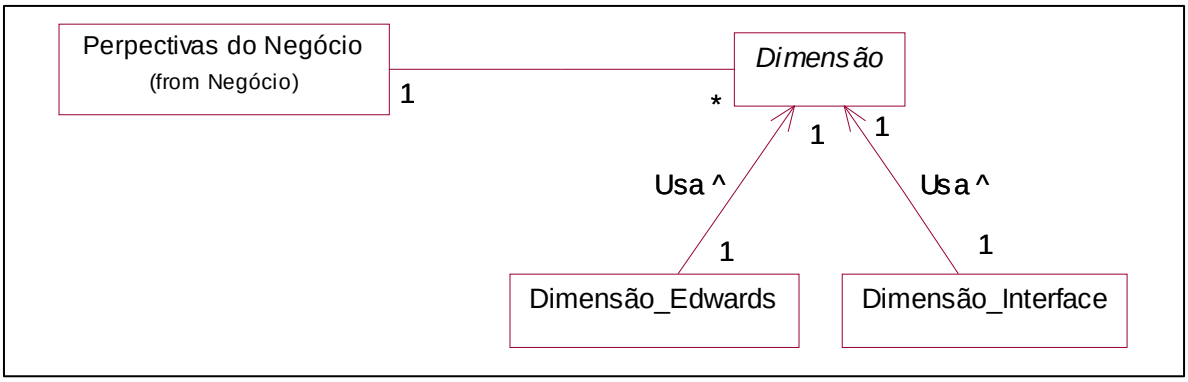

Figura 6.11 Padrão adotado para mapeamento

6.2.3 Extensão do Modelo Lógico

Utilizaremos nesta seção o modelo conceitual utilizado em [Jorge, 2001] para exemplificar o mapeamento e extensão da camada técnica.

No diagrama da figura 6.12 temos as classes do modelo conceitual *DW_C, Cubo_C, Dimensão_C, Dimenao_Cubo_C, Atributo_C* referenciando respectivamente as classes abstratas do modelo lógico *BD_L, Cubo_L, Dimensao_L, Dimensao_Cubo_L e Atributo_L*. As classes do modelo lógico do *framework* não determinam um padrão ou uma implementação específica para um esquema lógico ─ R-OLAP, OR-OLAP, etc. Por isso, no modelo do *framework* lógico todas as classes são abstratas. Elas são utilizadas para o acoplamento das futuras extensões.

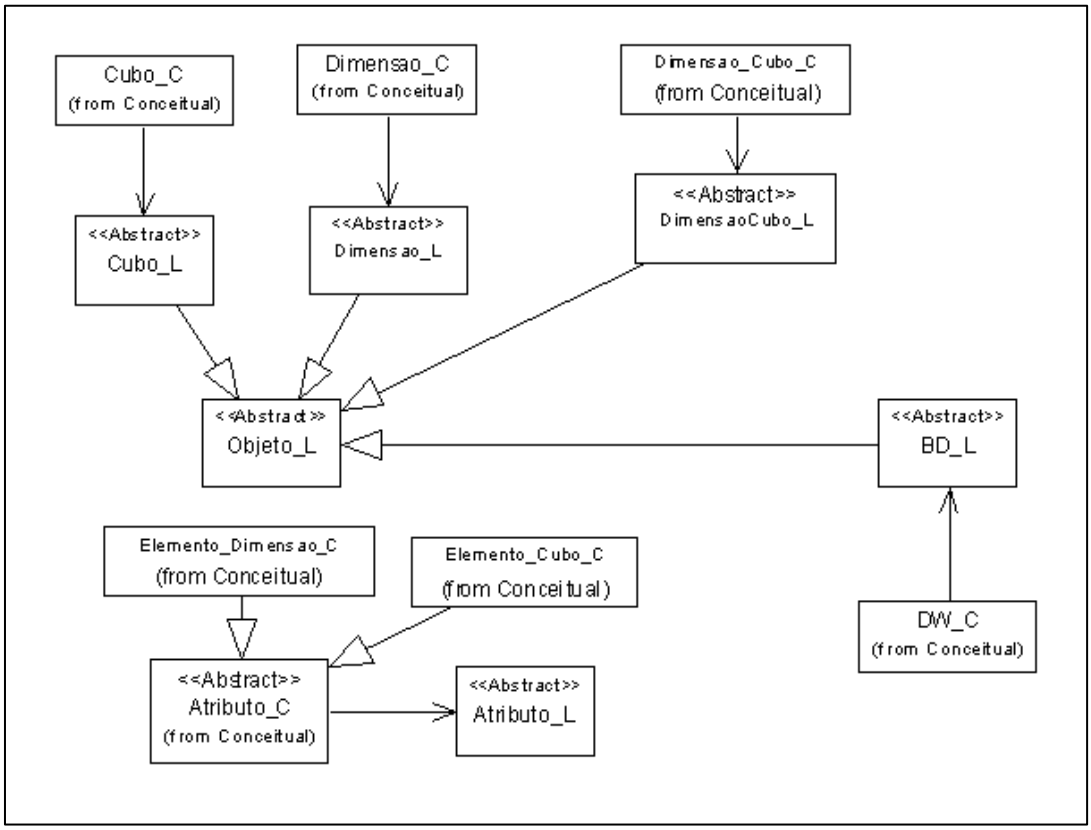

Figura 6.12: Diagrama de classes UML do modelo de metadados lógico do *framework*.

No mapeamento do modelo lógico com o modelo físico, as classes do modelo lógico herdam da superclasse *Objeto_L* que tem um coleção de objetos do tipo *Elemento_F* como ilustra a figura 6.13. A classe abstrata *Elemento_F* será especificada de acordo com a aplicação que estende o *framework*.

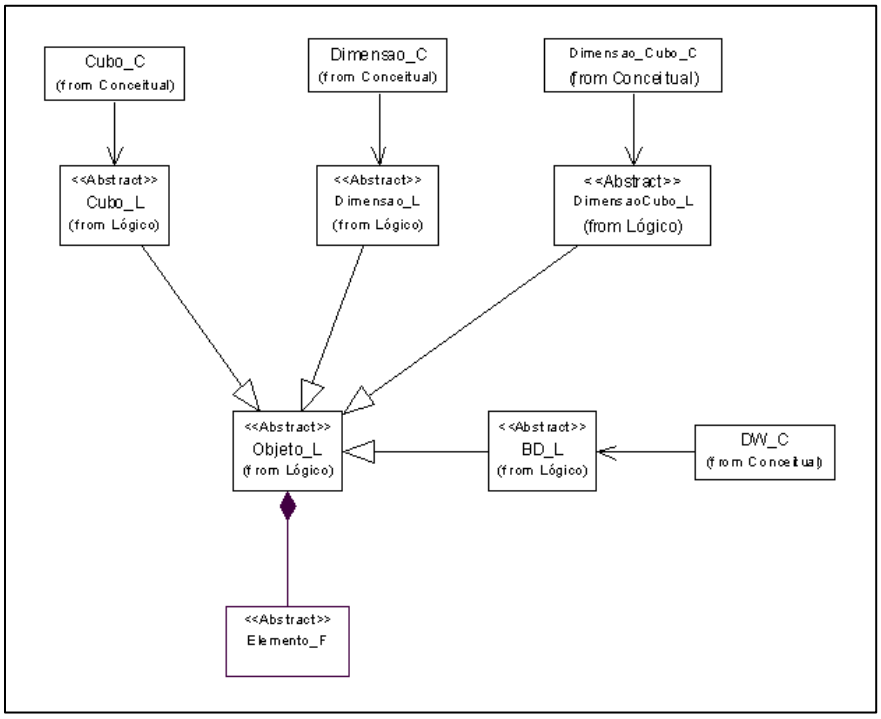

Figura 6.13: Diagrama de classes UML do modelo de metadados físico do *framework*.

As extensões do *framework* para o modelo de metadados lógico e físico são projetadas para uma tecnologia. Exemplificamos duas extensões.

E.1 Metadado-R SqlServer 7

O diagrama de classes da figura 6.14 apresenta um exemplo de extensão do modelo de metadados para representar esquemas DW em estrela relacional implementado no SqlServer 7.

Conforme o diagrama da figura 6.14 são implementadas as classes *BDOdbc, CuboR, DimensaoR, DimensaoCuboR e AtributoR* que estendem respectivamente as classes *DW_L, Cubo_L, Dimensão_L e Atributo_L* do modelo lógico do *framework.*

As classes da extensão, pintadas na figura 6.14, possuem atributos e operação para descrever esquemas em estrela convencionais ─ nome das tabelas de fatos e de dimensões, métodos para a junção das tabelas de dimensões à tabela de fatos etc. Implementam-se também as classes do modelo físico do *framework*, por exemplo, a classe *Index* que estende a classe *Elemento_F*.

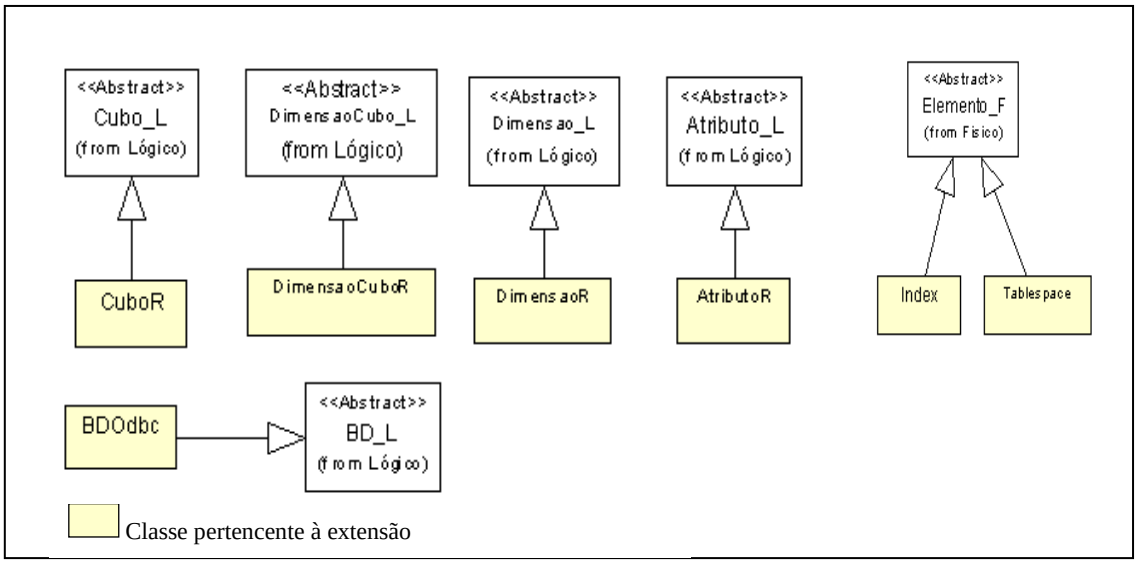

Figura 6.14: Diagrama de classes UML do modelo de metadados lógico e físico da extensão Metadado-R SqlServer 7.

E.2 Metadado-OR Oracle 8i

O diagrama de classes da figura 6.15 é muito semelhante ao da figura 6.14, com a diferença de ser mais simplificado. O motivo da simplificação deve-se à representação de um esquema em estrela OR implementado no Oracle 8i. No nosso trabalho consideramos um esquema projetado com as associações entre dimensões e fatos através de referências. O que proporcionou a eliminação da representação das associações (chaves estrangeiras) no esquema. No modelo refletiu-se na eliminação da extensão da classe *DimensaoCubo_L*.

As classes da extensão OR possuem alguns atributos e operação diferentes da extensão R ─ nome do tipo, atributos complexos, etc. Como na extensão Metadado-R SqlServer 7, implementam-se também as classes do modelo físico do *framework*.

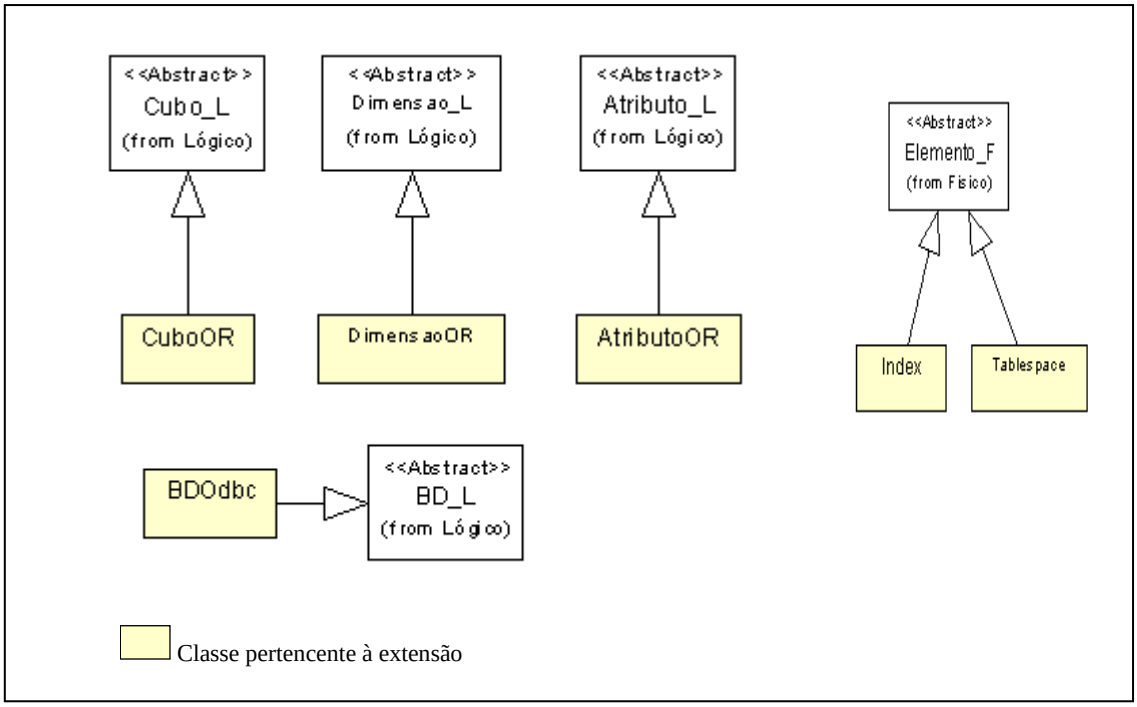

Figura 6.15: Diagrama de classes UML do modelo de metadados lógico e físico da extensão Metadado-OR Oracle 8i.

Capítulo 7

Conclusões e Trabalhos Futuros

Os benefícios que um Repositório de Metadados pode oferecer a um ambiente de Tecnologia da Informação tais como histórico de informações e o grande aumento de qualidade nas pesquisas são notórios. Todavia, a utilização do mesmo em empresas ainda está longe de ser uma tarefa fácil e de baixo custo. A tecnologia envolvida necessita de mão-de-obra qualificada e escassa, e atualmente só grandes empresas têm capital e estrutura para usufruir de tamanha qualidade na administração de informações. Devido a estes fatores, empresas de várias categorias estão tentando desenvolver seu próprio repositório utilizando-se de seus próprios conhecimentos e experiências. Isso certamente resultará no desenvolvimento de Repositórios que possam ser acessíveis financeiramente a pequenas e médias empresas.

Analisando a tecnologia de persistência a ser escolhida para o armazenamento dos metadados, a realização deste trabalho deixou claros alguns pontos importantes:

- O modelo relacional foi amplamente empregado nos últimos quinze anos e não é possível simplesmente ignorar todas as aplicações já construídas. Esses sistemas devem coexistir com as novas aplicações orientadas a objeto;

- Os Bancos de Dados Relacionais estão tornando-se ou adquirindo características de OO-relacional, um ser híbrido. Isto mostra que, mais cedo ou mais tarde, os Bancos de Dados que quiserem sobreviver deverão mudar sua estrutura até chegar, no futuro, em um Banco de Dados efetivamente OO no qual os problemas de desempenho serão minimizados ou resolvidos de maneira clara e a baixo preço. É claro que tudo isso depende de uma aceitação do mercado. O interesse dos grandes fabricantes de *software* é essencial neste processo.

- Apesar do mapeamento entre uma linguagem OO e um Banco de Dados Relacional ser considerado “impuro”, a realidade tem demonstrado que este

mapeamento ainda é a alternativa mais viável, enquanto a promessa de abordagem OO dada pelos Bancos de Dados Orientados a Objetos não se realiza;

- Em se tratando de tecnologia objeto-relacional o suporte recebido das ferramentas *CASE* ainda é deficiente. Como conseqüência, existem no mercado poucos *softwares* que auxiliam ao desenvolvedor na modelagem objeto-relacional para utilizar a tecnologia com profundidade.

A base de dados OR do Metha Manager trouxe algumas vantagens em relação ao desempenho, pois notamos que a utilização de ponteiros e tabelas aninhadas tornam as consultas de um SGBOR mais rápidas e compactas que as consultas feitas através de junções no modelo relacional. Porém, não testamos massivamente o desempenho. De qualquer forma, observa-se que a falta de padrão para o modelo objeto-relacional provoca problemas de transportabilidade, pois cada SGBDOR possui suas idiossincrasias. Sendo assim, conclui-se que ainda é mais viável implementar o Repositório em uma base relacional.

Apesar de termos levado em conta a inexistência de um modelo de gerência de metadados para sistemas de DWing que seja aceito verdadeiramente como padrão, sabemos que se for definida uma versão padrão de meta modelo, a mesma habilitará produtos de apoio a decisão a compartilhar dados e informação. Isto, em troca, proveria ao *Data Warehouse* uma estrutura aberta, comum a todas as ferramentas envolvidas no processo. Havendo um meta modelo padronizado para todas as ferramentas a comunicação entre elas torna-se basicamente viável.

Para que possamos chegar a esse padrão o mesmo deve ser independente de qualquer tecnologia específica, plataforma, ou implementação. Sendo mais explicito um meta modelo padrão não deve estar baseado em qualquer tecnologia específica ou exigir ser implementado em qualquer plataforma específica. Por exemplo, o padrão deve poder ser implementado em um *Mainframe*, Microcomputadores e etc. O meta modelo padrão deve ser desenvolvido em colaboração com todos os fabricantes de s*oftware* e *hardware* chaves, dando ênfase no projeto que não deve ser desenvolvido por qualquer fabricante.

7.1 Contribuições

Como os trabalhos relacionados analisados limitam-se a projetos de ferramentas OLAP com modelos de metadados acoplados a uma determinada tecnologia, ou seja, ferramentas não extensíveis, apresentamos uma inovação na nossa pesquisa.

Com o Metha Manager é possível construir um projeto de DW documentando todas as fases. Como conseqüência, a manutenção e alterações na equipe de desenvolvimento não são traumáticas. Além disso, o analista de negócio pode obter a semântica de todos os dados do negócio analisado, e o analista de sistemas pode navegar desde a camada de negócio até a camada técnica, averiguando todo inter-relacionamento existente.

Por fim, é importante salientar a experiência alcançada no processo de construção de um *framework*. Adotamos um processo já adequado às necessidades de desenvolver uma aplicação não convencional, em que a reutilização é requerida desde a fase conceitual até a fase de implementação.

Contudo, a construção do *framework* trouxe algumas dificuldades tais como:

- Desenvolvimento exaustivo para o alcance da reutilização;
- Custo alto, visto que é importante desenvolver duas aplicações antes de iniciar o projeto do *framework*;
- Para o desenvolvedor, a criação de novas extensões no Metha Manager demanda um esforço e um conhecimento do projeto do *framework.*

7.2 Trabalhos Futuros

Como nosso projeto, por questões de tempo, não atingiu todas as metas para a construção de um repositório “perfeito”, podemos identificar possíveis trabalhos futuros. Enumeramo-los:

- Implementação de um controle de versões dos programas e/ou procedimentos que envolvem o ciclo de um DW para auxiliar programadores do mesmo;
- Tratamento da inserção de metadados de negócio não-estruturados (fotos, textos etc..) no repositório, por exemplo, associar uma notícia de jornal a uma consulta ao nível de negócio;

- Importação de metadados através de XML;
- Evolução do *framework* caixa-branca para *caixa-preta* com o intuito de diminuir o esforço empregado para a reutilização [Roberts, 2000];
- Prover a possibilidade de extensão também para a camada de negócio;
- Analisar uma solução de mapeamento entre a camada semântica e de negócio no sentido de capturar o significado de classes de um modelo conceitual A para um modelo conceitual B;
- Extensão do modelo atual para uma abrangência maior de metadados que descrevam a transformação dos dados;
- Adaptação ao metamodelo CWM.

Bibliografia

[Agrawal, 1999] AGRAWAL R. & GUPTA A. & SARAWAGI S., *Modeling Multidimensional Databases*, IBM, 1999.

[Banese, 2000] BANCO DO ESTADO DE SERGIPE. *Levantamento de Necessidades e Documentação do Projeto e Implementação do Data Warehouse do Banco do Estado de Sergipe S.A.* .2000.

[Barquin *et al*., 1997] BARQUIN, R. C. *et al*. *Planning and Designing The Data Warehouse*. New Jersey: Prentice Hall, 1997.

[Bhargava & Power, 2001] BHARGAVA, H., POWER, D. J. Power. *Decision Support Systems and Web Technologies: A Status Report*. Prepared for AMCIS 2001, Americas Conference on Information Systems, Boston, Massachusetts, August 3th - 5th, 2001, "Decision Support Systems" Mini Track. (URL *http://dssresources.com/papers/dsstrackoverview.pdf*).

[Bray *et al.*, 2001] BRAY, T., PAOLI, J., McQUEEN C. M. S. ,MALER, E. *Extensible Markup Language*. Word Wide Web, http://www.w3.org/TR/2000/REC-xml-20001006, Novembro 2001

[Bobrowski, 1998] BOBROWSKI S., *Oracle8 Architecture*, McGraw-Hill, 1998.

[Booch *et al*., 1999] BOOCH, G. *et al*. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[Braker & Moore, 1996] STONE BRAKER, M., MOORE, D. *Object-Relational DBMSs: The Next Great Wave*. San Francisco: Morgan Kaufmann, 1996.

[Cantor, 1998] CANTOR, M. R. *Object Oriented Project Management with UML*. John Wiley & Sons, 1998.

[Chang *et al.*, 2001] CHANG, Ben, SCARDINA, Mark, KARUN, K., KIRITZON, Stefan, MACKY, Ian, NOVOSELSKY, Anguel, RAMAKRIHNAN, Nirajan. *Oracle XML. O Manual Oficial***.** Rio de Janeiro: Campus, 2001.

[Chaudhuri & Dayal, 1997] CHAUDHURI, S., DAYAL, U. *An Overview of Data Warehousing and OLAP Technology*. SIGMOD Record, vol. 26, no. 1, pp. 65-74, March 1997.

[Coalition, 1998] Metadata Coalition. *Metadata Interchange Specification version 1.1*. World Wide Web, *http://www.mdcinfo.com*, August 1998.

[Coalition, 1999] Metadata Coalition. *Open Information Model version 1.0*. World Wide Web, *http://www.mdcinfo.com*, August 1999.

[Date, 1998] DATE C.J. & DARWEN H., *Foundation For Object/Relational Databases - The Third Manifesto*, Addison-Wesley, 1998.

[Dinter *et al*., 1998] DINTER, B. *et al*. *The OLAP Market: State of the Art and Research Issues*. Proceedings of the ACM 1st International Workshop on Data Warehousing and OLAP, pp. 22-27, Washington, United States, 1998.

[Elmasri & Navathe, 1994] ELMASRI, R., NAVATHE, S. B. *Fundamentals of Database Systems*. 2nd edition. CA: Benjamim/Cummings, 1994.

[Firestone, 1997] FIRESTONE J. M., *Object-Oriented Data Warehousing Information Systems*, Inc., White Paper, 1997.

[Flanagan, 1997] FLANAGAN, D. *Java in a Nutshell*. 2nd edition. CA: O'Reilly, 1997.

[Fowler & Scott, 1999] FOWLER, M., SCOTT, K. *UML Distilled*: A Brief Guide to the Standard Object Modeling Language. 2nd edition. Addison Wesley Longman, 1999.

[Fowler, 1999] FOWLER, Martin *Dealing with Roles*, World Wide Web, *http://www.martinfowler.com/apsupp/roles.pdf*, 1999.

[Gamma et al., 1994] GAMMA, E., HELM, R., JOHNSON, R., VLISSIDES, J., *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[Giovinazzo, 2000] GIOVINAZZO,William. *Introduction to CWM* – Oracle site (http://oracle.com/oramag/oracle/00-jul/o40ind.html).

[Golfarelli, 1998] GOLFARELLI, M., MAIO, D., RIZZI, S. *Conceptual Design of Data Warehouses from E/R Schemes*. IEEE Proceedings of the Hawaii International Conference On Systems Sciences, January 1998.

[Grand, 1998] GRAND M., *Patterns in Java: a catalog of reusable design patterns illustrated with UML*, Volume I, John Wiley & Sons, 1998.

[Gupta, 1997] GUPTA, V. R. *An Introduction to Data Warehousing*. System Services Corporation, August 1997. *http://www.system-services.com/dwintro.htm*.

[Hammer *et al*., 1995] HAMMER J. *et al*. *The Stanford Data Warehousing Project*. Computer Science Departament, Stanford University, 1995.

[Harding, 1998] HARDING, J.M.; *Metadata Management*; Vision Unlimited; 1998.

[Huynh, 2000] HUYNH, T. N., MANGISENGI, O., TJOA, A. M., *Metadata for Object-Relational Data Warehouse*. Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'2000), pages (3-1)-(3-9), 2000.

[IBM, 1999] IBM, *Building Object-Oriented Frameworks*, White Paper, 1999.

[Inmon, 1996] INMON, W. H., ZACHMAN, John A. *Data Stores Data Warehousing and the Zachman Framework – Managing Enterprise Knowledge*. Editora Campus, 1996.

[Inmon, 1997] INMON, W. H. *Como Construir o Data Warehouse*. Tradução da segunda edição. Rio de Janeiro: Campus, 1997.

[Iyengar, 2000] IYENGAR, Sridhar. CWM Audio Briefing:*The Key to Integrating Business Intelligence* – OMG Site(http://www.omg.org/technology/cwm/index.htm), 2000.

[Jarke *et al.*, 1999] JARKE, M. *et al*. *Fundamentals of Data Warehouses*. New York: Springer-Verlag, 1999.

[Jorge, 2001] JORGE, E. M. F. *Análise, Projeto e Implementação de um Servidor de Dw Extensível*, Dissertação de Mestrado - UFPB-COPIN, 2001.

[Kimball, 1996] KIMBALL, R. *The Data Warehouse Toolkit*. New York: John Wiley & Sons, 1996.

[Kimball, 1996b] KIMBALL, R. *Aggregate Navigation With (Almost) No Metadata*. DBMS Data Warehouse Supplement, August 1996.

[Kimball, 1997] KIMBALL, R. *A Dimensional Modeling Manifesto*. DBMS and Internet Systems, July 1997.

[Kimball *et al*., 1998] KIMBALL, R. *et al*. *The Data Warehouse Lifecycle Toolkit*. New York: John Wiley & Sons, 1998.

[Klein, 1999] KLEIN L. Z., *A Tecnologia Objeto-Relacional em Ambientes de Data Warehouses: Uso de Séries Temporais como Tipo de Dado Não Convencional*, M. L. M. Campos & A. K. Tanaka, Anais do XIV SBBD, Florianópolis-SC, 1999, 365-378.

[Landin, 1995] LANDIN N & NIKLASSON A., *Development of Object-Oriented Framework*, Conden:Lutedx,1995.

[Lehmann & Jaszewski, 1999] LEHMANN, P., JASZEWSKI, J. *Business Terms as a Critical Success Factor for Data Warehousing*. Proceedings of the International Workshop on Design and Management of Data Warehouses, Heidelberg, Germany, 1999.

[Marco, 2000] MARCO, David. *Building and Managing the Meta Data Repository*: A Full Lifecycle Guide. Wiley Computer Publishing, 2000.

[Nascimento, 2001] NASCIMENTO, André Vinicius R. P., *Uma Interface para Consultas a Sistemas de Suporte à Decisão Baseada em Linguagem Natural, Casamento de Padrões e Metadados*, Dissertação de Mestrado - UFPB-COPIN, 2001.

[OMG, 2001] OMG. World Wide Web, www.omg.org/technology/cwm/index.htm, 2001.

[Poe et al., 1998] POE, V. *et al*. *Building a Data Warehouse for Decision Support*. Prentice Hall PTR, 1998.

[Power, 1997] POWER, D. J. *Justifying a Data Warehouse Project*. The On-Line Executive Journal for Data-Intensive Decision Support, vol. 2, no. 5, February 1998.

[Power, 2000] POWER, D. J. *Supporting Decision-Makers: An Expanded Framework.* DSSREsources.COM, World Wide Web, *http://dssresources.com/papers/supportingdm/sld001.htm*, 2000.

[Raden, 1996] RADEN, N. *Modeling the Data Warehouse*. World Wide Web, *http://user.aol.com/nraden/iw0196_1.htm,* 1996.

[Ramakrishnan, 1998] RAMAKRISHNAN, R. *Database Management Systems*. USA: WBC/McGraw-Hill, 1998.

[Rational, 1999] RATIONAL. World Wide Web, http:://www.rational.com/uml/index.htmpl, 1999.

[Roberts, 2000] ROBERTS D. & JOHNSON R, *Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks*, University of Illinois,2000.

[Rogers, 1997] ROGERS G. F., *Framework-Based Software Development in C++*, Prentice Hall, 1997.

[Rudloff, 1996] RUDLOFF, D. *Terminological Reasoning and Conceptual Modeling for Datawarehouse*. KRDB-96, Budapest.

[Rumbaugh *et al*., 1991] RUMBAUGH, J. *et al*. *Object Oriented Modeling and Design.* Prentice Hall, 1991.

[Sachdeva, 1999] SACHDEVA, S. *Metadata for Data Warehouse*. Sybase, Inc. World Wide Web, *http://www.sybase.com/services/dwpractice/meta.html*, 1999.

[Sampaio, 2001]. SAMPAIO M. C. & JORGE E. F.& BAPTISTA C. S., *Metadata for an Extensible Data Warehouse Server*, UFPB-COPIN, 2001.

[Sherman, 1997] SHERMAN, R. P. *Metadata: The Missing Link - Business Information Directories Catalog Decision-Support Information Throughout the Enterprise*. DBMS and Internet Systems, August 1997.

[Stöhr *et al*., 1999] STÖHR, T., MÜLLER, R., RAHM, E. *An Integrative and Uniform Model for Metadata Management in Data Warehousing Environments*. Proceedings of the International Workshop on Design and Management of Data Warehouses, Heidelberg, Germany, 1999.

[Srivastava & Chen, 1999] SRIVASTAVA, J., CHEN P. *Warehouse Creation: A Potential Roadblock to Data Warehousing*. IEEE Transactions on Knowledge and Data Engineering, vol. 11, no. 1, pp. 118-126, February 1999.

[Ullman & Widow, 1997] ULLMAN, J. D., WIDOW, J. *A First Course in Database Systems*. New Jersey: Prentice Hall, 1997.

[Vaduva, 2000] A. VADUVA, K. R. DITTRICH. *Metadata Management for Data Warehousing: Between Vision and Reality*, Technical Report 2000.08., Department of Information Technology, University of Zurich, December 2000.

[Vaduva, 2001] A. VADUVA, K. R. DITTRICH. *Metadata Management for Data Warehousing*: *Between Vision and Reality*. In Proc. of the International Database Engineering & Applications Symposium, IDEAS'01, Grenoble, France July 2001.

[Vetterli, 2000] VETTERLI T. & VADUVA, A. & STAUDT M., *Metadata Standards for Data Warehousing: Open Information Model vs. Common Warehouse Metadata*, SIGMOD Record, Vol 29, No. 3, 2000.

[Vetterli, 2001] A. VADUVA, T. VETTERLI: *Metadata Management for Data Warehousing*: *An Overview*. Intl. Journal of Cooperative Information Systems (IJCIS), 10(3), September 2001.

Apêndice

XML gerado para o servidor Edwards

Este anexo apresenta a definição dos arquivos DTD (estrutura) e XML (dados) que compõem os metadados gerados para o servidor Edwards. Cada DW no Edwards é representado por um arquivo XML que contém informações sobre os metadados conceituais e lógicos. São apresentados dois arquivos DTD:

1. O modelo conceitual (independente da tecnologia de armazenamento)
2. O modelo conceitual e lógico para a extensão Relacional e Objeto-Relacional

Por fim, são apresentados os arquivos XML baseados nos DTD´s para um esquema DW Vendas.

1.1 O esquema conceitual (independente da tecnologia de armazenamento)

2.2 O esquema conceitual e lógico para a extensão Relacional

DTD

1 O modelo conceitual (independente da tecnologia de armazenamento)

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT DW (dw)>
<!ELEMENT dw (nome, dimensoes, cubos, hierarquias)>
<!ELEMENT dimensoes (dimensao+)>
<!ELEMENT dimensao (nome, atributos)>
<!ELEMENT atributos (atributo+)>
<!ELEMENT atributo (nome, tipo, medida?)>
<!ELEMENT cubos (cubo)>
<!ELEMENT cubo (nome, cubodimensoes, atributos)>
<!ELEMENT cubodimensoes (cubodimensao+)>
<!ELEMENT cubodimensao (nome, grao)>
```

<!ELEMENT hierarquias (hierarquia+)>
<!ELEMENT hierarquia (nome, lista)>
<!ELEMENT lista (elemento_d+)>
<!ELEMENT elemento_d (nomepai)>
<!ELEMENT grao (#PCDATA)>
<!ELEMENT medida (#PCDATA)>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT nomepai (#PCDATA)>
<!ELEMENT tipo (#PCDATA)>

2. O modelo conceitual e lógico para a extensão Relacional

<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT DW (dw)>
<!ELEMENT dw (nome, login, senha, datasource, tecnologiadw, dimensoes, cubos, hierarquias)>
<!ELEMENT dimensoes (dimensao+)>
<!ELEMENT dimensao (nome, tabela, atributos)>
<!ELEMENT atributos (atributo+)>
<!ELEMENT atributo (nome, tipo, medida?, campo)>
<!ELEMENT cubos (cubo)>
<!ELEMENT cubo (nome, tabela, cubodimensoes, atributos)>
<!ELEMENT cubodimensoes (cubodimensao+)>
<!ELEMENT cubodimensao (nome, grao, linkdimensao, linkcubo)>
<!ELEMENT hierarquias (hierarquia+)>
<!ELEMENT hierarquia (nome, lista)>
<!ELEMENT campo (#PCDATA)>
<!ELEMENT elemento_d (nomepai, nome)>
<!ELEMENT grao (#PCDATA)>
<!ELEMENT datasource (#PCDATA)>
<!ELEMENT linkcubo (#PCDATA)>
<!ELEMENT linkdimensao (#PCDATA)>
<!ELEMENT lista (elemento_d+)>
<!ELEMENT login (#PCDATA)>
<!ELEMENT medida (#PCDATA)>

<!ELEMENT nome (#PCDATA)>

<!ELEMENT nomepai (#PCDATA)>

<!ELEMENT senha EMPTY>

<!ELEMENT tabela (#PCDATA)>

<!ELEMENT tecnologiadw (#PCDATA)>

<!ELEMENT tipo (#PCDATA)>

Obs: O DTD Objeto-Relacional é igual ao DTD relacional com a diferença do elemento cubodimensao que não possui os atributos *linkdimensao e linkcubo*

<!ELEMENT cubodimensao (nome, grao)>

XML (DW VENDAS)

1.1 O modelo conceitual (independente da tecnologia de armazenamento)

```
<?xml version="1.0"?>
<DWVendas>
        <dw>
                <nome>Vendas_SQLServer</nome>
                <dimensoes>
                        <dimensao>
                                <nome>Tempo</nome>
                        <atributos>
                                        <atributo>
                                                <nome>id</nome>
                                                <tipo>NUMERIC</tipo>
                                        </atributo>
                                        <atributo>
                                                <nome>dia</nome>
                                                <tipo>NUMERIC</tipo>
                                        </atributo>
                                        <atributo>
                                                <nome>mes</nome>
                                                <tipo>NUMERIC</tipo>
                                        </atributo>
                                        <atributo>
                                                <nome>Ano</nome>
                                                <tipo>NUMERIC</tipo>
                                        </atributo>
                                        <atributo>
                                                <nome>Data</nome>
                                                <tipo>TEXT</tipo>
```

```
                    </atributo>
                    <atributo>
                        <nome>MesAno</nome>
                        <tipo>TEXT</tipo>
                    </atributo>
                </atributos>
            </dimensao>
            <dimensao>
                <nome>Produto</nome>
                <atributos>
                    <atributo>
                        <nome>id</nome>
                        <tipo>NUMERIC</tipo>
                    </atributo>
                    <atributo>
                        <nome>Nome</nome>
                        <tipo>TEXT</tipo>
                    </atributo>
                    <atributo>
                        <nome>Categoria</nome>
                        <tipo>TEXT</tipo>
                    </atributo>
                    <atributo>
                        <nome>Familia</nome>
                        <tipo>TEXT</tipo>
                    </atributo>
                </atributos>
            </dimensao>
            <dimensao>
                <nome>Loja</nome>
                <atributos>
<atributo>
                        <nome>id</nome>
                        <tipo>NUMERIC</tipo>
                    </atributo>
                    <atributo>
                        <nome>Nome</nome>
                        <tipo>TEXT</tipo>
                    </atributo>
                    <atributo>
                        <nome>Estado</nome>
                        <tipo>TEXT</tipo>
                    </atributo>
                </atributos>
            </dimensao>
            <dimensao>

                <nome>MesAno</nome>
                <atributos>
<atributo>
```

```
                                        <nome>id</nome>
                                        <tipo>NUMERIC</tipo>
                                </atributo>
                                <atributo>
                                        <nome>MesAno</nome>
                                        <tipo>TEXT</tipo>
                                </atributo>
                        </atributos>
                </dimensao>
                <dimensao>
                        <nome>Categoria</nome>
                        <atributos>
    <atributo>
                                        <nome>id</nome>
                                        <tipo>NUMERIC</tipo>
                                </atributo>
                                <atributo>
                                        <nome>Nome</nome>
                                        <tipo>TEXT</tipo>
                                </atributo>
                        </atributos>
                </dimensao>
                <dimensao>
                        <nome>Familia</nome>
                        <atributos>
                                <atributo>
                                        <nome>id</nome>
                                        <tipo>NUMERIC</tipo>
                                </atributo>
                                <atributo>
                                        <nome>Nome</nome>
                                        <tipo>TEXT</tipo>
                                </atributo>
                        </atributos>
                </dimensao>
</dimensoes>
<cubos>
                <cubo>
                        <nome>c1</nome>

                        <cubodimensoes>
                                <cubodimensao>
                                        <nome>Tempo</nome>
                                        <grao>Data</grao>
                                </cubodimensao>
                                <cubodimensao>
                                        <nome>Produto</nome>
                                        <grao>Noem</grao>
                                </cubodimensao>
                                <cubodimensao>
```

```
                                        <nome>Loja</nome>
                                        <grao>Nome</grao>
                                    </cubodimensao>
                                </cubodimensoes>
                                <atributos>
                                    <atributo>
                                        <nome>Quantidade</nome>
                                        <tipo>NUMERIC</tipo>
                                        <medida>true</medida>
                                    </atributo>
                                    <atributo>
                                        <nome>Faturamento</nome>
                                        <tipo>NUMERIC</tipo>
                                        <medida>true</medida>
                                    </atributo>
                                </atributos>
                            </cubo>
                        </cubos>
                        <hierarquias>
                            <hierarquia>
                                <nome>produto_cat</nome>
                                <lista>
                                    <elemento_d>
                                        <nomepai>Produto</nomepai>
                                    </elemento_d>
                                    <elemento_d>
                                        <nomepai>Categoria</nomepai>
                                    </elemento_d>
                                    <elemento_d>
                                        <nomepai>Familia</nomepai>
                                    </elemento_d>
                                </lista>
                            </hierarquia>
                            <hierarquia>
                                <nome>data_mesano</nome>
                                <lista>
                                    <elemento_d>
                                        <nomepai>Tempo</nomepai>
                                    </elemento_d>
                                    <elemento_d>
                                        <nomepai>MesAno</nomepai>
                                    </elemento_d>
                                </lista>
                            </hierarquia>
                        </hierarquias>
                    </dw>
                </DWVendas>
```

2.2 O esquema conceitual e lógico para a extensão Relacional

```
<?xml version="1.0"?>
<ConfigDWVendas>
	<dw>
		<nome>DW_Vendas</nome>
		<login>sa</login>
		<senha></senha>
		<datasource>DWVENDAS1</datasource>
		<tecnologiadw>1</tecnologiadw>
		<dimensoes>
			<dimensao>
				<nome>Tempo</nome>
				<tabela>dwvendas.dbo.Tempo</tabela>
				<atributos>
					<atributo>
						<nome>id</nome>
						<tipo>NUMERIC</tipo>
						<campo>id</campo>
					</atributo>
					<atributo>
						<nome>dia</nome>
						<tipo>NUMERIC</tipo>
						<campo>dia</campo>
					</atributo>
					<atributo>
						<nome>mes</nome>
						<tipo>NUMERIC</tipo>
						<campo>mes</campo>
					</atributo>
					<atributo>
						<nome>Ano</nome>
						<tipo>NUMERIC</tipo>
						<campo>ano</campo>
					</atributo>
					<atributo>
						<nome>Data</nome>
						<tipo>TEXT</tipo>
						<campo>data</campo>
					</atributo>
					<atributo>
						<nome>MesAno</nome>
						<tipo>TEXT</tipo>
						<campo>mesano</campo>
					</atributo>
				</atributos>
			</dimensao>
			<dimensao>
				<nome>Produto</nome>
				<tabela>dwvendas.dbo.Produto</tabela>
				<atributos>
```

```
                    <atributo>
                        <nome>id</nome>
                        <tipo>NUMERIC</tipo>
                        <campo>id</campo>
                    </atributo>
                    <atributo>
                        <nome>Nome</nome>
                        <tipo>TEXT</tipo>
                        <campo>nome</campo>
                    </atributo>
                    <atributo>
                        <nome>Categoria</nome>
                        <tipo>TEXT</tipo>
                        <campo>categoria</campo>
                    </atributo>
                    <atributo>
                        <nome>Familia</nome>
                        <tipo>TEXT</tipo>
                        <campo>familia</campo>
                    </atributo>
                </atributos>
            </dimensao>
            <dimensao>
                <nome>Loja</nome>
                <tabela>dwvendas.dbo.Loja</tabela>
                <atributos>
<atributo>
                    <nome>id</nome>
                    <tipo>NUMERIC</tipo>
                    <campo>id</campo>
                </atributo>
                <atributo>
                    <nome>Nome</nome>
                    <tipo>TEXT</tipo>
                    <campo>nome</campo>
                </atributo>
                <atributo>
                    <nome>Estado</nome>
                    <tipo>TEXT</tipo>
                    <campo>estado</campo>
                </atributo>
                </atributos>
            </dimensao>
            <dimensao>

                <nome>MesAno</nome>
                <tabela>dwvendas.dbo.MesAno</tabela>

                <atributos>
<atributo>
```

```
                                        <nome>id</nome>
                                        <tipo>NUMERIC</tipo>
                                        <campo>id</campo>
                                </atributo>
                                <atributo>
                                        <nome>MesAno</nome>
                                        <tipo>TEXT</tipo>
                                        <campo>mesano</campo>
                                </atributo>
                        </atributos>
                </dimensao>
                <dimensao>
                        <nome>Categoria</nome>
                        <tabela>dwvendas.dbo.Categoria</tabela>
                        <atributos>
  <atributo>
                                        <nome>id</nome>
                                        <tipo>NUMERIC</tipo>
                                        <campo>id</campo>
                                </atributo>
                                <atributo>
                                        <nome>Nome</nome>
                                        <tipo>TEXT</tipo>
                                        <campo>nome</campo>
                                </atributo>
                        </atributos>
                </dimensao>
                <dimensao>
                        <nome>Familia</nome>
                        <tabela>dwvendas.dbo.Familia</tabela>
                        <atributos>
                                <atributo>
                                        <nome>id</nome>
                                        <tipo>NUMERIC</tipo>
                                        <campo>id</campo>
                                </atributo>
                                <atributo>
                                        <nome>Nome</nome>
                                        <tipo>TEXT</tipo>
                                        <campo>Nome</campo>
                                </atributo>
                        </atributos>
                </dimensao>
</dimensoes>
<cubos>
                <cubo>
                        <nome>c1</nome>
                        <tabela>dwVendas.dbo.Vendas</tabela>
                        <cubodimensoes>
                                <cubodimensao>
```

```
                    <nome>Tempo</nome>
                    <grao>id</grao>
                    <linkdimensao>dwVendas.dbo.Tempo.ID</linkdimensao>
                    <linkcubo>dwVendas.dbo.Vendas.ID_Tempo</linkcubo>
                </cubodimensao>
                <cubodimensao>
                    <nome>Produto</nome>
                    <grao>id</grao>
                    <linkdimensao>dwVendas.dbo.Produto.ID</linkdimensao>
                    <linkcubo>dwVendas.dbo.Vendas.ID_Produto</linkcubo>
                </cubodimensao>
                <cubodimensao>
                    <nome>Loja</nome>
                    <grao>id</grao>
                    <linkdimensao>dwVendas.dbo.Loja.ID</linkdimensao>
                    <linkcubo>dwVendas.dbo.Vendas.ID_Loja</linkcubo>
                </cubodimensao>
            </cubodimensoes>
            <atributos>
                <atributo>
                    <nome>Quantidade</nome>
                    <tipo>NUMERIC</tipo>
                    <medida>false</medida>
                    <campo>quantidade</campo>
                </atributo>
                <atributo>
                    <nome>Faturamento</nome>
                    <tipo>NUMERIC</tipo>
                    <medida>false</medida>
                    <campo>faturamento</campo>
                </atributo>
            </atributos>
        </cubo>
</cubos>
<hierarquias>
        <hierarquia>
            <nome>produto_cat</nome>
            <lista>
                <elemento_d>
                    <nomepai>Produto</nomepai>
                    <nome>id</nome>
                </elemento_d>
                <elemento_d>
                    <nomepai>Categoria</nomepai>
                    <nome>id</nome>
                </elemento_d>
                <elemento_d>
                    <nomepai>Familia</nomepai>
                    <nome>id</nome>
                </elemento_d>
```

```
                    </lista>
                </hierarquia>
                <hierarquia>
                    <nome>data_mesano</nome>
                    <lista>
                        <elemento_d>
                            <nomepai>Tempo</nomepai>
                            <nome>id</nome>
                        </elemento_d>
                        <elemento_d>
                            <nomepai>MesAno</nomepai>
                            <nome>id</nome>
                        </elemento_d>
                    </lista>
                </hierarquia>
            </hierarquias>
        </dw>
</ConfigDWVendas>
```

Glossário

API	Aplication Programming Interface
BD	Banco de Dados
CASE	Computer-Aided Software Engineering
DTD	Document Type Description
DW	Data Warehouse
DWing	Data Warehousing
EDW	Enterprise Data Warehouse
ER	Entidade-Relacionamento
FDW	Federated Data Warehouse
HTTP	Hipertext Transfer Protocol
JDBC	Java Database Connectivity
MDIS	Metadata Interchange Specification
MOLAP	Multidimensional OLAP
ODBC	Open Database Connectivity
ODS	Operational Data Store
OIM	Open Information Model
OLAP	On-Line Analytical Processing
OLTP	On-Line Transaction Processing
OO	Orientação a Objetos
OR	Objeto-relacional
RMI	Remote Method Invocation
ROLAP	Relational OLAP
SGBD	Sistema Gerenciador de Banco de Dados
SGBDR	Sistemas de Gerência de Bancos de Dados Relacionais
SGBDOO	Sistemas de Gerência de Bancos de Dados Orientados a Objetos
SGBDOR	Sistemas de Gerência de Bancos de Dados Objeto-Relacionais
UDT	User Defined Type
SQL	Structured Query Language

SSD	Sistema de Suporte à Decisão
UML	Unified Modeling Language
XML	Extensible Markup Language