ANÁLISE, PROJETO E IMPLEMENTAÇÃO DE UM SERVIDOR DE DW EXTENSÍVEL

EDUARDO MANUEL DE FREITAS JORGE

Análise, Projeto e Implementação de um Servidor de DW Extensível

Eduardo Manuel de Freitas Jorge

Dissertação submetida à Coordenação do Programa de Pós-Graduação em Informática da Universidade Federal da Paraíba – Campus II, como parte dos requisitos necessários para a obtenção do grau de Mestre em Informática.

Área de Concentração: Banco de Dados

Marcus Costa Sampaio
(orientador)

Campina Grande, Paraíba, Brasil
Agosto de 2001

Ficha Catalográfica

JORGE, Eduardo Manuel de Freitas

J82A

Análise, Projeto, Implementação de um Servidor DW Extensível.

Dissertação (Mestrado) – UFPB/CCT/COPIN, Campina Grande, Agosto de 2001.

140p. Il.

Orientador: Marcus Costa Sampaio

1. Banco de Dados
2. Data Warehousing
3. Framework.

CDU: 681.3.07B

ANÁLISE, PROJETO E IMPLEMENTAÇÃO DE UM SERVIDOR DE DW EXTENSÍVEL

EDUARDO MANUEL DE FREITAS JORGE

DISSERTAÇÃO APROVADA EM 17.08.2001

PROF. MARCUS COSTA SAMPAIO, Dr.
Orientador

PROF. CLAUDIO DE SOUZA BAPTISTA, Ph.D
Examinador

PROF. DÉCIO FONSECA , Dr.
Examinador

CAMPINA GRANDE – PB

Dedico esta dissertação aos meus pais,
Eduardo e Lúcia, pelo seu amor e dedicação,
e ao meu amor, Camila, pela sua existência.

AGRADECIMENTOS

Em primeiro lugar agradeço aos meus pais, Eduardo e Lúcia; a minha irmã, Elisa; a minha sobrinha, Gabriela e a minha noiva, Camila.

Agradecimentos especiais aos revisores deste trabalho: a minha mãe, professora Lúcia Maria Dias de Freitas Jorge; a minha noiva, Camila Silva Pereira e ao meu orientador, Doutor Marcus Costa Sampaio.

Ao professor Doutor Marcus Costa Sampaio pelos ensinamentos e pela qualidade da orientação prestada durante todo o trabalho.

Ao professor Doutor Jacques Sauvé pelos conhecimentos transmitidos.

Aos meus companheiros de morada nesta etapa da minha vida: Alex, Antônio e Rodrigo.

A todos os funcionários da Copin, em especial, a Ana e a Vera pelo total apoio.

Aos meus colegas de mestrado: André, Alberto, Edeyson, Hilmer e Rodrigo pela eterna troca de conhecimentos durante o curso.

Ao diretor da Faculdade de Ciências Contábeis da UCSAL, Fernando Azevedo, pelo grande incentivo e confiança em mim depositados.

Por fim, a todos da família, mas, principalmente, a minha avó, Maria Angélica, pela sua alegria de viver e pelo exemplo de dedicação ao próximo.

RESUMO

Assistimos ao florescimento de diversas pesquisas na área de Sistema de Apoio à Decisão, sendo Data Warehousing a de maior repercussão, por ser uma solução completa, contemplando desde a coleta dos dados até a sua transformação em informação. Este trabalho apresenta a análise, projeto e implementação de um Servidor de Data Warehouse Extensível — S-DW-E. O S-DW-E é um framework para construção e uso de Data Warehouses, e tem como característica principal a extensibilidade dos seus metadados para contemplar o requisito de adaptar-se a qualquer tecnologia OLAP, seja ela ROLAP — Relacional, Objeto-relacional, ou Puramente Orientada a Objeto —, seja ela MOLAP.

ABSTRACT

Research in Decision Support Systems is flourishing, Data Warehousing being the one with the most repercussion, since it provides a complete solution, contemplating steps from data collection until its transformation in information. This work presents the analysis, design and implementation of the Extensible Data Warehouse Server – E-DW-S. E-DW-S is a framework for the construction and use of Data Warehouses and has as main characteristics the extensibility of its metadata repository to fulfill the requirement that it adapt itself to any OLAP technology, be it ROLAP – Relational, Object-Relational, or simply Object-Oriented – or be it MOLAP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API	—	Aplication Programming Interface
BD	—	Banco de Dados
DDL	—	Data Definition Language
DML	—	Data Manipulation Language
DTD	—	Document Type Description
DW	—	Data Warehouse
DWing	—	Data Warehousing
HTTP	—	Hipertext Transfer Protocol
JDBC	—	Java Database Connectiviy
MOLAP	—	Multidimensional OLAP
ODBC	—	Open Database Connectivity
OLAP	—	Online Analytic Processing
OLTP	—	Online Transaction Processing
OO	—	Orientação a Objetos
OR	—	Objeto-Relacional
RAD	—	Rapid Aplication Development
RMI	—	Remote Method Invocation
ROLAP	—	Relacional OLAP
SADs	—	Sistemas de Apoio à Decisão
S-DW-E	—	Servidor DW Extensível
SGBD	—	Sistema de Gerência de Banco de Dados

SGBDR	—	Sistemas de Gerência de Banco de Dados Relacionais
SGBDOO	—	Sistemas de Gerência de Banco de Dados Orientado a Objetos
SGBDOR	—	Sistemas de Gerência de Banco de Dados Objeto-Relacional
UDT	—	User Defined Type
UML	—	Unified Modeling Language
XML	—	Extensible Markup Language

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

No cenário atual de globalização, é de suma importância para as organizações sistemas de informação consistentes, rápidos e acessíveis.

Por outro lado, é comum a informação estar dispersa em diversos bancos de dados ou outras fontes de informação, o que dificulta o seu acesso integrado de forma veloz e consistente.

Esses ambientes de sistemas heterogêneos caracterizam-se por ser verdadeiras "ilhas de informação" [KLE99].

O problema das "ilhas de informação" tem um impacto menor em relação a sistemas rotineiros de produção — também chamados de sistemas operacionais —, em que a ausência de dados integrados é menos crítica. Nesses sistemas, a informação é detalhada, não histórica e departamentalizada, o que favorece mesmo a existência de "ilhas de informação".

Diferentemente, porém, das aplicações operacionais, aplicações ou sistemas gerenciais necessitam de informação menos detalhada e mais estratégica, implicando em dados integrados e históricos. Executar consultas gerenciais sobre sistemas operacionais é absolutamente contraproducente, pois o desempenho seria insuportavelmente baixo.

Sistemas gerenciais são apropriadamente chamados de Sistemas de Apoio à Decisão — SADs. Eles surgiram com a finalidade de suprir a demanda de informação integrada, histórica e consolidada, propiciando um ambiente conveniente à análise de dados, lidando sempre com um grande volume de dados.

Na verdade, SADs não são algo novo ou distante das empresas. Algum tipo de SAD já vinha sendo desenvolvido nas organizações, só que de forma específica e empírica.

1.1 Enfoque Data Warehousing para SADs

Assistimos ao florescimento de diversas pesquisas na área de SADs, sendo Data Warehousing — DWing — a de maior repercussão, por ser uma solução completa, abarcando desde a coleta dos dados até a sua transformação em informação. O coração de um sistema de DWing é um banco de dados especial chamado de Data Warehouse — DW. Um DW geralmente é centralizado e sintonizado para consultas de grande complexidade a dados integrados, envolvendo muitos dados.

Uma típica consulta a um DW é a emissão de uma série histórica de vendas de um determinado produto, com vistas à previsão do comportamento de vendas futuras do mesmo produto.

Ferramentas para o acesso a DW são conhecidas como ferramentas OLAP ("On-Line Analytic Processing") [KIM98]. Tais ferramentas fazem uso da infra-estrutura do DW para prover suporte necessário às consultas dos usuários. Podemos citar as duas principais tecnologias envolvidas na construção de ferramentas OLAP:

- ROLAP ("Relacional OLAP") - suporte a consultas OLAP por meio de Sistemas de Gerência de Bancos de Dados Relacionais — SGBDR;
- MOLAP ("Multidimensional OLAP") - suporte a consultas OLAP baseadas em arrays multidimensionais.

Os SGBDRs não atendem em alguns aspectos (estruturas de dados, funções, linguagem, etc) às aplicações que diferem das OLTP ditas não convencionais como Dwing. As

deficiências dos SGBDRs tradicionais, mais propriamente do modelo puramente relacional, são bem conhecidas [STO96]. Em conseqüência, os SGBDRs têm evoluído no sentido de fazer convergir a tecnologia relacional com o paradigma da orientação a objeto, resultando nos SGBDs objeto-relacionais — SGBDORs. SGBDORs são talhados para o suporte a tipos complexos de dados.

Paralelamente ao refinamento do modelo de dados relacional, os SGBDRs têm se tornado cada vez mais eficientes, tanto na capacidade de armazenar grandes volumes de dados, como no desempenho de consultas complexas. É permitido afirmar que a última geração de SGBDR, incluindo nela os SGBDORs, é um meio eficaz de implementação de sistemas de DWing.

No que diz respeito a SGBDs baseados em arrays multidimensionais, eles são, em princípio, mais eficientes que SGBDRs para o processamento de consultas OLAP. Contudo, a experiência tem mostrado que arrays multidimensionais são pouco flexíveis a mudanças, e têm problema de escala [KIM98]. Pode-se concluir que a tecnologia MOLAP é indicada apenas para sistemas de DWing com estruturas de dados estáveis, desde que o DW seja de pequeno porte.

1.2 Objetivos do Trabalho

O principal objetivo deste trabalho é a construção de um ambiente de desenvolvimento de sistemas de DWing, que possa utilizar qualquer tecnologia existente, seja ela ROLAP — relacional, objeto-relacional, ou puramente orientada a objeto —, seja ela MOLAP. Um outro requisito do ambiente de construção de DWing é que ele seja ainda flexível o suficiente para acomodar sistemas de DWing baseados em qualquer nova tecnologia que venha a aparecer. O ambiente é chamado de Servidor DW Extensível — S-DW-E. No S-DW-E, todos os DW têm uma interface OLAP comum, seja o DW construído com tecnologia ROLAP, ou MOLAP, ou outra tecnologia qualquer. A interface comum é fundada no mesmo modelo conceitual de DW.

Os ganhos advindos com a utilização do S-DW-E são vários. Enumeramos:

1. Uma empresa tem um sistema de DWing implementado com tecnologia MOLAP. Por diversas razões, ela quer implementar um outro sistema de DWing com tecnologia ROLAP. Com o S-DW-E, todo o projeto e o código da modelagem conceitual do DW é reaproveitado, com evidentes ganhos de produtividade.
2. Uma empresa deseja fazer "benchmarking" das tecnologias ROLAP e MOLAP utilizando uma mesma aplicação de DWing. Com o S-DW-E, esta tarefa fica enormemente facilitada.
3. Caso posteriormente surja uma nova e promissora tecnologia de Dwing pode-se estender o ambiente de forma suave. Com o S-DW-E, a implementação de um novo sistema de DWing, segundo a nova tecnologia, é acelerada pela reutilização de parte do projeto e do código de outras implementações de sistemas de DWing no S-DW-E.
4. Uma empresa adquiriu as tecnologias ROLAP, MOLAP e `outras´. Ela incentiva a não dependência de uma particular tecnologia. O S-DW-E vem para facilitar esta tarefa.

1.3 Relevância do Trabalho

A relevância do trabalho reside nos seguintes pontos:

- Até onde vai nosso conhecimento, não existe na literatura registro de esforço similar de construção de um servidor extensível de DWing;
- A construção de ferramentas OLAP que não sejam dependentes de uma particular tecnologia é uma necessidade premente;
- A inexistência de um modelo de gerência de metadados para sistemas de DWing que seja aceito verdadeiramente como padrão, e o fato de que o projeto de um gerente de metadados para um servidor extensível de DWing implica em novas pesquisas em metadados, levam ao concomitante desenvolvimento de um novo gerente de metadados para DWing.

1.4 Estrutura do Documento

Este documento está estruturado da seguinte forma. O Capítulo I é esta introdução. O Capítulo II apresenta uma visão geral sobre a abordagem DWing para a integração de dados.

O Capítulo III é sobre avanços em engenharia de software. O Capítulo IV descreve em detalhes os requisitos e a análise do S-DW-E. O Capítulo V é dedicado ao projeto e à implementação do S-DW-E. O Capítulo VI conclui a dissertação e discute algumas perspectivas do trabalho.

CAPÍTULO II

ABORDAGEM DATA WAREHOUSING DE INTEGRAÇÃO DE DADOS: UMA VISÃO GERAL

Neste capítulo, descreve-se a arquitetura genérica de sistemas de Data Warehousing — DWing — para a integração de dados de diversas fontes, possivelmente heterogêneas. Em seguida, discute-se metadados para DWing. Seguem-se seções sobre os temas: análise multidimensional — OLAP ("On-Line Analytic Processing"); álgebra OLAP; representação e implementação de DW; tecnologia objeto-relacional em ambientes de DWing.

2.1 Uma Arquitetura Básica para Sistemas de DWing

O enfoque DWing propõe uma solução completa, desde a coleta dos dados até a sua transformação em informação, para o problema da integração de dados de diversas fontes. Os dados integrados e consolidados ficam armazenados em bancos de dados especiais chamados de Data Warehouse (DW). Um sistema de DWing engloba:

- *Ferramentas de "Back-End", que fazem o processo de consistência e migração dos dados de fontes diversas pertencentes a sistemas operacionais OLTP;*
- *Processo de armazenamento de informações de forma integrada e agregada em um repositório de dados especial e centralizado chamado Data Warehouse (DW);*

- *Ferramentas de "Front-End" para consultas "ad-hoc" OLAP por parte de tomadores de decisão.*

A figura 2.1 ilustra de forma resumida uma arquitetura genérica de um sistema de DWing.

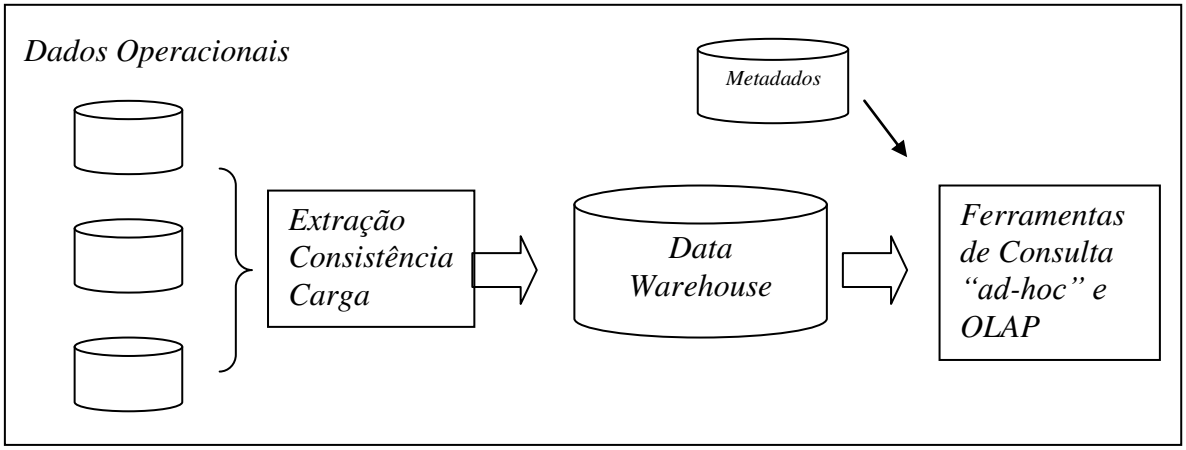


Figura 2.1: Arquitetura genérica de um sistema de DWing

Dados operacionais são fontes de informação, comumente BDs Operacionais Relacionais. Podem ser também Sistemas de Arquivos ou BDs Pré-Relacionais.

No módulo de Extração, Consistência e Carga está o Software de Aquisição de Dados ("back-end"), que extrai dados de fontes heterogêneas (Dados Operacionais), consolida-os, sumariza-os e carrega-os no DW. Este software pode ser muito complexo e é mais apropriadamente chamado de Sistema de Extração/Integração/Carga.

Servidor de DW é um software de gerência de DW, incluindo o repositório de metadados que descreve o DW. Veremos em detalhes o repositório de metadados e o DW nas duas seções seguintes.

Ferramentas de Consulta ("front-end") representam o conjunto de ferramentas para a interface entre o usuário final e o DW, permitindo consultas para a tomada de decisão.

2.2 Metadados para Sistemas de DWing

Os Metadados em um sistema podem ser entendidos como o conjunto de informações sobre a estrutura e significado dos dados, das aplicações e dos processos que manipulam os dados. Em bancos de dados relacionais, por exemplo, metadados descrevem esquemas de bancos de dados — definições de tabelas, colunas, visões, chaves primárias, chaves estrangeiras, etc.

Metadados desempenham um papel crucial em sistemas de DWing. Isto é devido à grande complexidade dos ambientes de DWing. Em conseqüência, é essencial a gerência cooperativa dos metadados por parte dos administradores, desenvolvedores e usuários de sistemas de DWing [STÖ99].

Os metadados para os bancos de dados são construídos pelo Sistema de Gerência de Bancos de Dados (SGBD). Eles são armazenados em um metabanco que é criado e mantido pelo próprio SGBD — Catálogo ou Dicionário de Dados. Os desenvolvedores de aplicações se servem do Catálogo para obter informações sobre esquemas de bancos de dados. Os administradores de bancos de dados também fazem uso do Catálogo para definir operações de controle de acesso e ajuste de desempenho dos bancos de dados, entre outras atividades próprias dos administradores. O SGBD precisa do Catálogo para poder compilar os programas de acesso aos bancos de dados. Podemos concluir que, se um único SGBD é utilizado, a manutenção de metadados — incluindo as consultas — é um problema bem resolvido pelo catálogo do SGBD.

A gerência de metadados é um problema muito mais complicado quando se trata de ambientes de DWing. Para tal ambiente, diversos SGBDs podem estar envolvidos, conforme a arquitetura apresentada na figura 2.1. Até mesmo `primitivos´ sistemas de arquivos (“legacy systems”) podem estar envolvidos. Um gerente de metadados agora precisa descrever globalmente o sistema de DWing. Listamos os principais tipos de metadados para DWing, não cobertos pelos catálogos tradicionais dos SGBDs:

- Informação sobre as fontes de dados do DW;
- Informação sobre como os dados são transformados, filtrados e pré-processados durante a extração e carga do DW;

- Histórico dos “jobs” administrativos (carga, agregação, backup);
- Regras para resolver conflitos de integração de dados;
- Informação sobre como foram feitas as agregações;
- Histórico da evolução do sistema de DWing (controle de versões);
- Esquema conceitual do DW.

Metadados podem ser classificados como:

- Metadados Técnicos (“Back room metadata” [KIM98])
- Metadados Semânticos (“Front room metadata” [KIM98])

Metadados Técnicos cobrem informação sobre:

1. Esquemas dos sistemas operacionais e do DW;
2. Esquemas lógico e físico do DW (p.e. ROLAP, MOLAP);
3. Dependências e mapeamentos entre as fontes operacionais e o DW, nos níveis lógico e físico;
4. Históricos dos dados e das ações dos diferentes tipos de usuário — administradores, desenvolvedores e usuários OLAP.

Os Metadados Técnicos são extraídos dos catálogos dos SGBDs, das bibliotecas dos ambientes de programação (“COBOL libraries”), das ferramentas de exportação/importação de dados. Além dos metadados, um software gerente de metadados técnicos deve suportar um modelo uniforme de representação dos diferentes tipos de metadados técnicos, bem como de navegação nos metadados. Típicos usuários de metadados técnicos são administradores e desenvolvedores de aplicação.

Metadados Semânticos cobrem os seguintes tipos de metadados:

1. Esquema conceitual do DW: dimensões (critérios de agregação de dados) e atributos de informações agregadas;
2. Dependências entre o esquema conceitual e o esquema lógico do DW.

Um software gerente de metadados semânticos deve ter ao menos duas funções: (f1) suporte à navegação ao longo da visão conceitual do DW; (f2) processamento de consultas *ad*

hoc no nível conceitual do DW. Típicos usuários de metadados semânticos são usuários OLAP e desenvolvedores de aplicação.

Por fim, metadados para DWing não tem um padrão de fato aceito [VET00]. As ferramentas de DWing disponíveis no mercado suportam o gerenciamento parcial dos metadados. Algumas ferramentas têm seu foco na extração, transformação e carga (Platinum Decision Base, Ardent DataStage, Eti Extract). Outras se atentam para consultas OLAP (MicroStrategy Dss Agent, Cognos Impromptu, Business Objects).

Temos ainda alguns relatos de tentativas de padronização de metadados [STÖ99], mas sem nenhum sucesso. As propostas de padrões existentes são diversas e incompletas. Assim, faz-se necessário um gerente de metadados, para ambientes de DWing, integrado e com um modelo uniforme.

2.3 Análise Multidimensional

Como foi afirmado na Introdução, o coração de um sistema de DWing é o DW. Nesta seção, trataremos exclusivamente de DW. Um DW deve obedecer a, no mínimo, dois requisitos:

- *Diferentes níveis (granularidades) de informação;*
- *Suporte a operações de análise multidimensional — operações OLAP.*

Numa típica seção OLAP, um usuário deseja obter informações (*fatos*) de várias perspectivas (*dimensões*), e em diferentes níveis. Por exemplo, o valor das vendas (*fatos*) pode ser analisado por filial, por produto e por mês (*dimensões*). Dentre as diversas dimensões num esquema multidimensional, deve-se enfatizar a dimensão tempo, já que esta sempre se faz presente em DW, pois informações analíticas são normalmente projetadas e analisadas comparativamente numa faixa de tempo, por exemplo, mês, trimestre, semestre, etc.

Estas considerações levam à definição de um DW como sendo um banco de dados multidimensional. Para ilustrar, uma planilha é um banco de dados bidimensional (figura 2.2).

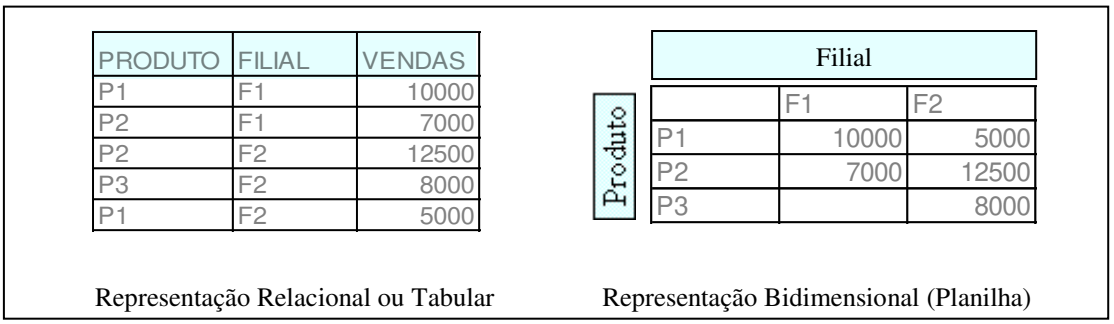


Figura 2.2 : Comparação entre a representação Relacional e a Bidimensional.

Infelizmente, planilhas são muito limitadas para DW: segundo [KIM98], é comum um DW ter até doze dimensões. Trata-se, genericamente, de um array multidimensional. No jargão técnico, um array multidimensional, no contexto de DW, é chamado de *cubo de dados*.

Um DW também é um banco de dados temporal, no sentido em que uma das dimensões é sempre o tempo.

Um problema imediato a resolver é a questão da representação de um cubo de dados, pois a partir de três dimensões há uma dificuldade de representação e definição das estruturas. Em termos de implementação (níveis lógico e físico), adota-se até o presente duas representações de cubo de dados: (r1) array multidimensional — MOLAP; (r2) esquema relacional em estrela — ROLAP.

2.3.1 Array Multidimensional — MOLAP

Cubos de dados são naturalmente implementados sob a forma de arrays multidimensionais — tecnologia MOLAP. Em um array, os índices (`colunas´) representam as dimensões de um negócio, enquanto que as suas células (`pontos´, `elementos´) contêm valores (fatos), geralmente numéricos e agregáveis, associados às dimensões. Com arrays, agregar/recuperar dados por `coluna´, ou por conjuntos de `colunas´ — serviços OLAP típicos — são operações naturais e extremamente eficientes.

Infelizmente, arrays não são a panacéia. Eles não são adequados a DW com grande volume de dados. O problema está no alto grau de `esparsidade´ dos cubos de dados reais. A título de ilustração, foi observado que um arquivo com 200 Mb de dados de processos

operacionais gera aproximadamente 5Gb de dados analíticos [SAM98]. Generalizando, estima-se que somente 10 a 15% das células de um array de um DW são realmente preenchidos.

Há muitos outros problemas com arrays. Serviços típicos dos SGBDRs, como segurança, manutenção de esquemas, linguagem declarativa padronizada de manipulação e definição de dados, otimização de consultas, tolerância a falhas, controle de acessos concorrentes, etc, são difíceis de implementar em arrays, sendo, em conseqüência, somente parcialmente atendidos. Também, um sem número de técnicas para tratar o problema da "esparsidade" não têm sido suficientes para viabilizar a tecnologia MOLAP para grandes DW. Finalmente, não tem havido acordo entre os proprietários de SGBDs baseados em arrays, no sentido da padronização dos serviços e interfaces.

Normalmente, grandes DWs têm sido implementados com tecnologia relacional (SGBDRs ROLAP) — assunto da próxima seção — enquanto pequenos e estáveis DW — Data Marts para fins de alguns tipos de consulta — construídos dos DWs relacionais são implementados com tecnologia MOLAP. É comum, um mesmo proprietário oferecer de forma integrada as duas soluções. Um exemplo é a Oracle, com o SGBDR Oracle8i para DW e a ferramenta Express (MOLAP) para a construção de Data Marts a partir de DW Oracle8i.

2.3.2 Simulação Relacional de Arrays Multidimensionais — ROLAP

A tecnologia relacional, dominando o mercado de SGBDs há duas décadas, atingiu uma grande maturidade, sem deixar de evoluir constantemente. Serviços como segurança, interface usuário-sistema padrão, tolerância a falhas e controle de concorrência são robustos e eficientes. O problema da `esparsidade´ inexiste: uma tabela relacional armazena unicamente `pontos´ reais de um array multidimensional (estritamente armazena o equivalente a `pontos´). A simulação relacional de arrays multidimensionais apresentada em [KIM96] é conhecida como *esquema em estrela* ("star schema").

Um esquema em estrela consiste de uma tabela central que armazena fatos e múltiplas chaves estrangeiras para relacionar a tabela de fatos com as tabelas que representam as dimensões. As tabelas de dimensão são geralmente de pequeno volume de dados[1] textuais.

[1] Quando comparado com o volume da tabela de fatos.

Tipicamente, usuários OLAP fazem "browsing" em tabelas de dimensão para escolher determinados valores, visando restringir o acesso à tabela de fatos. Uma das razões para que uma tabela de fatos seja normalmente volumosa é a existência da dimensão Tempo: é comum um fato permanecer on-line por 5, 10 anos. Um esquema em estrela é ilustrado na figura 2.3.

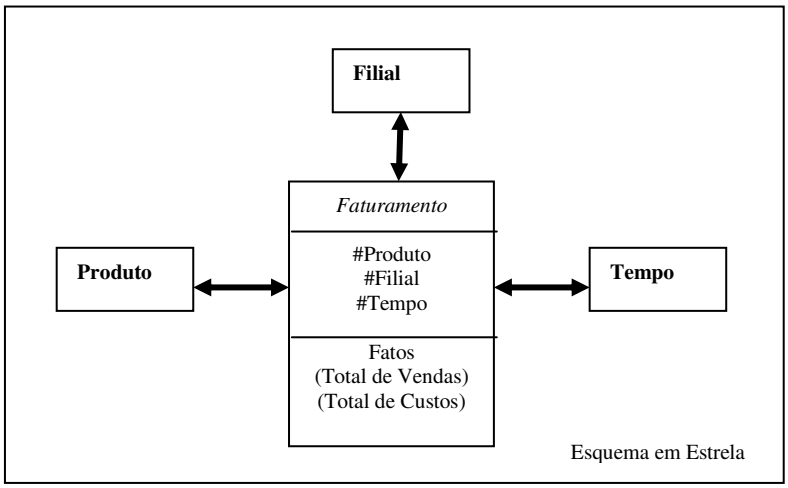


Figura 2.3: Exemplo da representação de um esquema em estrela

O esquema em estrela (figura 2.3) é composto por três tabelas: *Produto*, *Filial* e *Tempo*, que representam as dimensões do DW de Vendas. As três dimensões se relacionam com a tabela de fatos *Faturamento* através das chaves estrangeiras #Produto, #Filial e #Tempo. Os campos *Total de Vendas e de Custos* da tabela Faturamento contém os valores das vendas diárias de um produto numa filial e servem como medidas para operações de agregação.

2.4 Simulação Objeto-Relacional de Arrays Multidimensionais — OR-OLAP

Esta seção discute a viabilidade e conveniência do emprego da tecnologia Objeto-Relacional — OR no projeto do DW. A base da tecnologia OR-OLAP é a tecnologia ROLAP. Portanto, o esquema em estrela relacional continua sendo a solução para a simulação de arrays multidimensionais. Nesta seção, trataremos alguns pontos em que a tecnologia OR pode ser aplicada para criar um esquema em estrela.

2.4.1 A Tecnologia Objeto-Relacional

Embora os SGBDRs tenham sido um sucesso nos últimos 20 anos, eles são claramente insuficientes para suportar aplicações ditas não convencionais, como as aplicações "Web", aplicações multimídia, entre outras. Tais aplicações são suportadas por tecnologias orientadas a objeto.

Os SGBDRs têm evoluído no sentido de fazer convergir a tecnologia relacional para o paradigma da orientação a objeto, resultando nos SGBDOR. Os SGBDRs não atendem em alguns aspectos (estruturas de dados, funções, linguagem, etc) às aplicações que diferem das OLTP ditas não convencionais.

SGBDs OR são conhecidos como Servidores Universais. Eles têm como característica principal a possibilidade de extensão do banco de dados com a definição de novos tipos de dados e funções pelos desenvolvedores. Outros mecanismos como polimorfismo e encapsulamento também estão presentes, e baseiam-se na flexibilidade da gerência de tipos de dados proporcionados por esses sistemas [KLE99a].

A linguagem SQL com os comandos DML ("Data Manipulation Language") e DDL ("Data Definition Language") Objeto-Relacionais está sendo chamada de SQL-99.

Apesar dos SGBDORs lidarem com tipos não atômicos de dados, suas estruturas de dados ainda são baseadas em tabelas, mantendo assim, a compatibilidade com o SQL padrão (SQL-92).

Os principais fornecedores de SGBDs relacionais como Oracle, Informix, IBM já lançaram as novas versões dos seus produtos com características OR e compatibilidade parcial com o padrão SQL-99.

Citamos algumas considerações importantes em relação à tecnologia Objeto-Relacional:

- A tendência é que os SGBDs relacionais que não tiverem extensões OR perderão espaço no mercado de SGBDs.
- SGBDs OR são uma mudança menos `radical´ para as empresas do que SGBDs orientado a objetos.
- Os principais fornecedores de SGBDs Relacionais estão criando versões de seus produtos adicionando a tecnologia OR.
- A tecnologia OR permite solucionar problemas das aplicações chamadas de não convencionais ou complexas.

2.4.2 A Tecnologia Objeto-Relacional em DWing

Para aplicar a tecnologia OR na construção de esquemas DW, o esquema em estrela relacional `convencional´ é modificado, sendo-lhe adicionandas características de orientação a objeto.

Os fatos e as dimensões têm uma nova representação em relação ao esquema em estrela relacional. Muitos dos recursos disponíveis em banco de dados OR permitirão solucionar alguns problemas mal resolvidos no esquema em estrela convencional. Assim, é possível a criação de um esquema multidimensional alternativo ao esquema em estrela `convencional´ (soluções mais elegantes).

A proposta de criação de um esquema em estrela OR propõe um mapeamento das tabelas do esquema em estrela para classes [KLE99b]. No esquema em estrela `convencional´ as dimensões e os fatos têm uma relação m:n. Já no esquema em estrela Objeto-Relacional não existe mais esta diferença, pois tanto os fatos como as dimensões são objetos e relacionam-se através de associações. Essas associações não são, necessariamente, mapeadas convencionalmente como chaves estrangeiras. Por exemplo, poderão ser `mapeadas´ como referências, recursos que muitos gerenciadores OR (Oracle 8i, IBM) disponibilizam.

Relacionamos os pontos principais em que a tecnologia OR pode ser adotada para a construção de um esquema em estrela OR.

- Hierarquia de Dimensões

A representação das hierarquias de dimensões num esquema em estrela não é representada com clareza. Uma solução comumente adotada é a normalização das tabelas de dimensões, mas com o problema de dificultar a navegação dos usuários pelas dimensões (Snowflake).

Num esquema OR, uma hierarquia de dimensões é naturalmente representada. A dificuldade da navegação é removida, pois uma classe, herdando as propriedades de suas classes ancestrais, é `auto-contida´, ou tudo se passa como se todas as suas propriedades residissem nela.

- Hierarquia de Fatos

  A hierarquia de fatos acontece quando um fato pode ser representado de uma ou mais formas. Pode-se exemplificar, através de um fato de uma conta bancária. Este pode ser um lançamento numa conta corrente ou numa conta poupança, ou seja, uma situação clara de hierarquia de fatos.

  Quando é necessário fazer uma hierarquia de fatos, a solução adotada no esquema em estrela relacional é a junção entre duas tabelas de fatos. O problema é que para o modelo relacional junções entre tabelas de fatos é algo critico. Já num esquema OR pode-se fazer uso de generalização/especialização (herança) para solucionar este tipo de problema.

- Relacionamentos m:n entre Dimensões

  A solução padrão no esquema relacional para esta situação é a criação de uma nova tabela para relacionar as tabelas de dimensão. A desvantagem dessa abordagem é o aumento do número de tabelas no esquema criando, assim, um esquema maior e por conseqüência mais complexo.

  A solução OR pode implementar esse tipo de relacionamento como uma agregação através dos tipos coleção. O relacionamento passaria a ser tratado como um atributo não atômico das próprias dimensões.

- Uso de Objetos Complexos

  Com o recurso de UDT (“*User Defined Type”*) os elementos do esquema OR (dimensões, fatos e hierarquias) podem ser compostos com tipos definidos pelos usuários do banco de dados. Por exemplo, cria-se um tipo *Endereço*, que

contém todos os campos pertinentes ao endereço (Logradouro, Complemento, Bairro, etc). O tipo *Endereço* pode ser utilizado em diferentes dimensões do esquema OR.

- Incorporação de Métodos

  Consultas OLAP para serem resolvidas com SQL padrão exigem muito esforço e muitas vezes não possuem o desempenho adequado. O recurso de criação de funções *ad hoc* pelos usuários em SGBDOR possibilita resolver consultas OLAP complexas de forma mais simples. Além do mais, permite `encapsular´ operações nos elementos do esquema OR. Pode-se utilizar métodos personalizados em cláusulas `select´, `from´ e `where´. Por exemplo, *"... FROM Produtos_com_Vendas_Crescentes(1998, 2000) ..."*.

- Extensibilidade Além de UDTs

  A extensibilidade de um SGBDOR vai muito além de simples UDTs. Para ilustrar, novos tipos de índice — índices definidos pelos usuários — podem ser criados; sua incorporação ao SGBDOR é *stricto sensu*, o Otimizador de Consultas sendo devidamente instruído para reconhecer o novo tipo de índice, utilizando-o em situações em que ele claramente contribui para melhorar o desempenho das consultas. Os SGBDORs disponibilizam normalmente pacotes que lidam com esses tipos de dados — "Oracle Cartridge", "Informix Data Blade", etc. Novos pacotes podem ser dinamicamente construídos. Os pacotes são constituídos de um conjunto de classes com atributos e métodos para lidar com os novos tipos de dados.

É ainda prematuro afirmar que a tecnologia Objeto-Relacional (OR) aplicada a DWing traga necessariamente melhores resultados práticos do que a tecnologia Relacional. As possibilidades de sê-lo, no entanto, são potencialmente muito grandes, podendo-se apostar no rápido amadurecimento da tecnologia OR.

Na seqüência, serão abordadas algumas operações OLAP típicas. A discussão se dará no nível conceitual, isto é, não importando se a tecnologia subjacente é MOLAP, ROLAP ou OROLAP.

2.5 Uma Típica Sessão OLAP

No processo de análise de informação, os tomadores de decisão interagem com um DW, visando obter informações em vários níveis (granularidades), e sob diferentes dimensões.

Um executivo operando um SAD está sempre se perguntando *por quê ?* Ao fazer uma consulta, ele recebe informações analíticas que o deixam interessado em alguma coisa que necessita ser melhor observada.

Em tal processo, o executivo pode necessitar obter mais detalhes ("drill down") ou menos detalhes ("drill up") das informações para análise das tendências do seu negócio. Tendências são, geralmente, comparações de dados em um certo nível de sumarização através do tempo. O executivo `navega´ em diversos níveis de `granularidade´ do DW para descobrir as tendências do seu negócio.

Com base no esquema da figura 2.3, apresentamos um exemplo de uma consulta para análise dimensional. Considere a tabela 2.1 para a análise de um usuário executivo.

A tabela apresenta informações do total do mês e a comparação percentual com o mês anterior das vendas de um produto. Este relatório é uma sumarização dos dados do DW Vendas ─ mudança de granulariade dia para mês ("drill up").

Produto	Filial	Vendas no Mês	Compração com o Mês Anterior
Papel A4	Salvador	20.000	12%
Papel A4	***Campina Grande***	***3.000***	***-5%***
Total		23.000	7%

Tabela 2.1 : Relatório Comparação das Vendas Mensais[1].

Neste contexto, o executivo deseja saber a razão para os 5% negativos (Tabela 2.1). Assim, ele executa uma operação de "Drill down", acrescentando o atributo `Tamanho da

embalagem´ do produto. Esta operação em que o tipo do relatório é alterado é também chamada de “Slice and dice”.

Produto	Filial	Tamanho	Vendas no Mês	Comparação Com o Mês Anterior
Papel A4	Salvador	A	10.000	11%
Papel A4	Salvador	B	10.000	12%
Papel A4	Total		20.000	12%
Papel A4	Campina Grande	A	1.000	-4
Papel A4	Campina Grande	B	1000	-5
Papel A4	Campina Grande	C	1.000	-4
Papel A4	Total		3.000	-5 %
Total			23.000	7%

Tabela 2.2 : Relatório Comparação das Vendas Mensais[2].

Infelizmente, o resultado da tabela 2.2 não ajuda o executivo a diagnosticar o problema. Ele então executa sucessivamente “drill up” e um novo “drill down” com “slice and dice”. Agora ele observa o mesmo contexto sobre a variável `Equipe de Vendas´.

Produto	Filial	Equipe de Vendas	Vendas no Mês	Comparação Com o Mês Anterior
Papel A4	Salvador	X1	2.000	5%
Papel A4	Salvador	Y1	18.000	19%
Papel A4	Total		20.000	12%
Papel A4	Campina Grande	X2	2.000	4%
Papel A4	Campina Grande	Y2	800	1%
Papel A4	***Campina Grande***	***Z2***	***200***	***-20%***
Papel A4	Total		3.000	-5 %
Total			23.000	7%

Tabela 2.3 : Relatório Comparação das Vendas Mensais[3].

O executivo com o resultado da tabela 2.3 finalmente descobre uma notável inconsistência em suas equipes de venda, podendo agora tomar providências realmente efetivas.

O que se observa é que algumas operações em DW são comuns em sistemas OLAP. Operações como:

- *"Slicing-dicing" seleção de informações.*
- *"Drill down" mais detalhes de informações.*
- *"Drill up" menos detalhes de informações.*
- *"Drill across" combinação de informações.*

A próxima seção apresenta um modelo de dados multidimensional, conceitual e formal para consultas OLAP. Ele é fundado sobre um conjunto de operações multidimensionais primitivas, sobre as quais as implementações das operações OLAP "drill up", "drill down", "drill across" e "slicing-dicing" são grandemente facilitadas.

2.5.1 Um Modelo Multidimensional Conceitual e Formal

A principal razão do sucesso dos SGBDRs foi o desenvolvimento da álgebra relacional, propiciadora da linguagem SQL, de alto poder de expressão para consultas OLTP [ULL97]. De maneira análoga, faz-se necessária uma álgebra multidimensional como base para o desenvolvimento de consultas OLAP. O trabalho de R. Agrawal [AGR99] é pioneiro neste campo. Segue-se uma descrição detalhada desse modelo.

A estrutura de dados do modelo consiste de um ou mais cubos de dados. Um *cubo de dados* é exemplificado na figura 2.4.

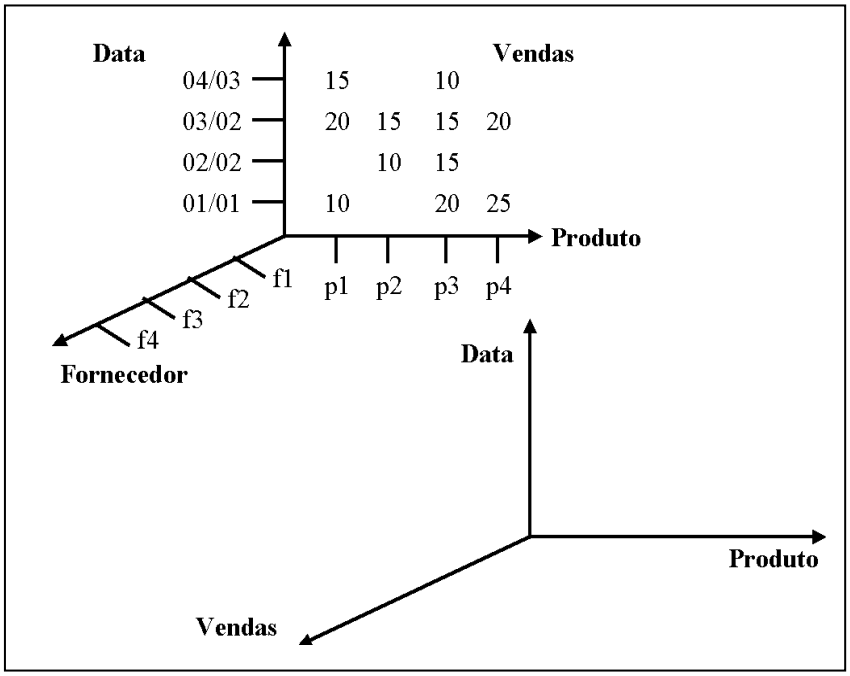


Figura 2.4: Exemplo de um Cubo de Dado.

Os componentes que definem formalmente um *cubo de dados* são:

- *K dimensões, para cada dimensão um nome $D_{i,}$ e um domínio $dom_{i.}$*
- *Elementos definidos como E(C) de $dom_1x..xdom_K$ igual a uma n-tupla, ou "0", ou "1". As n-tuplas representam os elementos dos fatos. No caso de uma tabela de fatos sem elementos serão representados por "1", ou "0" conforme a existência de $dom_1x..xdom_K$ ou não .*
- *Dimensões e medidas tratadas simetricamente. A figura 2.4 evidencia que Vendas tanto pode ser um fato como uma dimensão. No caso de ser uma dimensão, os `fatos´ são representados por "1" ou "0".*

No modelo relacional as operações de projeção, união e junção sobre tabelas resultam sempre em uma tabela. Fazendo analogia com a álgebra relacional, no modelo multidimensional operadores operam sobre um cubo de dados, resultando em um outro cubo de dados. Um desses operadores é *Restrict* Uma operação Restrict é representada na figura 2.5.

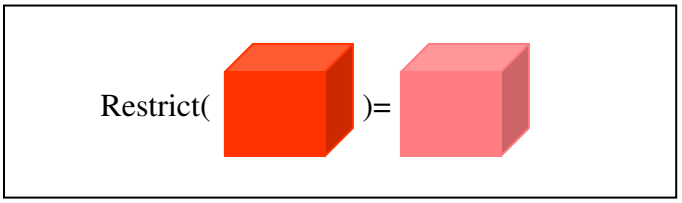


Figura 2.5: Operação em um Cubo de Dados.

Na figura 2.5, a operação Restrict recebe o cubo vermelho com entrada processa um filtro e retorna um novo cubo (rosa). As operações primitivas do modelo são: *Push, Pull, Restrict, Destroy, Join e Merge*. Seguem-se os detalhes dessas primitivas.

2.5.2 Operadores Multidimensionais Básicos

No trabalho de R. Agrawal [AGR99] são descritos os operadores multidimensionais básicos (*Push, Pull, Restrict*, *Destroy*, *Merge, Join*), que possibilitam resolver consultas OLAP, desde as mais simples até as mais complexas. Típicos exemplos de consultas OLAP complexas envolvem *Slicing-dicing*, *Drill down*, *Drill up*, *Drill across*. Com os operadores de R. Agrawal, consultas complexas podem ser decompostas em uma seqüência de operações básicas envolvendo os operadores multidimensionais.

1 PUSH

A operação Push é usada para converter dimensões em elementos que podem então ser manipulados usando a função f_{elem}, que será definida quando da descrição do operador *Merge.* Este operador é necessário para permitir que dimensões e medidas sejam tratados uniformemente. A figura 2.6 exemplifica este operador.

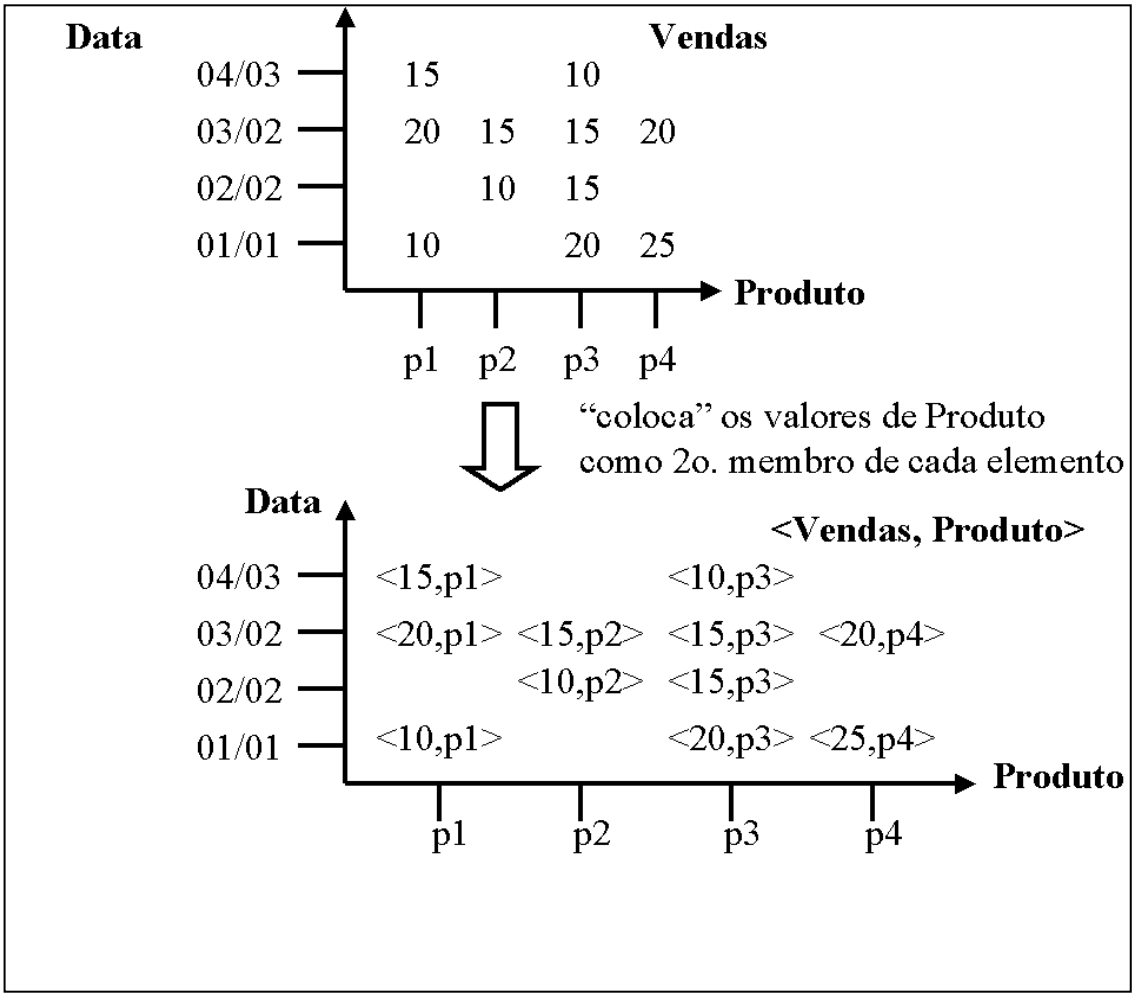


Figura 2.6: A Operação Push sobre a Dimensão Produto

- *Entrada: um cubo de k dimensões (C), e uma dimensão D_i.*
- *Saída: C com cada elemento $\neq 0$ estendido por um membro adicional, o valor da dimensão D_i para aquele elemento. Se o elemento não é uma n-tupla, mas um "1", então ele é convertido para uma 1-tupla que contém o correspondente valor da dimensão.*
- *Matematicamente: push(C, D_i) = C_{ans}.*

  $E(C_{ans})(d_1, ..., d_k) = g \oplus \langle d_i \rangle$ onde $g = E(C_{ans})(d_1, ..., d_k)$. O operador $\oplus$ é definido para resultar "0" se g = "0", é $\langle d_i \rangle$ se g = "1", e em todos os outros casos ele concatena as duas tuplas.

2 PULL

Esta operação é a inversa do operador Push. Pull cria uma nova dimensão para um membro especificado de cada elemento do cubo. O operador Pull é útil para converter elementos em dimensões, e assim, serem usados em operações de agregação e junção. Ele permite também simetria de dimensões e medidas. A Figura 2.7 exemplifica este operador.

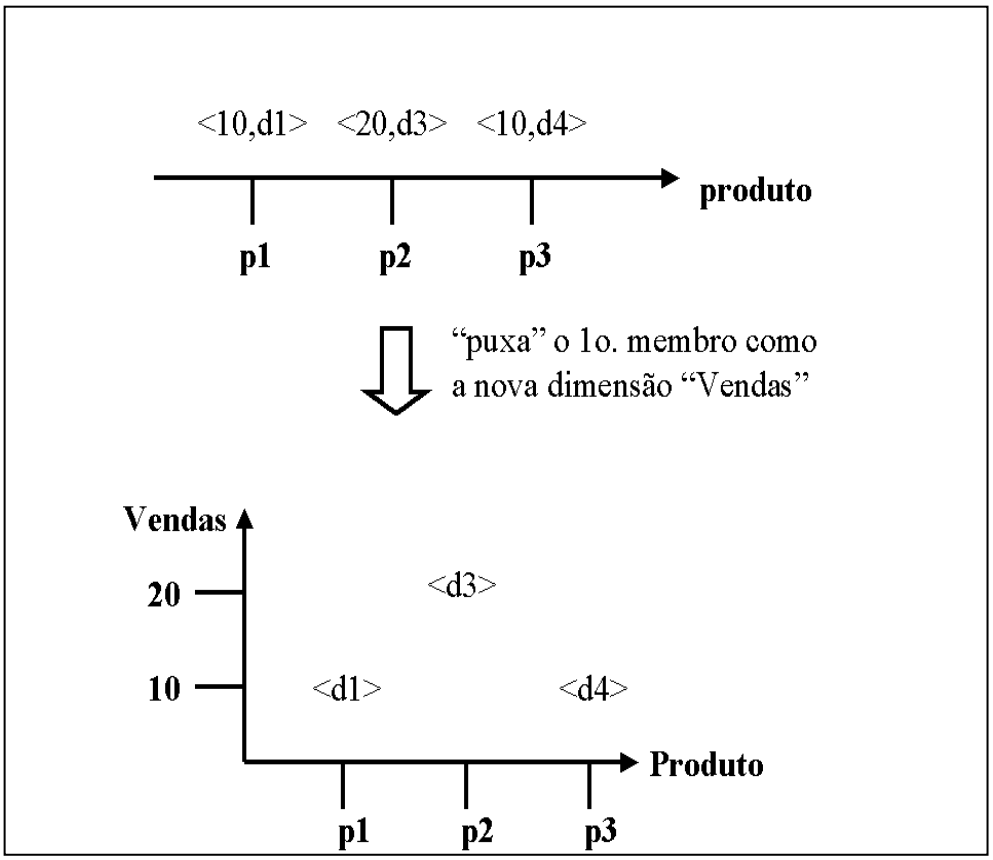


Figura 2.7: A Operação Pull que Cria a Dimensão Vendas

- *Entrada: C, a nova dimensão D, inteiro i.*
- *Saída: C_{ans} com uma dimensão adicional D que é obtida "puxando" o $i^{ésimo}$ membro de cada elemento do cubo.*
- *Restrição: todos os elementos ≠ "0" de C devem ser n-tuplas.*
- *Matematicamente: pull(C, D, i) = C_{ans}, 1 <= i <= n.*

  D torna-se a $k+1^{ésima}$ dimensão do cubo.

  $dom_{k+1}(C_{ans})$ = {e /e é o $i^{ésimo}$ membro de algum elemento de $E(C)(d_1, ..., d_k)$}.

$E(C_{ans})(d_1, ..., d_k, e_i) = <e_1, ..., e_{i-1}, e_{i+1}, ..., e_n>$ *se* $E(C)(d_1, ..., d_k) = <e_1, ..., e_i, ..., e_n>$, *senão* $E(C_{ans})(d_1, ..., d_k, e_i) = 0$.

3 *DESTROY*

É usado sempre que for necessário reduzir o número de dimensões. O operador remove a dimensão D_i que tem no seu domínio um único valor.

- *Entrada: C, dimensão* D_i.
- *Saída:* C_{ans} *com a dimensão* D_i *ausente.*
- *Restrição:* D_i *ter um único valor.*
- *Matematicamente: destroy(C,* D_i*)* C_{ans} *tem k-1 dimensões.*

4 RESTRICT

O operador *Restrict* opera sobre uma dimensão de um cubo, removendo os valores da dimensão que não satisfazem a condição estabelecida. A figura 2.8 ilustra uma operação Restrict. Note que este operador realiza a operação "*slicing/dicing*" da terminologia OLAP.

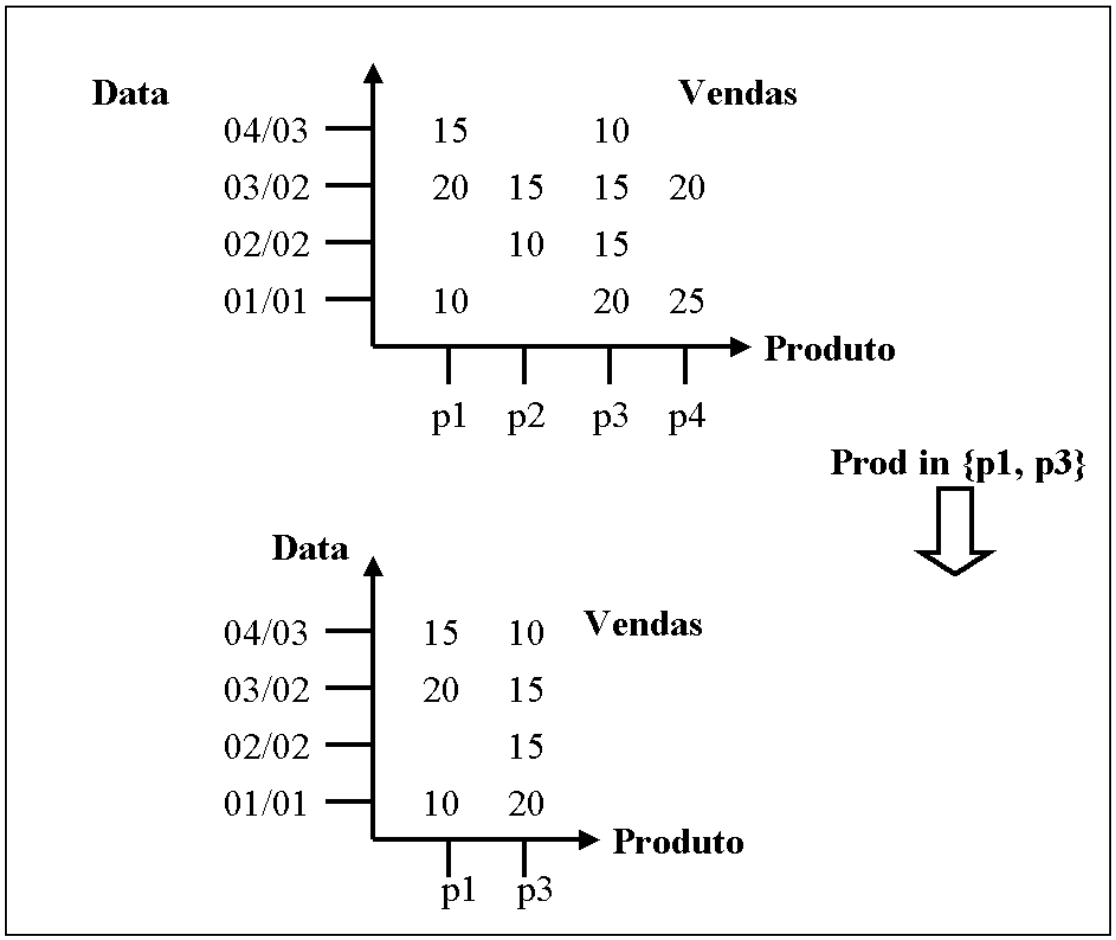


Figura 2.8: A Operação Restrict

- *Entrada: Cubo C e predicado P definido em D_i.*
- *Saída: Novo cubo C_{ans} obtido por remover de C aqueles valores da dimensão D_i que não satisfazem o predicado P. Se nenhum elemento da dimensão D_i satisfaz P então C_{ans} é vazio.*
- *Matematicamente: restrict(C, D_i , P) = C_{ans}.*
  $dom_j(C_{ans})$ (domínio da dimensão j de C_{ans}) = $dom_j(C)$ se 1 <= j <= k & j ≠ i senão $dom_j(C_{ans}) = P(dom_j(C))$.

5 JOIN

A operação *Join* é usada para relacionar informações entre dois cubos. O resultado de juntar um cubo *m*-dimensional C com um cubo *n*-dimensional C1 sobre *k* dimensões, chamadas *dimensões de junção* ("joining dimensions"), é um cubo C_{ans} com *m+n-k*

dimensões. Cada dimensão de junção D_i de C combina com exatamente uma dimensão D_{xi} de C1; a correspondente dimensão resultante terá valores que são a união dos valores de D_i e de D_{xi} (transformações podem, eventualmente, ser aplicadas aos valores de D_i e D_{xi}). Os valores de C_{ans} podem ser obtidos por alguma função f_{elem} aos elementos pertinentes de C e C1. Mais detalhes consultar [AGR 99].

6 MERGE

A operação *"merge"* é uma operação de agregação. A figura 2.9 mostra como hierarquias são implementadas usando o operador *"merge"*. Múltiplos membros de elementos sobre cada dimensão são "merged" para produzir uma dimensão com um domínio possivelmente menor (*"drill up"*). A função de agregação é especificada de maneira *ad hoc*.

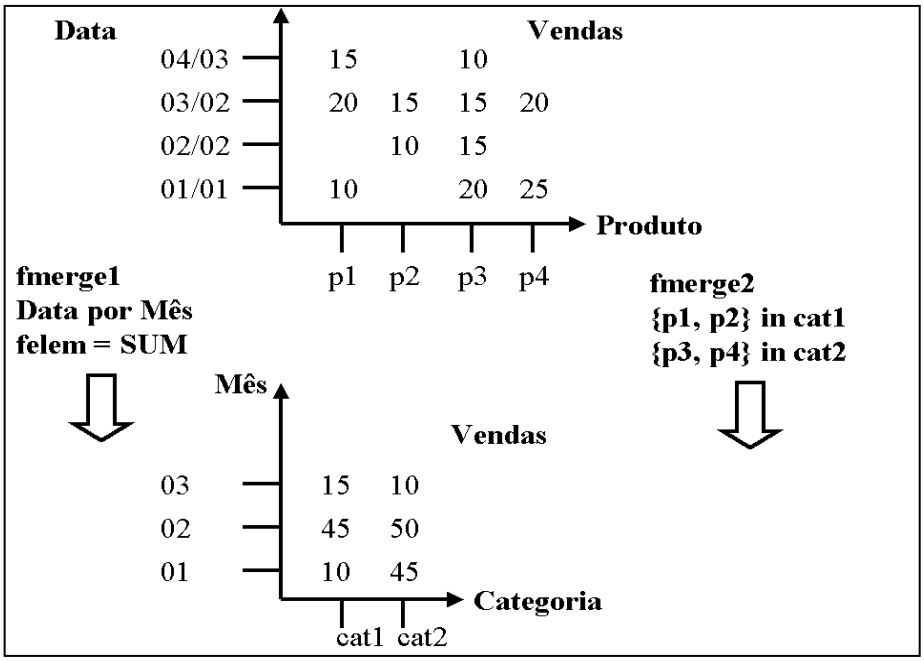


Figura 2.9: Merge das Dimensões Data e Produto, Usando f_{elem} = *sum*

- *Entrada: C, função de agregação f_{elem} e m pares (dimensão, f_{merge})*
- *Saída: Cubo C_{ans} da mesma dimensionalidade de C. A dimensão D_i de C_{ans} substitui a dimensão D_i de C, via uma função f_{merge}. Um elemento correspondendo a um valor de D_i de C_{ans} é obtido de elementos correspondentes de C, via f_{elem}.*
- *Matematicamente: merge(C, {[D_1, f_{merge1}], ..., [D_m, f_{mergem}]}, f_{elem}) = C_{ans}.*

$dom_i(C_{ans}) = \{f_{mergei}(e) \mid e \in dom_i(C)\}$ *se* $1 <= i <= m$, *senão* $dom_i(C_{ans}) = dom_i(C)$.
$E(C_{ans})(d_1, ..., d_k) = f_{elem}(\{t \mid t = E(C)(d'_1, ..., d'_k)$ *em que* $f_{mergei}(d'_i) = d_i$ *se* $1 <= i <= m$, *senão* $d'_i = d_i$.

2.5.3 Exemplo de Resolução de Consultas OLAP

Esta seção apresenta a resolução de algumas consultas OLAP, utilizando operadores de R.Agrawal. As consultas são efetuadas sobre um típico DW de Vendas (para supermercado) apresentado na figura 2.10. As dimensões Produto, Categoria (grupos de produtos), Loja, Dia, MesAno e Ano fazem parte do DW (perspectivas do negócio). O DW Vendas possui ainda um *cubo de dados* C_{Vendas} associado com as dimensões Loja, Dia, Produto. Os elementos do *cubo de dados (vendas)* são as vendas efetuadas de um produto, em um dia e em uma loja (fatos). Seguem duas consultas e as suas resoluções.

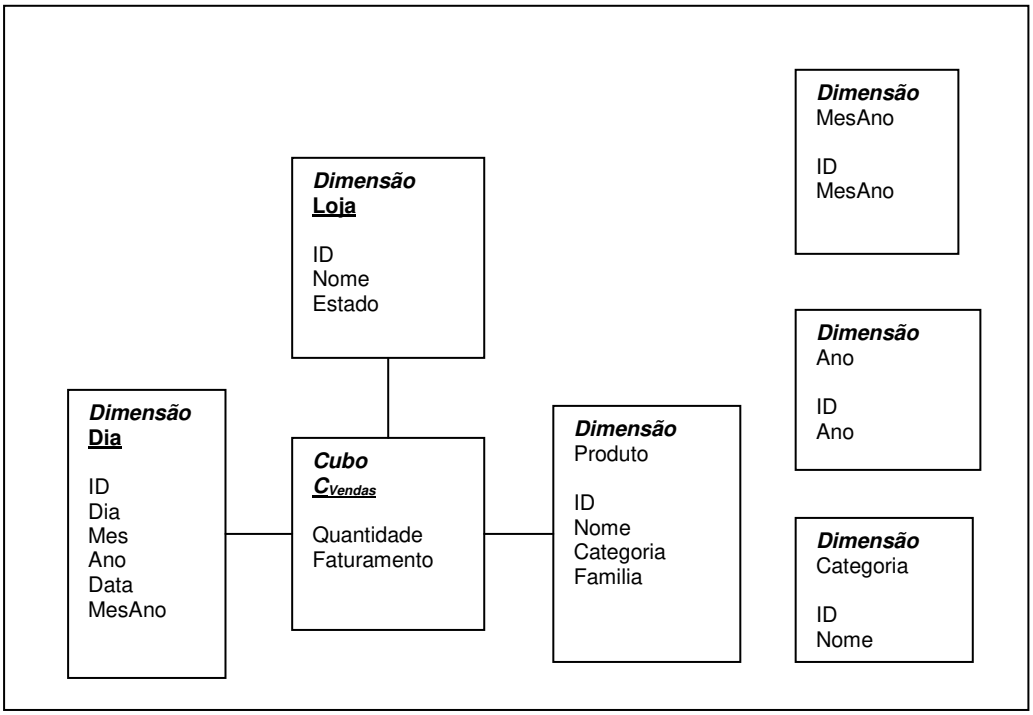


Figura 2.10: Esquema Conceitual de um DW de Vendas.

- *"O total das vendas anuais das categorias relacionadas com produto P1";*
  - $C_1 = Restrict(C_{Vendas,}\ Produto = P1)$
  - $C_2 = Destroy(C_1, Loja)$
  - $C_3 = Merge\ (C_2,\{[Dia\ por\ Ano],[Produto\ por\ Categori]\},Sum)$

- *Resposta Cubo de Dados C_3*

- *"O crescimento das vendas mensais do 01-2000 em relação à 01-1999 do produto P1";*
  - *C_1 = Restrict($C_{Vendas,}$ Produto =P1)*
  - *C_2 = Merge(C_1,{[Dia por MesAno]},Sum)*
  - *C_3= Restrict(C_2,MesAno={01/1999,02/2000})*
  - *C_4= Destroy(C_3, Loja)*
  - *C_5=Merge(C_4,[{MesAno por CrescimentoP}],$F1_{elem}$)*
    - *$F1_{elem}$ = (X-Y)/Y, onde X = Total das vendas do produto P1 no mês 01-2001 e Y = Total das Vendas do Produto P1 no mês 01-1999*
  - *Resposta Cubo de Dados C_5*

CAPÍTULO III

AVANÇOS EM ENGENHARIA DE SOFTWARE

Reutilização de software é o grande desafio dos projetistas de software. A engenharia de software vem evoluindo substancialmente para ir ao encontro a essa necessidade. Dentre as diversas metodologias existentes, Orientação a Objetos (OO) é o principal paradigma para a construção de software reutilizável. Apesar do grande sucesso do paradigma OO, nem sempre os resultados, em relação a reutilização, são obtidos de forma fácil.

Este capítulo se inicia apresentando duas técnicas de reutilização de software: Framework e Padrões de Projeto. Segue-se uma seção sobre um processo de desenvolvimento de software para aplicações do tipo framework baseado no Processo Unificado [JAC98]. Por fim, resumimos o que foi discutido no Capítulo II e III e sua aplicabilidade no projeto do S-DW-E.

3.1 Considerações Sobre Reutilização de Software

Nesta seção, aborda-se o tema *reutilização*. O objetivo é discutir os vários níveis — análise, projeto ou código — de reutilização. Em seguida, faz-se o estudo de dois tipos de reutilização: Framework Orientado a Objeto e Padrões de Projeto.

3.1.1 Motivação para a Reutilização de Software

O desenvolvimento de projetos de software não é uma tarefa fácil. Os projetos, geralmente, têm que ser robustos, atuar sobre problemas complexos e estar implantados em curto prazo (“time-to-market”).

A reutilização é reconhecida como um importante modo para se alcançar um aumento na produtividade em projetos de software, pois possibilita agregar funcionalidades pré-existentes na produção de novos software. Ressalte-se também a possibilidade de programadores iniciantes criarem software complexos através da utilização de padrões de projetos descritos por programadores mais experientes.

Além do mais, a reutilização de pedaços de software garante uma maior qualidade ao projeto, visto que os blocos a serem reutilizados já estão testados e validados por uma ou mais aplicações. Portanto, o direcionamento no desenvolvimento de um projeto de software é para que não se construa nada que já exista e que possa ser reutilizado.

Entretanto, é necessário produzir software genéricos e extensíveis que possam ser aplicados a uma gama de aplicações. Ao longo dos anos, muitas metodologias surgiram na engenharia de software com o intuito de reduzir a complexidade e aumentar a produtividade no desenvolvimento. A Orientação a Objetos é reconhecida hoje como o principal paradigma para atender a essa redução da complexidade.

3.1.2 Reutilização e Orientação a Objeto

As vantagens da Orientação a Objetos, como paradigma para a reutilização de software, são bem conhecidas [MAR 95].

- As abstrações podem corresponder às coisas do domínio do problema
    - O nível é mais natural.
    - É mais fácil comunicar-se com o usuário ou “domain expert” na linguagem dele.

- Os mesmos objetos existem em todas as fases e uma notação única (objetos) facilita portanto a integração entre fases de desenvolvimento (passar de uma fase para a outra).
- É mais fácil entender o domínio do problema quando este é quebrado em pedaços: gerenciamento da complexidade através da modularização.
- O mesmo pode ser dito no domínio do computador (projetando e programando com objetos).
- A abstração controla a complexidade (escondendo algo através da separação da interface e da implementação).
- A encapsulação facilita as mudanças (através do isolamento).
- A hierarquia (grafo de herança) permite uma melhor reutilização.

Um dos mecanismos da Orientação a Objetos para alcançar a reutilização de código é o uso de biblioteca de classes que encapsulam determinadas funcionalidades que possam ser reutilizadas. Embora classes sejam unidades de código reutilizáveis, uma classe para cumprir sua funcionalidade, geralmente, tem dependências com outras classes. Para a realização de tarefas complexas é necessário que um grupo de classes trabalhem conjuntamente, caracterizando assim, um componente de software.

O conceito de componente de software pode ser entendido como uma composição de classes que juntas formam uma unidade para cumprir certas responsabilidades. Componentes devem possuir uma interface pública para a comunicação com os blocos de código externos. Eles têm como principal característica o funcionamento como `*caixas pretas'*, onde o desenvolvedor não precisa, necessariamente, conhecer os detalhes de sua implementação.

Através do uso de componentes, a reutilização popularizou-se. Um exemplo clássico são as ferramentas RAD ("Rapid Aplication Development") gráficas (Visual Basic, Delphi, Centura, etc), que fornecem componentes visuais aplicáveis nas construções de interfaces gráficas. Os componentes visuais (VBX, OCX, etc) disponíveis nessas ferramentas proporcionam um aumento na produtividade de desenvolvimento de sistemas comerciais.

Além da reutilização de código, existem outros tipos de reutilização. Por exemplo, a reutilização de projeto atua não somente no patamar de código, como também nos projetos conceitual e lógico.

A questão que se coloca é:

> *Como `encapsular', em bibliotecas ou em componentes, comportamentos genéricos para um domínio de aplicação[1], que possam ser usados em software de mesmo domínio[2], só que com características específicas?*

A próxima seção trata de Framework, uma técnica de desenvolvimento de software que permite a construção de projetos reutilizáveis em todos os níveis.

3.2 Framework Orientado a Objetos

A reutilização em um projeto de sistemas deve contemplar todos os níveis de um projeto e não se restringir somente à codificação, ou seja, deve englobar análise, projeto, codificação e testes.

Framework provê reutilização em alto nível. Portanto, através da técnica de projeto de software framework, é possível construir projetos extensíveis, em que a reutilização está focada desde o domínio da aplicação.

É importante destacar que framework não é alternativa à Orientação a Objetos ou a componentes, mas sim, trata-se de um recurso adicional de projeto para reutilizar algo mais do que código.

Na verdade, é difícil imaginar a construção de um framework que não seja OO. A Orientação a Objetos possui características como abstração, `encapsulamento´, herança e colaboração de classes que viabilizam a sua criação. Da mesma forma, um framework é construído com base em componentes.

---

[1] Contexto em que uma aplicação está inserida

Na seqüência, tratamos o tema Framework com mais detalhes, envolvendo o conceito, as diferenças entre framework e bibliotecas de classes, o projeto de um framework, e a construção de aplicações baseadas em frameworks. Por fim, discutimos a importância do tema.

3.2.1 Conceito de Framework

Na literatura existente são várias as definições de framework. Nesta seção, vamos enfatizar as duas que, na nossa opinião, representam melhor o conceito de framework.

Iniciamos com uma definição genérica: *"Framework na sua essência pode ser definido como um conjunto de blocos de software que os programadores podem usar, estender, ou `customizar´, com pouco esforço, para um domínio específico de problema."* [IBM 99].

Já uma definição mais técnica determina que "*Um framework é um conjunto de classes colaborativas abstratas e concretas, que pode ser usado como um `template' (gabarito) para resolver uma família de problemas relacionados. Ele é usualmente estendido através de subclasses, para obter-se o comportamento específico de uma aplicação.*" [LAR00]

É importante enfatizar que existe uma diferença substancial entre framework e biblioteca de classes.

3.2.2 Framework Versus Biblioteca de Classes

A compreensão do conceito de framework relaciona-se com o entendimento de sua diferença com biblioteca de classes. A confusão entre os dois conceitos é uma incompreensão clássica entre desenvolvedores de software.

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

[2] Uma fronteira formal que define um particular assunto ou área de interesse [LAR00].

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.

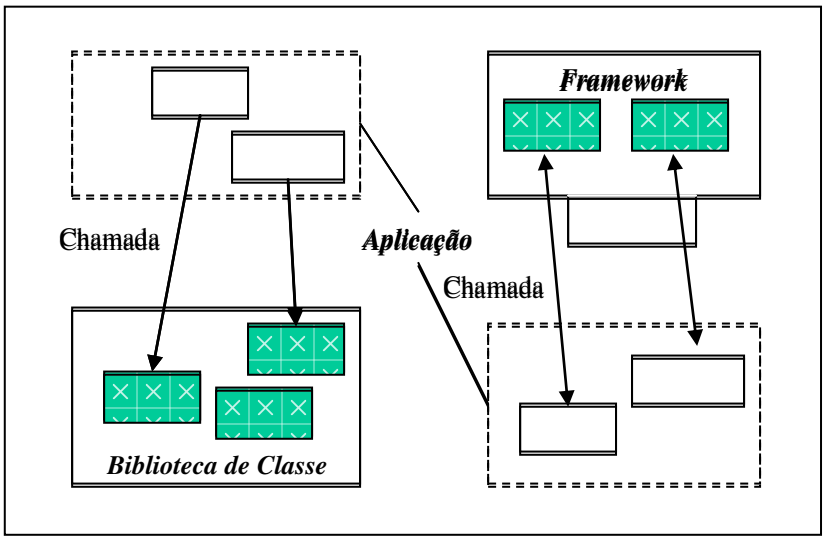


Figura 3.1: Diferença no fluxo de controle entre Framework e bibliotecas de classes [LAN95].

A figura 3.1 ilustra duas aplicações. A primeira aplicação possui classes que fazem chamadas às classes pertencentes a uma biblioteca de classes. A segunda aplicação foi criada a partir de um Framework. Neste caso, são as classes do Framework que fazem as chamadas às classes da aplicação.

A construção de um framework não é uma tarefa trivial. Passemos a discuti-la.

3.2.3 Framework e Reutilização de Software

Ao projetar um Framework, é imprescindível levar em conta que sua estrutura deve servir para a construção de diversos sistemas, que pertencem a um mesmo domínio de aplicação. Este projeto genérico deve capturar e `encapsular´ todas as funcionalidades comuns às futuras aplicações que serão desenvolvidas, como ilustra a figura 3.2.

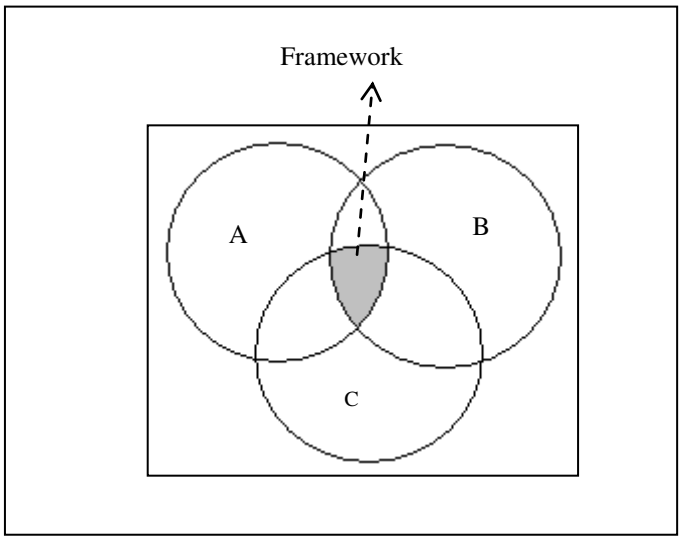


Figura 3.2: Interseção de três aplicações.

A figura 3.2 sugere três aplicações A, B e C, representadas por círculos. Estas aplicações possuem funcionalidades específicas e funcionalidade comuns. As funcionalidades comuns — framework — estão representadas no gráfico pela área hachurada que é a interseção dos três círculos. Quanto maior a interseção entre as aplicações, mais funcionalidades poderão estar `encapsuladas' no framework.

Um framework deve ser projetado para ser usado através do princípio "Hollyood Principle", isto é, "Não nos chame, nós iremos chamar você". As classes desenvolvidas nas aplicações irão receber mensagens das classes do framework (figura 3.2). Assim, é possível que as classes definidas nas aplicações façam o resto do trabalho específico que o framework deixou em aberto.

Ainda ao projetar-se um framework, deve-se construir uma aplicação quase que completamente, faltando apenas `pedaços´ para serem completados com as funcionalidades específicas das futuras aplicações. Mais precisamente, a arquitetura do framework deve ser tal que possibilitará às aplicações que vierem a ser construídas utilizar a sua infra-estrutura. A figura 3.3 dá uma idéia disto. A aplicação é a `soma´ do framework e do `bloco´ com as extensões que `encapsulam' as funcionalidades específicas.

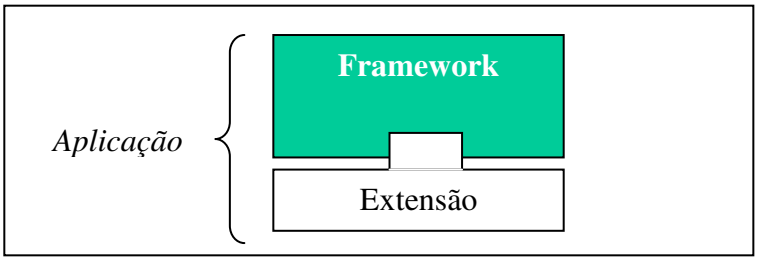


Figura 3.3: Aplicação construída utilizando a infra-estrutura de um Framework

Pode-se observar também na figura 3.3 que o `bloco´ do framework é `maior´ que o `bloco´ da extensão, sugerindo que foi alcançado um alto grau de reutilização, o que é desejável.

Por fim, um framework deve ser tratado como um produto que necessita de documentação e suporte. Como produto ele deve ser planejando para ser distribuído e sofrer manutenções constantes.

3.2.4 Construindo Aplicações a partir de um Framework

Para construir aplicações a partir de um framework o desenvolvedor, preliminarmente, deve:

1. Verificar se o framework atende às necessidades da aplicação.
2. Analisar o grau de complexidade para incorporá-lo à aplicação que será gerada.
3. Distinguir qual é a camada (subsistema) que é fixa, ou seja, o framework, e qual é a camada variável (o subsistema que será desenvolvido).

Fica óbvio então que, para começar, o framework a ser reutilizado deve ser muito bem projetado, e dispor de uma excelente documentação. A documentação deve descrever em detalhes os passos a ser seguidos para a construção de uma aplicação através do framework, incluindo:

- que classes devem ser estendidas;
- que métodos devem ser implementados;
- que novas classes devem ser criadas.

Na ótica do usuário `implementador´ (aquele que vai estender o framework) o desenvolvimento da aplicação, na prática, implica numa `instanciação´ do framework. Por conseguinte, o desenvolvedor terá de estender por herança algumas classes abstratas ou utilizar composição em outras. Como foi dito antes, ele deve utilizar-se da documentação que detalha o que é necessário para estender o Framework. É como utilizar `ganchos´ existentes no framework, que permitem `plugar´ funcionalidades. Tratam-se de blocos de código que serão adicionados ao framework, para que o sistema fique habilitado a executar funcionalidades específicas da aplicação.

Ressalta-se que, ao desenvolver uma aplicação a partir de um framework, está-se na verdade reutilizando o projeto do framework como um todo, incluindo os níveis conceitual, lógico e físico.

3.2.5 Framework `A Bola da Vez´

O conceito de Framework não é algo recente e desconhecido na engenharia de software. Desde do final da década de 80, muitas pesquisas nesta área já tinham sido realizadas [WEI88] [TAL93].

Então, porque só mais recentemente o conceito de framework tem sido largamente aceito? Isto se deve:

1. À maturidade das linguagens orientadas a objeto (Java, C++, etc);
2. À demanda por projetos de software mais complexos e reutilizáveis;
3. Aos novos recursos da engenharia de software como padrões de projeto [GAM94], processo interativo incremental [JAC98] , UML [LAR00], etc.

Dentre os recursos citados, padrões de projeto têm uma importância significativa e podem ser de grande valia para os desenvolvedores de frameworks. A próxima seção trata de padrões de projeto, com o foco em framework.

3.3 Padrões de Projeto

O bom desenvolvimento de um software orientado a objeto depende de alguns fatores:

1. Uma boa decomposição do sistema em classes e objetos
2. O conhecimento pelos desenvolvedores dos diversos aspectos ligados à orientação a Objeto:
   - Encapsulação
   - Reutilização
   - Flexibilidade
   - Portabilidade
   - Desempenho

Um meio para contemplar todos esses requisitos é a utilização de *padrões de projeto*. Alguns motivos para isto são:

1. Desenvolvedores iniciantes podem utilizar padrões de projetos catalogados por "experts" para obter conhecimento e experiência na construção de aplicações orientadas a objeto.
2. Com a boa utilização de padrões de projeto a comunicação entre desenvolvedores, e a manutenção de sistemas, tornam-se menos complexas.
3. Padrões de projeto podem ajudar a encontrar abstrações que tornem o software mais flexível e reutilizável.

Cada padrão descreve um problema que ocorre várias vezes em um ambiente de desenvolvimento de software, podendo ser utilizado milhões de vezes em situações diferentes, constituindo-se assim num dos cernes da solução [GAM94].

Um padrão de projeto orientado a objeto descreve uma situação em que várias classes cooperam para realizar uma determinada tarefa, criando uma organização específica, ou seja, micro-arquiteturas de classe e objetos com seus papéis e colaborações. Essa micro-arquitetura é uma solução genérica a um problema que pode ocorrer em diversas aplicações. Dizemos que um padrão de projeto resolve um problema recorrente.

A definição de um padrão de projeto é ela própria padronizada da seguinte forma (tabela 3.1) [GAM94]:

Documentação de Padrões de Projeto	
Nome	• Identificador usado para descrever o problema, suas soluções e conseqüências, em uma ou duas palavras;
Descrição	• Apresenta o problema e seu contexto; • Indica quando devemos aplicar o padrão; • Pode apresentar problemas específicos de projeto, ou descrever estruturas de classes ou objetos que geralmente tornam o projeto inflexível (estruturas a evitar, portanto).
Solução	• Descreve as relações, as responsabilidades, as colaborações e os elementos que compõem a solução do projeto; • Serve como um gabarito que pode ser utilizado em várias situações.
Conseqüências	• Apresentam os resultados e "trade-offs" da utilização do padrão; • Ajudam a avaliar se os resultados obtidos justificam a adoção do padrão; • Apresentam os impactos em termos de flexibilidade, extensibilidade e transportabilidade.

Tabela 3.1: Documentação de Padrões de Projeto.

Padrões de Projeto têm tomado dimensões que sobrepõem a parte de projeto físico. Já existem padrões de análise, testes, etc. Na próxima seção, tratamos do relacionamento entre padrões de projeto e framework e exemplificaremos o padrão “Template Method”.

3.3.1 Padrões de Projeto e Framework

Num projeto de um framework, padrões de projeto ajudam a encontrar abstrações que tornam o software mais flexível e reutilizável.

Por exemplo, num framework temos o problema de como instanciar objetos de classes que foram “plugadas” no framework. Neste caso, poder-se-ia utilizar o padrão de projeto “Abstract Factory”, que provê uma solução genérica para este tipo de problema. Os padrões de projeto mais comuns utilizados em projetos de framework são: “Façade”, “Command”, “Template Method”, e “Abstract Factory”.

Dentre os padrões citados, “Template Method” é a base para a construção de um framework.

3.3.2 Template Method

“Template Method” define o esqueleto de um algoritmo para uma operação, transferindo alguns passos desta operação para as subclasses. Este padrão permite que uma subclasse redefina certos passos de um algoritmo, sem alterar a estrutura do mesmo.

Seja um problema prático *“Criar uma solução genérica de* “Login” *que possa ser reutilizada em qualquer projeto.”*

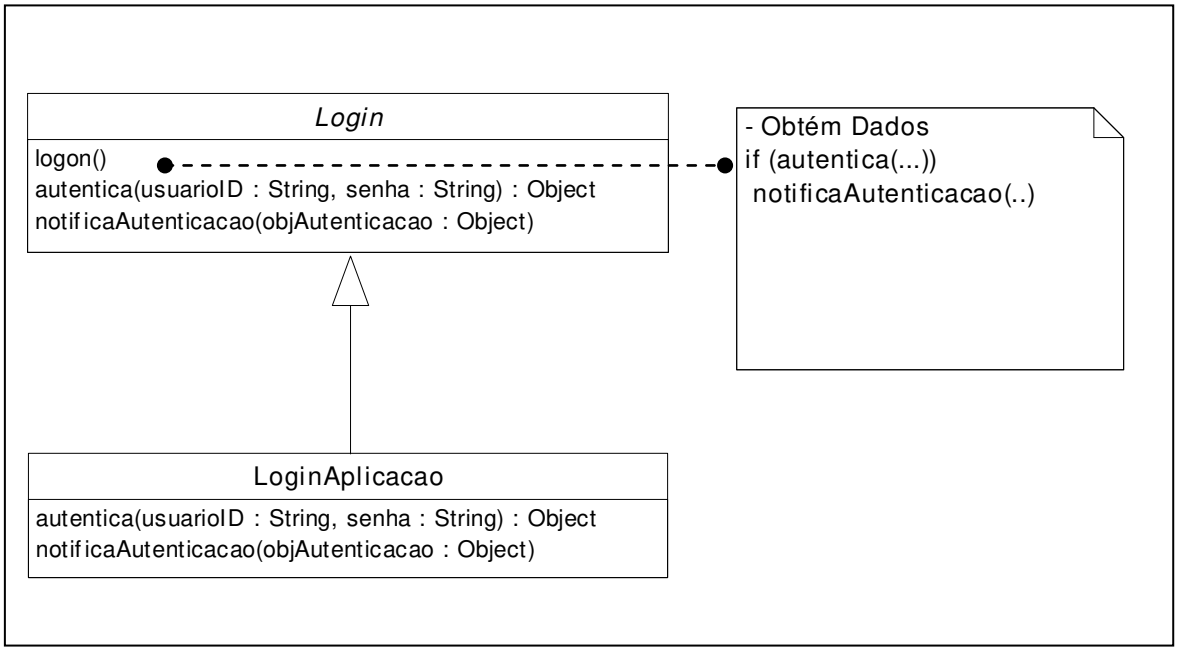


Figura 3.4: Classe e subclasse de Login

A figura 3.4 ilustra um digrama de classes com a solução do problema enunciado. Define-se uma classe abstrata Login — estrutura genérica preparada para ser `customizada´ pelas futuras aplicações. Esta classe contém um método “logon” — “Template Method” — que `encapsula´ a lógica do processo genérico de login. Este método fará chamadas aos métodos abstratos “autentica” e “notificaAutenticacao”. Os métodos abstratos serão implementados pela subclasse concreta LoginAplicacao que estende a classe Login. A classe LoginAlicação é responsável pela lógica específica da aplicação que está usando o `template´ de Login.

Adota-se uma solução padrão para problemas do tipo exemplificado. A figura 3.6 apresenta uma micro-arquitetura genérica, ou seja, servindo para qualquer aplicação que necessite do “Template Method”. Esta micro-arquitetura é base para construção de um Framework.

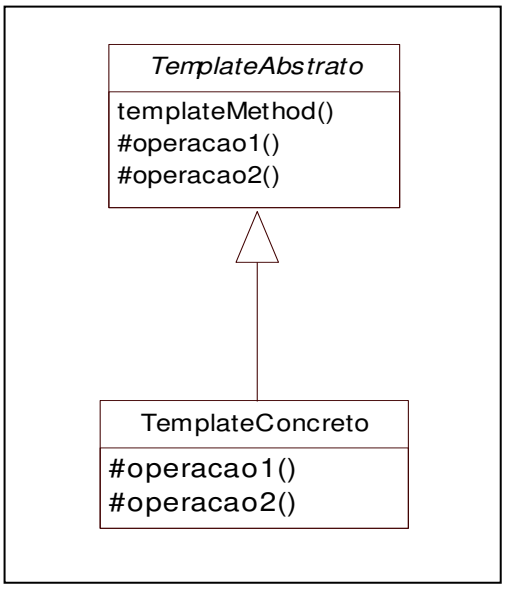


Figura 3.5: Padrão Template Method

A classe *TemplateAbstrato* é a classe abstrata que contém o comportamento genérico. Na aplicação de Login equivale a classe *Login*. Esta classe possui um método "*templateMethod*" que encapsula a lógica da solução e faz chamadas a operações abstratas (#operacao1, #operacao2). Por fim, temos a subclasse *TemplateConcreto* que estende a classe *TemplateAbstrato* para definir as operações, operação1 e operação2, com funcionalidades específicas.

A próxima seção trata de um processo de desenvolvimento voltado para a construção de um framework.

3.4 O Processo Unificado

A tendência é de desenvolvimento de sistemas de software cada vez maiores e mais complexos. Isto se deve à evolução dos computadores, que estão se tornando cada vez mais poderosos, fazendo com que os usuários alimentem mais expectativas em relação a eles. Esta tendência também tem sido influenciada pela expansão do uso da internet e da multimídia.

Esse contexto leva à necessidade de utilização de processos de desenvolvimento que sejam adequados ao dinamismo e às expectativas do mercado.Em nível de implementação, já existe um consenso de que OO é a chave para o desenvolvimento rápido de aplicações complexas. Em relação a análise e projeto, faltava uma notação unificada realmente eficaz. Esta lacuna foi preenchida com o surgimento da linguagem UML — "Unified Modeling Language" [JAC98]. UML é uma linguagem de modelagem, mas ela não define um processo de desenvolvimento. Como complemento à UML, surgiu uma especificação para um processo de desenvolvimento de software, chamada de Processo Unificado [JAC98].

O Processo Unificado contempla todo o ciclo de vida de um projeto, e guia as equipes de desenvolvimento nas atividades práticas de engenharia de software. Ele tem sua base em três conceitos chaves:

1. *Desenvolvimento orientado por caso de uso*
2. *Ênfase em uma arquitetura, desde o inicio*
3. *Abordagem iterativa e incremental*

1. Desenvolvimento Orientado por Caso de Uso

A construção de um software tem por objetivo atender às requisições dos usuários. As requisições ditas funcionais no Processo Unificado, desembocam nos *casos de uso* — pedaços de funcionalidades de sistemas que retornam valores em que os usuários estão interessados. O conjunto de casos de uso forma o diagrama de casos de uso, que descreve o contexto do sistema, com todas as suas entidades externas (Atores), funcionalidades (Casos de Uso) e os seus relacionamentos.

Na fase de análise, os casos de uso captam e validam as funcionalidades com os usuários. As fases seguintes, projeto, implementação e testes, serão norteadas pelos casos de uso.

2. Ênfase na Arquitetura (desde a fase de elaboração do projeto).

A arquitetura de um software é o mais importante aspecto do sistema, e refere-se à estrutura de subsistemas e de componentes. É um mapa do sistema, com as diferentes partes,

suas interações e mecanismos de interconexões. Devido à importância da arquitetura no desenvolvimento de software, o Processo Unificado propõe que ela seja definida o mais cedo possível, quase que em paralelo com a coleta dos requisitos.

Ao projetar-se a arquitetura, vários fatores são importantes.

- A plataforma que o software vai rodar ─ a arquitetura do computador, o sistema operacional, o gerenciador de Banco de Dados, etc;
- Considerações de desenvolvimento ─ linguagem de programação, internacionalização, etc;
- Integração com sistemas legados e requisitos não-funcionais ─ performance, confiabilidade, tipo de interface, etc;

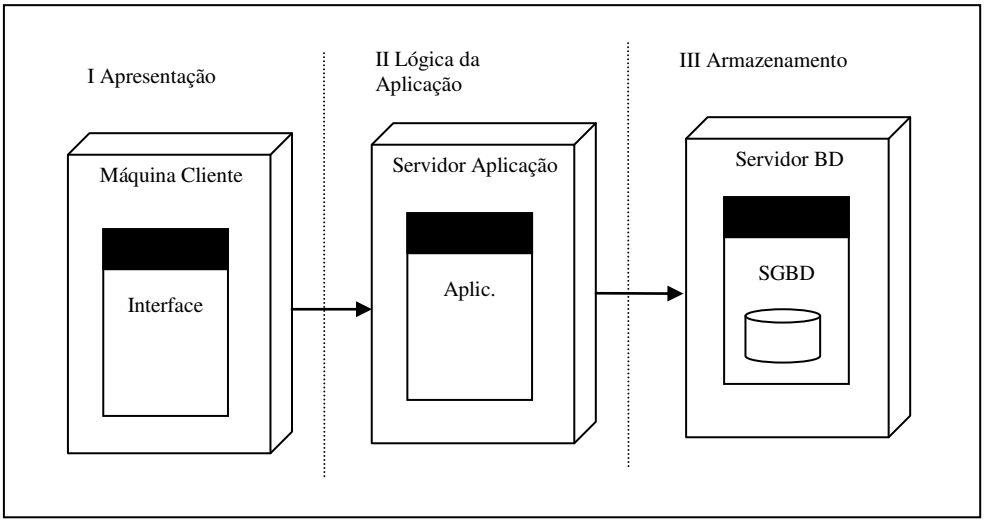


Figura 3.6: Visão clássica de um arquitetura em três camadas

A figura 3.6 apresenta um arquitetura comum para sistemas de informação. A arquitetura está dividida em três camadas: apresentação, lógica da aplicação e armazenamento. Na figura em questão cada camada está distribuída em máquinas diferentes (representadas pelas caixas). Listamos alguns elementos das camadas.

- I Apresentação – menus, botões, janelas, etc;
- II Lógica da Aplicação – objetos e regras de negócio;
- III Persistência – mecanismos de armazenamento persistente (XML, tabelas em um SGBD, etc);

3. Abordagem Iterativa e Incremental

Os processos de desenvolvimento convencionais têm por premissa ciclos de desenvolvimento em `cascata', ou seja, faz-se necessário o término de uma fase para o início de outra. Neste processo, o software só é "lançado" quando as fases de análise, projeto, implementação e testes (do software como um todo) já foram concluídas. Para sistemas que têm um ciclo de desenvolvimento extenso, temos o problema da primeira versão ser disponibilizada muito tempo após a coleta dos requisitos, que podem ter sofrido mudanças ao longo desse período.

O Processo Unificado iterativo e incremental difere dos processos convencionais, pois baseia-se num ciclo de desenvolvimento curto, em que as versões (incrementos) são disponibilizadas ao longo do ciclo de desenvolvimento do software.

Para a construção das versões, são feitas várias iterações. As iterações são uma seqüência de etapas — análise , projeto, implementação, testes e implantação — que detalham e ampliam o modelo. Ao final de um ciclo de interações, temos uma versão do sistema.

A vantagem desse processo é que ao longo de todo o processo podem ser feitas avaliações a respeito do cumprimento dos requisitos funcionais, desempenho, confiabilidade, cronograma, recursos utilizados, entre outros. Esta forma de trabalhar diminui riscos que são inerentes ao processo de desenvolvimento de um software. A figura 3.7 exemplifica graficamente o Processo Unificado iterativo e incremental [JAC98] .

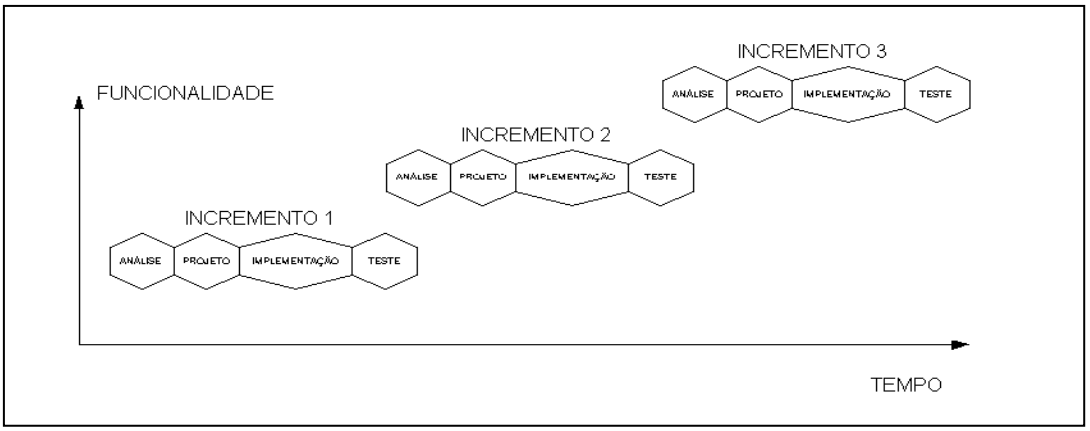


Figura 3.7: Processo Iterativo e Incremental

O gráfico da figura 3.7 apresenta a evolução dos incrementos no tempo. Cada incremento para ser construído passa por todo o ciclo de iterações. Ao final de um ciclo de iterações, temos um incremento com mais funcionalidades.

Durante o processo de desenvolvimento dos incrementos, deve-se utilizar técnicas de reutilização de software, para aumentar a produtividade dos desenvolvedores. A obtenção de bons resultados é dependente da experiência de profissionais que desenvolvem projetos (com o pensamento voltado para a reutilização), e do emprego de técnicas de reutilização, como Framework e Padrões de Projeto.

Ressalta-se que projetar um Framework difere de projetar uma aplicação convencional. Assim, existe a necessidade de adotar-se um processo específico voltada para a construção desse tipo de projeto.

3.5 Um Processo Voltado para o Desenvolvimento de Framework Orientado a Objeto

O processo de desenvolvimento apresentado nesta seção é fundamentado em [LAN95] Este trabalho segue a linha do processo unificado interativo e incremental, 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, deve-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 3.8 apresenta graficamente as etapas do processo de construção do Framework.

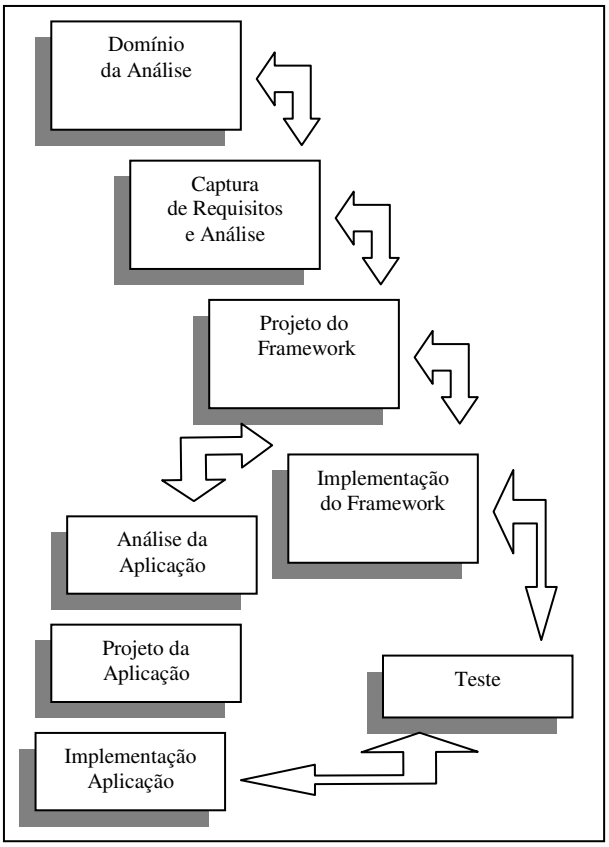


Figura 3.8: Etapas do processo de desenvolvimento do Framework.

As próximas subseções resumem cada etapa do processo.

3.5.1. 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 *domínio do escopo* e um *modelo estático* contendo os objetos e classes relacionadas com o domínio da aplicação.

3.5.2 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 [LAN95].

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.

Seguiu-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 3.9 o ilustra.

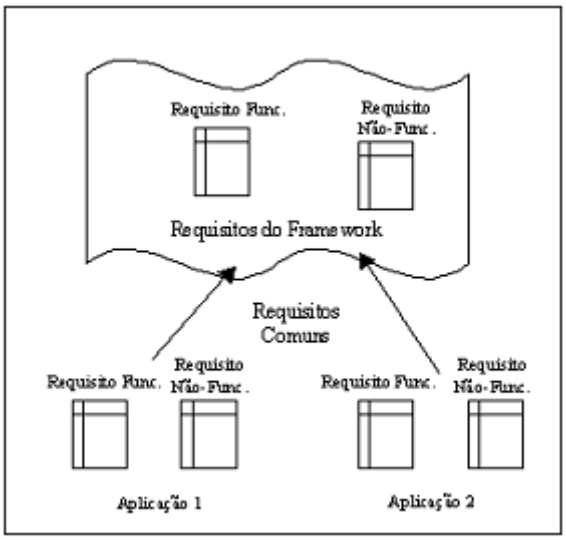


Figura 3.9: Separação dos requisitos das aplicações e dos requisitos pertencentes ao Framework.

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 Caso de Uso: “Uses” e “Extends”.

1. O relacionamento ”Uses” é utilizado quando um Caso de Uso faz uso da funcionalidade de um outro Caso de Uso para prover e complementar a sua funcionalidade.

2. O relacionamento “Extends” é utilizado quando um Caso de Uso especializa um outro Caso de Uso.

Os relacionamentos entre Caso 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”) ou usados (“uses”) 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).

3.5.3 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, digrama de classes, diagrama de estado 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 defini-se modelos gráficos com a disposição das camadas em sistemas computacionais. Para as camadas que estão em sistemas computacionais distintos deve-se elucidar como e feita à 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 grande 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 [LAR00]. Para a construção dos diagramas são utilizados os objetos e classe de negócio identificados na análise.

Listamos as principais atividades

1. Novos objetos e classes são identificados;
2. Associações entre classes são transformadas em referências, agregações e heranças;
3. Definição de métodos e atributos;

4. Estudo do ciclo de vida das classes mais relevantes (diagramas de estado);
5. 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 [LAN95]. Assim, provê-se classes que `encapsulam´ comportamento genérico das futuras aplicações.

3.5.4 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 com está ilustrado na figura 3.8. 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.

3.6 Conclusões

Os temas apresentadas nos Capítulos II e III são a base para a definição e construção do S-DW-E — Servidor de DW Extensível, objetivo principal deste trabalho.

A captura das funcionalidades comuns das ferramentas OLAP foi encapsulada no framework S-DW-E através de uma interface OLAP padrão. Assim, independentemente da tecnologia de implementação de um DW, o S-DW-E torna disponível uma interface OLAP comum, fundamentada no modelo conceitual formal de R. Agrawal, descrito no Capítulo II.

No modelo de Agrawal, a estrutura de dados é representada por um *cubo de dados multidimensional.* Qualquer operação de manipulação ou consulta é feita sobre um cubo de dados. Os operadores sobre cubos de dados multidimensionais “Pull”, “Push”, “Restrict”, “Merge”, “Destroy” e “Join” proporcionam meios para que os usuários resolvam típicas consultas OLAP.

Para a construção do S-DW-E 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 Servidor.

Uma solução proposta, inicialmente, por nós foi a de projetar um software que fosse “customizado” através de arquivos de configurações. Os arquivos de configurações conteriam informações para o mapeamento. Assim, o Servidor habilitar-se-ia a trabalhar com uma tecnologia OLAP.

Contudo, essa solução não foi levada adiante e tornou-se inviável, pois cada configuração exigia que novos algoritmos fossem implementados e adicionados à aplicação original.

Após o estudo de algumas soluções, verificou-se que a tecnologia de framework adequava-se perfeitamente às nossas necessidades.

Como processo de desenvolvimento, adotamos o Processo Unificado Iterativo e Incremental, resumido neste capítulo.

Os próximos dois capítulos tratam de apresentar em detalhes o S-DW-E.

CAPÍTULO IV

O FRAMEWORK S-DW-E: REQUISITOS E ANÁLISE

Este capítulo descreve a análise e o projeto conceitual de um servidor de DW extensível — S-DW-E —, objetivo principal deste trabalho. Um exemplo de utilização do S-DW-E é apresentado para ajudar a compreensão do leitor.

4.1 Contexto do S-DW-E: "Business Case"

O mercado possui um grande número de ferramentas de consultas OLAP. Muitas estão fortemente acopladas a um SGBD (Discoverer – Oracle, MetaCubo – Informix, etc). Para serem adaptadas a outros SGBDs, essas ferramentas teriam que ser praticamente refeitas.

Outras ferramentas OLAP, como BusinessObject, podem somente ser acopladas a SGBDs relacionais, visto que sua tecnologia é ROLAP. Por exemplo, não se pode fazer uso dos novos recursos dos SGBDs objeto-relacionais nesse tipo de ferramenta.

O Servidor de DW Extensível — S-DW-E —, objeto do nosso trabalho, não padece dessas limitações, visto que ele foi construído segundo a tecnologia de framework, discutida no Capítulo III.

Listamos as principais vantagens na adoção da técnica de projeto de software, framework, na construção do S-DW-E:

1. Em âmbito geral, o framework S-DW-E possibilita criar uma estrutura genérica para a parte comum das aplicações de consulta OLAP. Na sua essência, o S-DW-E é constituído de *classes abstratas*, que são estendidas com o comportamento especifico das aplicações, quaisquer que sejam as tecnologias específicas ─ ROLAP, OROLAP, MOLAP, etc.
2. O Framework S-DW-E é capaz de gerar 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), tirando proveito das otimizações específicas de cada SGBD que são encapsuladas nas extensões do Framework.
3. Para o desenvolvimento de uma nova aplicação OLAP, reutiliza-se a análise, o projeto, o código e os testes do framework S-DW-E, alcançando-se assim uma alta produtividade de desenvolvimento.

4.1.1 Trabalhos Relacionados

Considerando que o `coração´ do S-DW-E é o seu Gerente de Metadados Extensíveis, a ser discutido posteriormente, é interessante incursionar na literatura para examinar o que tem sido feito a respeito de padrões de metadados, e de metadados extensíveis.

Apesar de algumas tentativas de criar padrões de Metadados para DWing ainda não existe um padrão de fato aceito [VET 00]. 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 [STÖ99], 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).

4.1.2 Objetivos do S-DW-E

O S-DW-E é uma ferramenta de consulta OLAP para ambiente de DWing que, em sua versão atual, contempla a parte do DW e a parte "front-end" [SAM00]. Como o S-DW-E atualmente não trata de todos os elementos de um ambiente de DWing, ao longo do restante deste trabalho faremos referências unicamente a DWs. As funcionalidades disponíveis no S-DW-E para a construção e consulta a DWs visam atender a seus diferentes tipos de usuário: usuários OLAP, projetistas de DW, programadores de aplicação e administradores de banco de dados.

A essência do S-DW-E está em seu repositório de metadados, que 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 S-DW-E facilmente "customizado" para tecnologias ROLAP ou MOLAP de construção de DW, ou para futuras tecnologias de DW.

A facilidade da "customização" é possível porque S-DW-E 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 OLAP acopladas a uma tecnologia de gerência de DW. Na prática, temos então, através do S-DW-E, diferentes servidores OLAP: ROLAP, OROLAP, MOLAP, etc.

A técnica de framework, detalhada no Capítulo III, foi a solução adotada para a realização da parte genérica ou reutilizável do S-DW-E. 'Pedaços' adicionais de projeto e código — extensão de metadados, projeto e implementação de novos esquemas lógicos e físicos de DW envolvendo novas tecnologias — permitem a evolução do S-DW-E. Na seqüência são descritos a arquitetura do S-DW-E, seus requisitos funcionais e não funcionais. As descrições são acompanhadas de exemplos ilustrativos.

4.1.3 Arquitetura do S-DW-E

A arquitetura do S-DW-E está disposta em três camadas (figura 4.1).

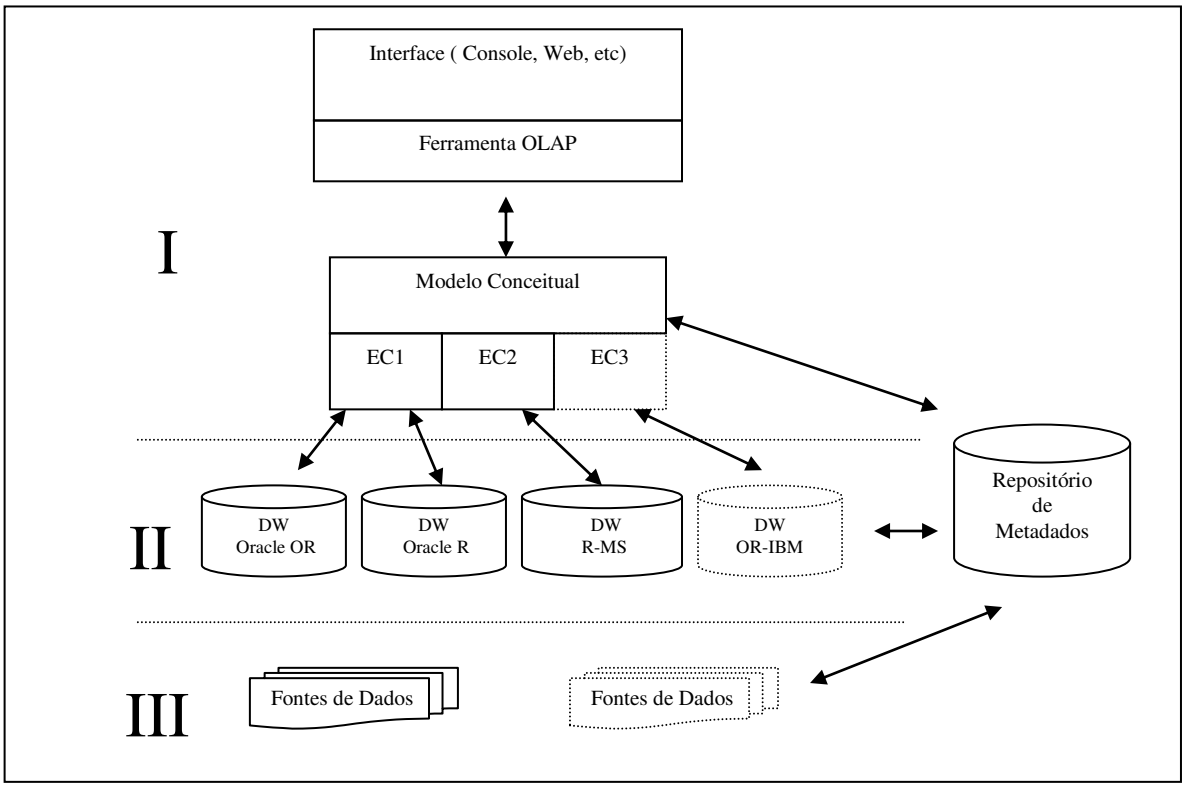


Figura 4.1: Arquitetura do S-DW-E.

A primeira camada — Camada Conceitual (I) — compreende uma ferramenta OLAP de consulta a DWs, cujos esquemas (EC_i) são segundo o modelo conceitual de R. Agrawal [AGR 99], discutido no Capítulo II. A interface usuário-DW por meio de um esquema conceitual, é pois padronizada, e independente das suas diversas tecnologias de implementação de DW.

A segunda camada — Camada de Implementação (II) — trata das implementações dos diversos esquemas conceituais de DW, tanto lógica como fisicamente. Por exemplo, o esquema EC_1 poderia ser implementado tanto com tecnologia OR-Oracle, como com tecnologia R-Oracle, para fins de comparação de desempenho com as tecnologias objeto-relacional e relacional. Por sua vez, o esquema EC_2 poderia ser implementado com tecnologia

Microsoft (R-SQLServer). Mais tarde, o DW com esquema EC3 seria acoplado (linhas pontilhadas, na figura) no S-DW-E com tecnologia OR-IBM.

A terceira camada — Camada Operacional (III) — compreende as fontes de dados operacionais que alimentam os diversos DWs e os serviços (extração, validação, carga) necessários para garantir a consistência dos DWs. Embora indispensável à construção de ambientes de DWing, esta camada não é tratada no presente trabalho.

Por fim, e permeando as três camadas, temos o Repositório de Metadados. Nele, são armazenadas e mantidas informações sobre as estruturas e os conteúdos das três camadas da arquitetura[2]. O repositório armazena e mantém também as dependências entre as camadas.

Enumeramos os tipos de informação das camadas I e II (tabela 4.1).

I- Conceitual	1. Modelo conceitual de dados; 2. Esquemas conceituais ; 3. Dependências entre as camadas I e II.
II- Lógico	1. Modelos lógicos de implementação; 2. Esquemas lógicos de implementação; 3. Dependências entre um modelo lógico e seu modelo físico.
II- Físico	1. Estruturas de armazenamento de tabelas; 2. Cardinalidade de tabelas e outras estatísticas sobre tabelas.

Tabela 4.1: Informações das Camadas I e II da Arquitetura do S-DW-E.

No S-DW-E, e de acordo com a terminologia de banco de dados, a diferença entre os esquemas e os modelos é que os esquemas são a extenção dos modelos, ou um esquema é a realização de um modelo em um DW.

[2] Lembramos que, no estágio atual de desenvolvimento do trabalho, somente as camadas I e II são contempladas.

Para a construção de um DW é necessária a definição de três esquemas inerentes ao S-DW-E:

- *DW conceitual;*
- *DW lógico;*
- *DW físico;*

Observa-se que existe uma forte relação entre os esquemas das diferentes camadas. O esquema DW conceitual, que é o esquema com o qual o usuário final interage, está relacionado com um esquema lógico. Esta relação existe para que as consultas executadas no esquema conceitual sejam transcritas para a linguagem do SGBD acoplado no Servidor.

Já o esquema lógico está relacionado com um esquema físico com o objetivo de facilitar a gerência do banco de dados pelo Administrador de Banco de Dados, e assim, obter o máximo de performance no processamento das consultas. Informações físicas sobre os esquemas de banco de dados podem ser representadas no esquema físico do DW.

4.2 Modelo Conceitual do S-DW-E

Descrevemos no diagrama de classes da figura 4.2 (modelo estático) as classes e suas associações, que descrevem o modelo conceitual de todos os DWs implementados no S-DW-E. As classes *DW, Cubo e Dimensão* e suas associações representam de forma genérica os futuros esquemas conceituais. Como no S-DW-E podem ser representados diversos esquemas conceituais, temos portanto a associação da classe *Servidor* com a classe *DW*. A classe *Servidor* tem a responsabilidade de fazer a interface com o Usuário DW para que este possa efetuar consultas aos esquemas conceituais. A classe *Servidor* colabora com a classe *Mapeador*, pois esta atua na resolução das consultas OLAP, `encapsulando´ funcionalidades para resolver os operadores do modelo conceitual de R. Agrawal[3]. A classe *Interpretador* auxilia a classe *Mapeador* na análise léxica e sintática das consultas. Por fim, temos a classe *CargaMetadado* que faz a carga do repositório de metadados com as informações relativas aos esquemas DW.

[3] No incremento atual do S-DW-E, estão implementadas as operações Restrict, Merge e Destroy, utilizando as tecnologias ROLAP e OROLAP de construção de DW.

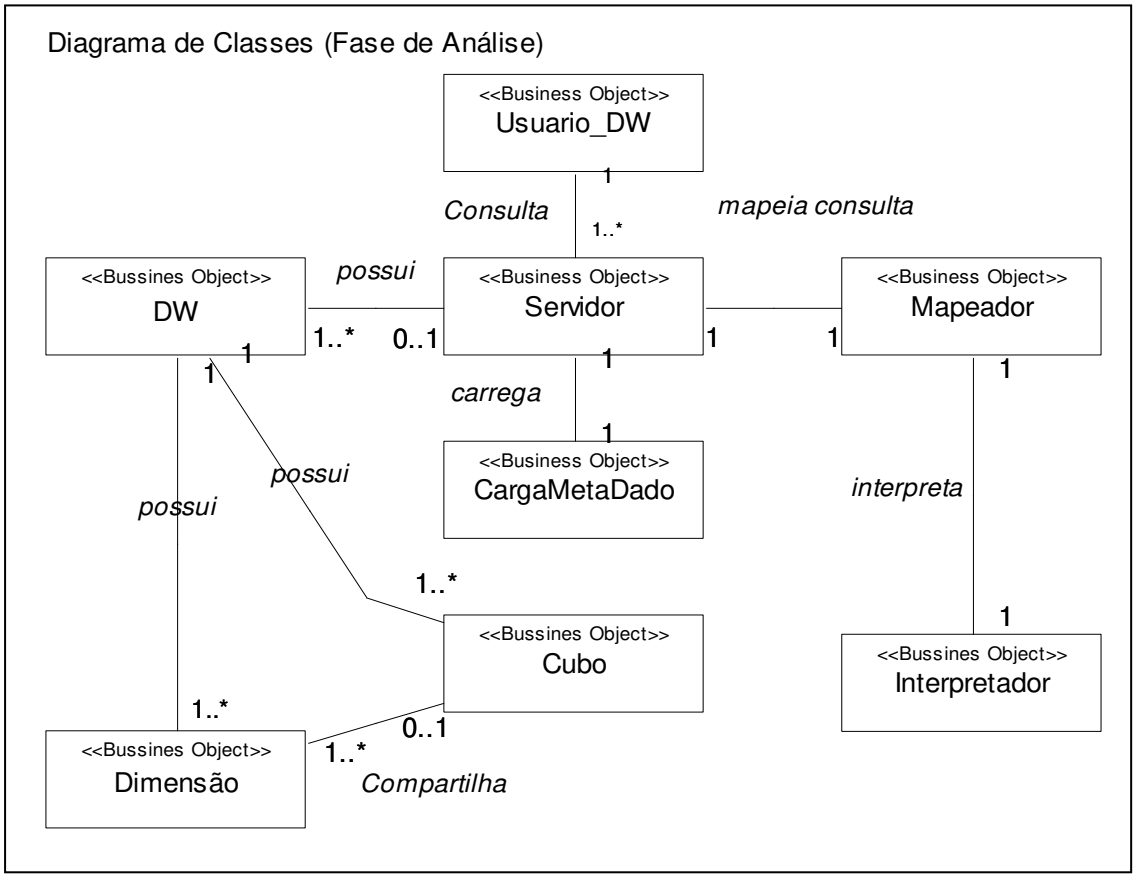


Figura 4.2: Diagrama de Classes S-DW-E fase de análise.

4.3 Requisitos Funcionais e Não-Funcionais

A análise e o projeto do S-DW-E levaram em conta 4 tipos de usuário:

- Usuário OLAP (ator Usuário DW) - Usuários que interagem com o S-DW-E para efetuar consultas OLAP através de esquemas conceituais (Tomadores de Decisão).

- Projetistas de DW (ator Projetista DW) - Usuários que fazem uso do modelo conceitual para definir esquemas conceituais de DW.

- Programadores de Banco de Dados (ator 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) .

- Administradores de Banco de Dados (ator Administrador DW) - 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.

Usuário DW, Projetistas DW, Implementador DW e Administrador DW são *atores* que interagem com o S-DW-E. Nas figuras 4.3 e 4.4, as interações são descritas por meio de dois *diagrama de casos de uso*. Nos cenários das figuras são apresentados os diagramas de casos de uso do S-DW-E, em que o S-DW-E é estendido para a implementação de um DW conceitual segundo duas tecnologias, R-SQLServer e OR-Oracle.

Por questões de visualização, dividimos o diagrama de caso de uso em dois.

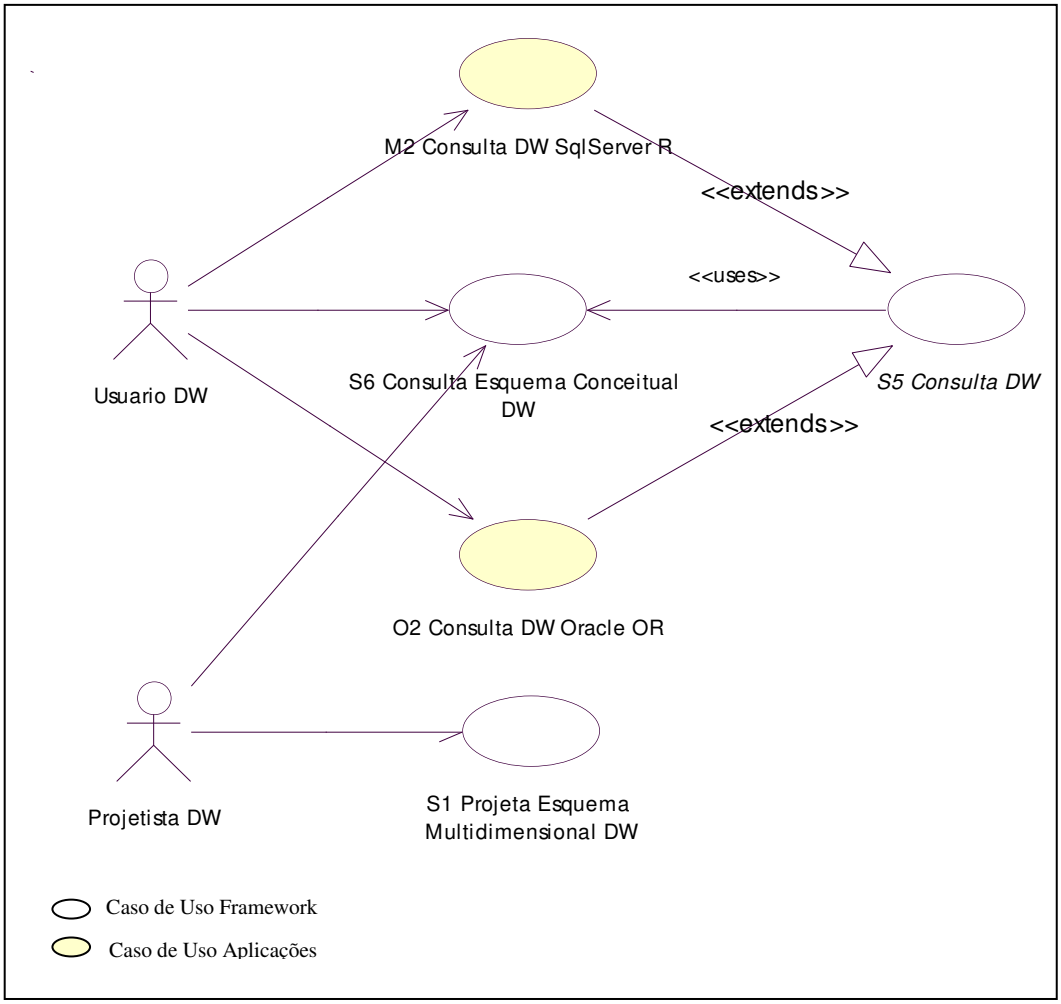


Figura 4.3: Diagrama de Caso de Uso para o cenário de consulta e projeto DW.

O primeiro diagrama (figura 4.3) descreve o cenário da interação do Usuário DW com o S-DW-E para consultas a DWs. Neste diagrama é descrito, também, a interação do Projetista DW que usa o servidor para definição de esquemas conceituais DW.

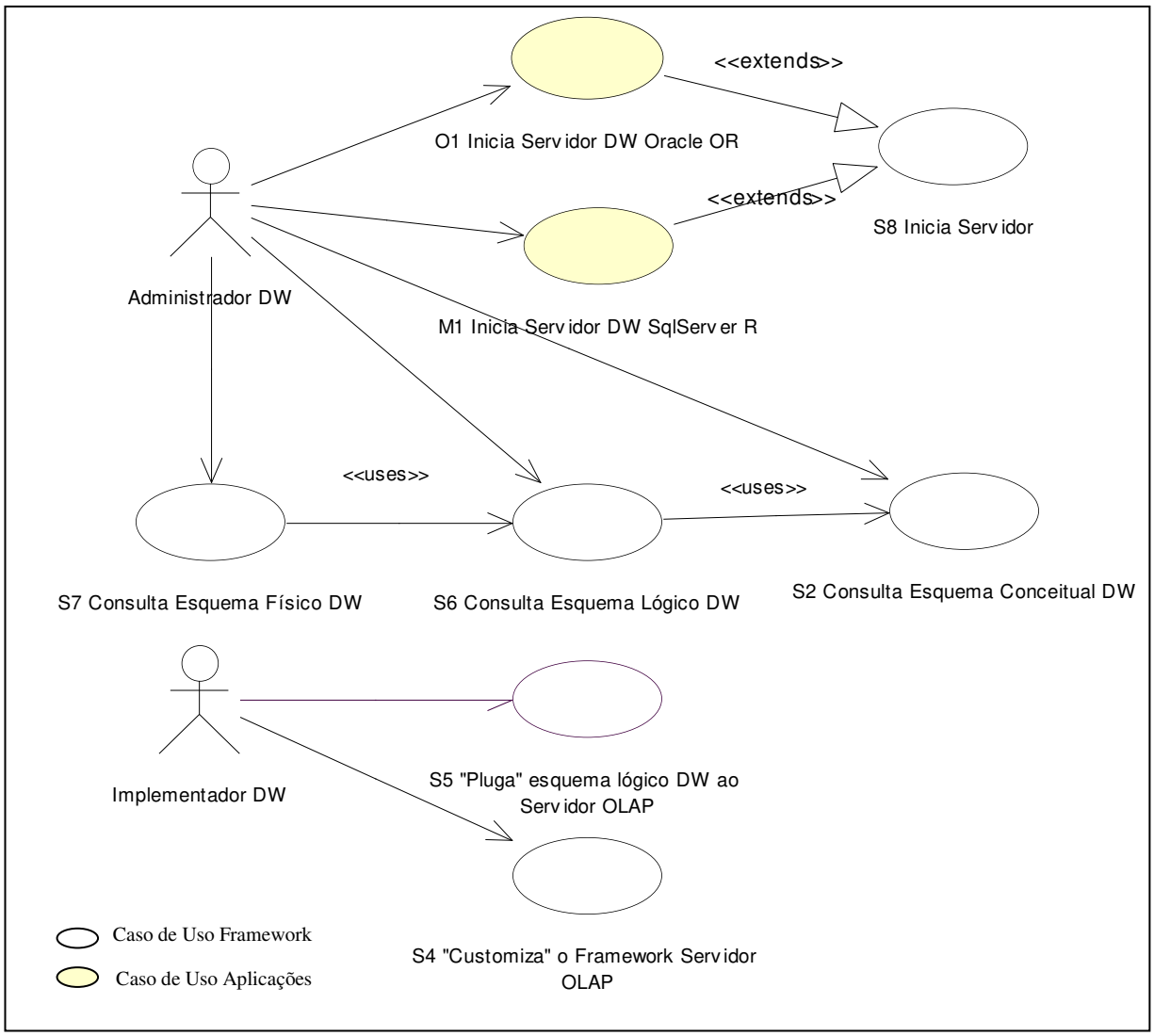


Figura 4.4: Diagrama de Caso de Uso para o cenário de “Cutomização”, inicialização e “plug” do Servidor.

O segundo diagrama (figura 4.4) descreve o cenário para os usuários que estão interessados em “customizar” o servidor, ou seja, habilitá-lo a operar com uma determinada tecnologia de armazenamento. Este cenário descreve ainda a interação do Administrador DW com a funcionalidade de iniciar o servidor de acordo com a sua extensão (tipo de servidor acoplado). É descrito, também, a interação do Implementador DW com a funcionalidade de acoplar esquemas lógicos que serão relacionados com os esquemas conceituais.

No diagrama de caso de uso ilustrado nas figuras 4.3 e 4.4 as funcionalidades do S-DW-E são representadas por elipses brancas. Os casos de usos de aplicações que estendem o S-DW-E são representadas por elipses de cor amarela.

Os casos de uso brancos, pertencentes ao servidor, podem ser concretos ou abstratos. Os caso de usos concretos são usados diretamente pelos Atores. Já os abstratos (textos em itálico) nunca serão usados diretamente pelos Atores e para cumprirem suas funcionalidades necessitam ser estendidos (“extends”) ou usados (“uses”) por outros casos de uso.

Requisitos Funcionais do S-DW-E (tabela 4.2)

Caso de Uso	Descrição
[S1] *Consulta DW (abstrato*)	Permite ao Usuário DW realizar consultas OLAP sobre um esquema conceitual, definido pelo Projetista DW, por meio de operadores *Restrict*, *Destroy*, *Merge* etc., do modelo conceitual adotado. Este caso de uso utiliza o caso de uso (S2) para realizar, validar e resolver as consultas OLAP.
[S2] Consulta Esquema Conceitual DW	Permite ao usuário DW consultar o esquema conceitual definido pelo Projetista DW. Exemplo: “Quais são as dimensões de um certo cubo? “.
[S3] Projeta Esquema Multidimensional DW	Permite ao Projetista DW criar um esquema multidimensional de um DW, com elementos: Cubo, Dimensão, Hierarquia de Agregação, Célula do Cubo e Atributo de Dimensão. O esquema em questão deve ser vinculado a um esquema lógico acoplado no S-DW-E. [S5]
[S4] "Customiza" o Framework Servidor OLAP	Permite ao Programador de Banco de Dados "customizar" o S-DW-E para uma nova tecnologia de implementação de DW. A “customização”consiste na extensão do S-DW-E, criando um tipo de servidor apto a utilizar uma

	determinada tecnologia de armazenamento (ROLAP, MOLAP, etc).
[S5] "Pluga" esquema lógico DW ao Servidor OLAP	Permite ao Programador de Banco de Dados acoplar um determinado esquema lógico DW ao Servidor OLAP para implementar um determinado esquema conceitual. Por exemplo, um esquema em estrela relacional, sintaxe Microsoft SQLServer, implementando um esquema conceitual multidimensional.
[S6] Consulta Esquema Lógico DW	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 [S2] Consulta Esquema Conceitual DW, para obter informações do esquema lógico. Todas as informações do esquema lógico estão relacionadas com o esquema conceitual.
[S7] Consulta Esquema Físico DW	Permite que sejam realizadas consultas sobre 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, Tablesapce, Espaço que esta ocupando, etc;"* Este caso de uso faz uso do caso de uso [S6] Consulta Esquema Lógico DW para realizar sua funcionalidade.
[S8] *Inicia Servidor*	Responsável por iniciar o S-DW-E. Neste processo é feita

(abstrato)	a carga de metadados nos níveis conceitual, lógico e físico das aplicações do S-DW-E.

Tabela 4.2:Requisitos Funcionais do S-DW-E

Requisitos Funcionais da Extensão Oracle (tabela 4.3).

Caso de Uso	Descrição
[O1] Inicia Servidor DW OR-Oracle	Estende o caso de uso [S8] Inicia Servidor do S-DW-E, com a carga dos metadados descrevendo os esquemas conceituais, lógicos e físicos relacionados com servidores OR-Oracle.
[O2] Consulta DW Oracle OR	Caso de uso que estende o caso de uso [S1] Consulta DW do Framework. Este caso de uso permite que o Usuário realize consultas OLAP sobre o esquema conceitual no Servidor DW OR-Oracle, a partir de operadores do esquema conceitual.

Tabela 4.3: Requisitos Funcionais da Extensão Oracle

Requisitos Funcionais da Extensão Microsoft (tabela 4.4)

Caso de Uso	Descrição
[M1] Inicia Servidor DW SQLServer R	Estende o caso de uso [S8] Inicia Servidor do S-DW-E, com a carga dos metadados descrevendo os esquemas conceituais, lógicos e físicos relacionados com servidores SQLServer R.
[M2] Consulta DW SqlServer R	Caso de uso que estende o caso de uso [S1] Consulta DW do Framework. Este caso de uso permite que o Usuário realize consultas

	OLAP sobre o esquema conceitual no Servidor DW SQLServer R, a partir de operadores do esquema conceitual.

Tabela 4.4: Requisitos Funcionais da Extensão Microsoft.

Com base nos objetivos enunciados, listamos os principais ***requisitos não-funcionais*** do S-DW-E:

- *[S1.1] Atendimento das requisições concorrentes dos usuários de forma eficiente.*

  Para atender as requisições concorrentes dos usuários de forma eficiente, o S-DW-E 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 S-DW-E faz uso das otimizações específicas do Servidor de Banco de Dados, obtendo assim, o máximo de desempenho do Servidor.

- *[S1.2] Transportabilidade entre ambientes operacionais.*

  O S-DW-E é facilmente transportável entre ambientes operacionais, pois foi todo desenvolvido em Java.

- *[S1.3] API para que ferramentas de "frontend" possam interagir com o S-DW-E.*

  Qualquer ferramenta de *"frontend"* pode fazer consultas OLAP utilizando os operadores e as estruturas multidimensionais do S-DW-E. Para isso, a ferramenta deve usar uma API (“Aplication Programming Interface”) fornecida pelo S-DW-E. Assim, temos um total desacoplamento do *"frontend"* com *"backend"*. API está descrita no Anexo III.

- *[S1.4] Ambiente Web de consultas “ad hoc”.*

  O S-DW-E disponibiliza um ambiente onde os usuários podem fazer consultas OLAP “*ad hoc”*. Para o usuário utilizar o ambiente, ele só necessita de uma máquina em rede que execute um "Browser" HTML. Entretanto, na versão atual, o usuário ainda precisa conhecer a BNF (Anexo I) dos operadores OLAP do modelo de consultas OLAP utilizado.

- *[S1.5] Operadores multidimensionais com alto poder de expressão para resolver consultas OLAP complexas.*

  Em sua versão atual, o S-DW-E suporta três operadores multidimensionais básicos (*Restrict*, *Destroy*, *Merge*), detalhados no Capítulo II, que possibilitam resolver uma gama variada de consultas OLAP, desde as mais simples até as mais complexas. O S-DW-E resolve típicas consultas OLAP complexas que envolvem *Slicing-dicing*, *Drill down*, *Drill up*, *Drill across*.

- *[S1.6] Suporte a múltiplas hierarquias de agregação* ad hoc.

  No modelo multimensional implementado pelo S-DW-E e detalhado no capítulo III, uma das propriedades de uma dimensão é a possibilidade de ter múltiplas hierarquias de agregação, definidas de maneira "*ad hoc"*. Por exemplo, uma das hierarquias da dimensão Tempo (onipresente em DWs) poderia ser *dia* $\rightarrow$ *mês* $\rightarrow$ *ano*. Uma outra poderia ser *trimestre* $\rightarrow$ *semestre* $\rightarrow$ *ano*, e assim por diante. Com o operador Merge, essas hierarquias podem ser definidas de maneira "*ad hoc"*.

- *[S1.7] Independência de tecnologias de Servidores de Banco de Dados.*

  O S-DW-E está preparado para adaptar-se a qualquer tecnologia de servidores de banco de dados. Desenvolvimento com base na tecnologia de framework (ver o capítulo III), o S-DW-E possibilita que ele seja estendido para que lhe seja acoplado qualquer Sistema de Gerência de Banco de Dados para dar suporte às consultas OLAP, com o mínimo de impacto.

- *[S1.8] Reutilização da análise, do projeto e do código do S-DW-E no desenvolvimento de extensões do S-DW-E para novas tecnologias de Servidores de Banco de Dados.*

  Outro benefício da tecnologia de framework, utilizada na construção do S-DW-E, é o de permitir ao desenvolvedor que vai implementar o código para "customizar" o servidor a utilizar novas tecnologias, reutilizar a análise, o projeto, a implementação e os testes do projeto do Framework S-DW-E.

- *[S1.9] Padrão de Nomes para Classes*

  No S-DW-E, possui um padrão de nomes para algumas classes, visando facilitar a identificação dos diferentes tipos de metadados (sufixos `_C´, `_L´ e `_F´, para classes de metadados conceituais, lógicos e físicos, respectivamente).

4.4 Planejamento dos Incrementos

O planejamento do primeiro incremento ou primeiro protótipo do S-DW-E teve como principio de escolha a funcionalidade mais relevante do sistema (Processo de Desenvolvimento, Capítulo III). O principal motivo para esta decisão foi deparar-se com as dificuldades de implementação e fazer uma verificação da arquitetura o mais breve possível. Os requisitos atendidos foram [S1] Consulta DW, [O2] Consulta DW Oracle OR e [M2] Consulta DW SqlServer R.

Fazendo parte do primeiro incremento, foram desenvolvidas duas extensões do framework S-DW-E. A primeira, utiliza o SGBD *Microsoft SQLServer 7 (SQLServer)* para prover consultas num ambiente ROLAP. A outra faz uso do *Oracle 8i (Oracle-OR)* para prover consultas num ambiente OROLAP.

Com o desenvolvimento do primeiro incremento observamos alguns direcionamentos do nosso trabalho.

- O framework está apto a ser adaptado a várias tecnologias de implementação de DWs? Sim.
- Quem prove mais recursos para a implementação de um DW com mais facilidade, a tecnologia ROLAP ou a tecnologia ORLAP? Verificou-se que os recursos Objeto-Relacionais possibilitaram criar mais facilmente um servidor OLAP.

O segundo e último incremento também levou em conta a prioridade das funcionalidades na sua definição. O requisitos contemplados foram [S2] Consulta Esquema Conceitual DW, [S8] *Inicia Servidor*, [O1] Inicia Servidor DW OR-Oracle e [M2] Consulta DW SQLServer R. O objetivo do segundo incremento foi prover um mecanismo de consulta

aos metadados conceituais e a carga dinâmica de metadados a partir de arquivos XML. Os demais requisitos que não foram implementados ficaram como opções para trabalhos futuros.

4.5 Estudo de Caso no S-DW-E

Nesta seção, ilustraremos um exemplo de uso do S-DW-E através de um estudo de caso, um DW Vendas. Todo processo será tratado em alto nível. Os detalhes de implementação de como a consulta é processada estão dispostas no Capítulo V.

O esquema conceitual do DW foi implementado em dois tipos de SGBDs: Oracle-OR e SQLServer. Deve-se, portanto, assumir que existem dois projetos de DW que têm esquemas conceituais iguais, mas com esquemas lógicos e físicos distintos. A figura 4.5 ilustra graficamente o esquema conceitual do DW Vendas.

Na figura 4.5 temos o cubo *C1* que é composto das dimensões *Loja, Tempo e Produto* e os elementos (Fatos) *Quantidade e Faturamento*. Para esse esquema são disponibilizadas ainda três dimensões: *MesAno, Família e Categoria*, para futuras agregações. As agregações são executadas conforme as hierarquias: *Data_MesAno e Produto_Cat*. Ou seja, para mudanças de `granularidade' do cubo *C1* deve-se obedecer a ordem das hierarquias. Por exemplo, para a o cubo *C1* mudar da dimensão *Tempo* (dia) para a dimensão *MesAno* deve-se utilizar a hierarquia Data_MesAno.

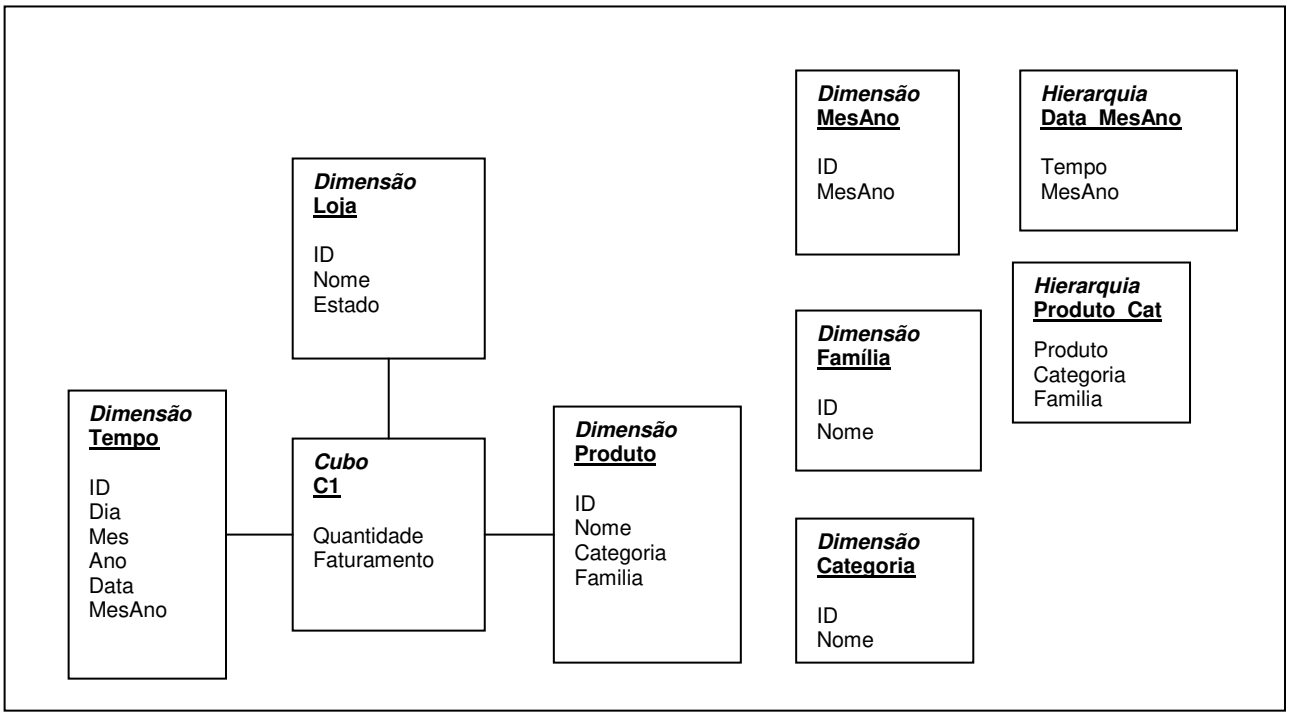


Figura 4.5: Esquema conceitual básico do DW Vendas.

Examinemos esta consulta *Qual o faturamento e a quantidade das vendas mensais para os produtos relacionados à categoria 'Cat1'?*.

Abaixo são descritos os 4 (quatro) passos principais para a resolução da consulta no ambiente de consulta *"adhoc"* para Web do S-DW-E.

- Passo 1: Conectar-se ao S-DW-E.

  O usuário através de um link acessa a página inicial do sistema de consulta *"adhoc"* para Web do S-DW-E. Ao acessar a página a conexão é estabelecida.

- Passo 2: Selecionar o DW e o Cubo inerentes à consulta.

  O usuário seleciona o DW (Vendas) sobre o qual deseja executar a consulta. O S-DW-E mostra então ao usuário os cubos de dados do DW Vendas, para que ele faça a seleção de um desses cubos (C1).

- Passo 3: Informar um identificador para a consulta e escrevê-la, com base na BNF da linguagem OLAP (Anexo I)

Finalizada a escrita da consulta, o usuário pressiona o botão Processa Consulta, como ilustra a figura 4.6. Este evento fará o S-DW-E fazer uma verificação léxica e sintática dos comandos do usuário.

Abaixo seguem os comandos Agrawal para a resolução da consulta do estudo de caso.

Consulta: Demo

- *C2=DESTROY(C1,LOJA.NOME) - Cada célula de C2 (Dia, Produto) agrega a quantidade e os valores das vendas de todas as lojas através da função de soma;*
- *C3=MERGE(C2,PRODUTO_CAT.PRODUTO.NOME|DATA_MESANO.TEMPO.DATA,SUM); - Cada célula de C3 (MesAno, Categoria) agrega a quantidade e os valores das vendas de todas os dias e produtos através da função de soma;*
- *C4=RESTRICT(C3,CATEGORIA.NOME,CAT1); - C4 conterá somente os produtos relacionados com a categoria "cat1";*

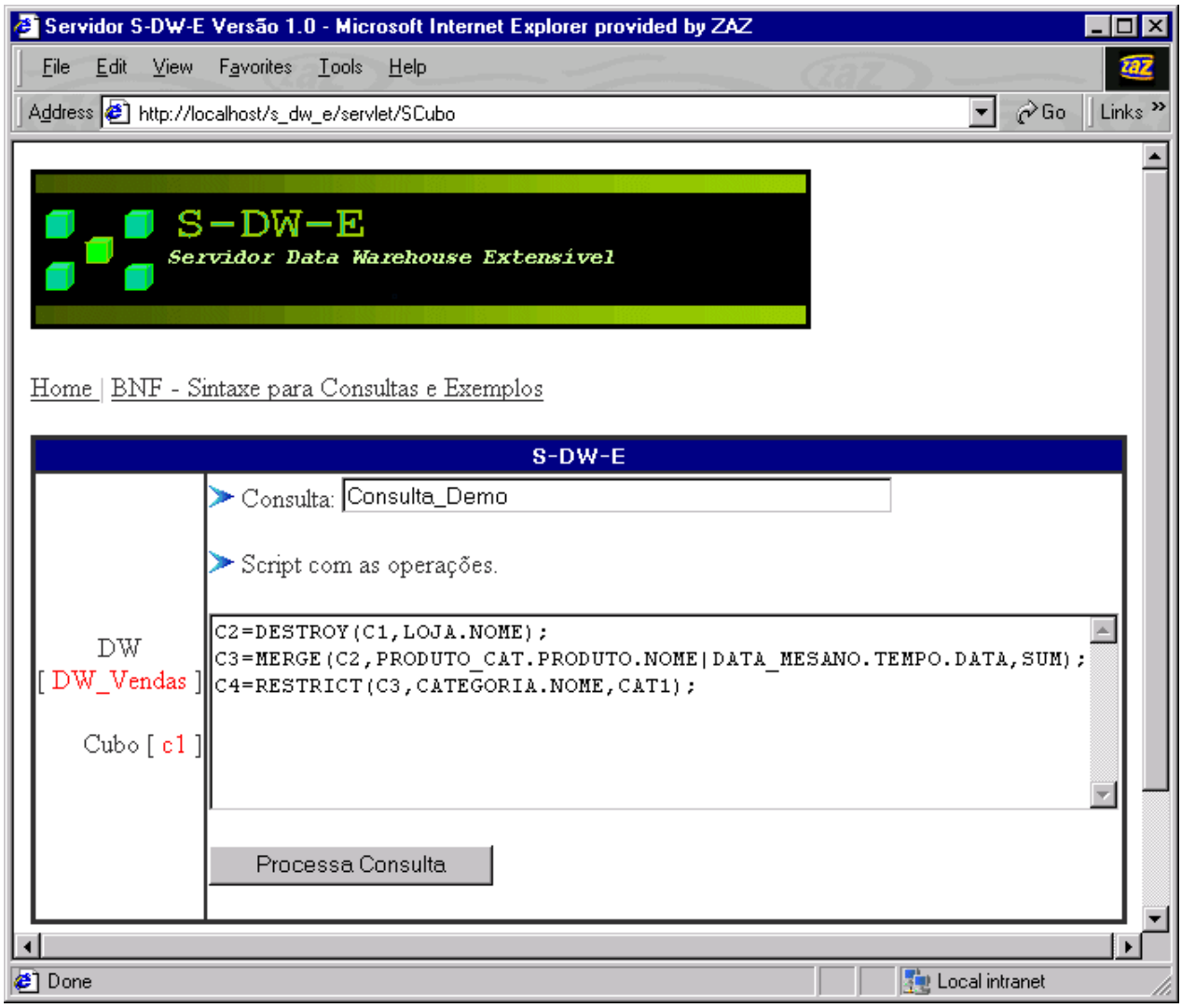


Figura 4.6: Interface do S-DW-E para o processamento de consultas *ad hoc*.

Caso haja algum erro, o usuário é reportado disto. Por exemplo "Operação <Restrict | Destroy | Merge > esperada linha=1", se o usuário escrever de forma equivocada *C2 = DESTRIOY(C1, LOJA. NOME);*

.

- Passo 4: Consultar os Cubos de Dados resposta.

  Após o S-DW-E processar a consulta, o usuário poderá obter os dados de qualquer um dos cubos de dados temporários (C2, C3,

C4), necessários à obtenção do cubo final (C4), como também os dados do cubo persistente C1. A figura 4.7 ilustra a obtenção dos dados do cubo C4.

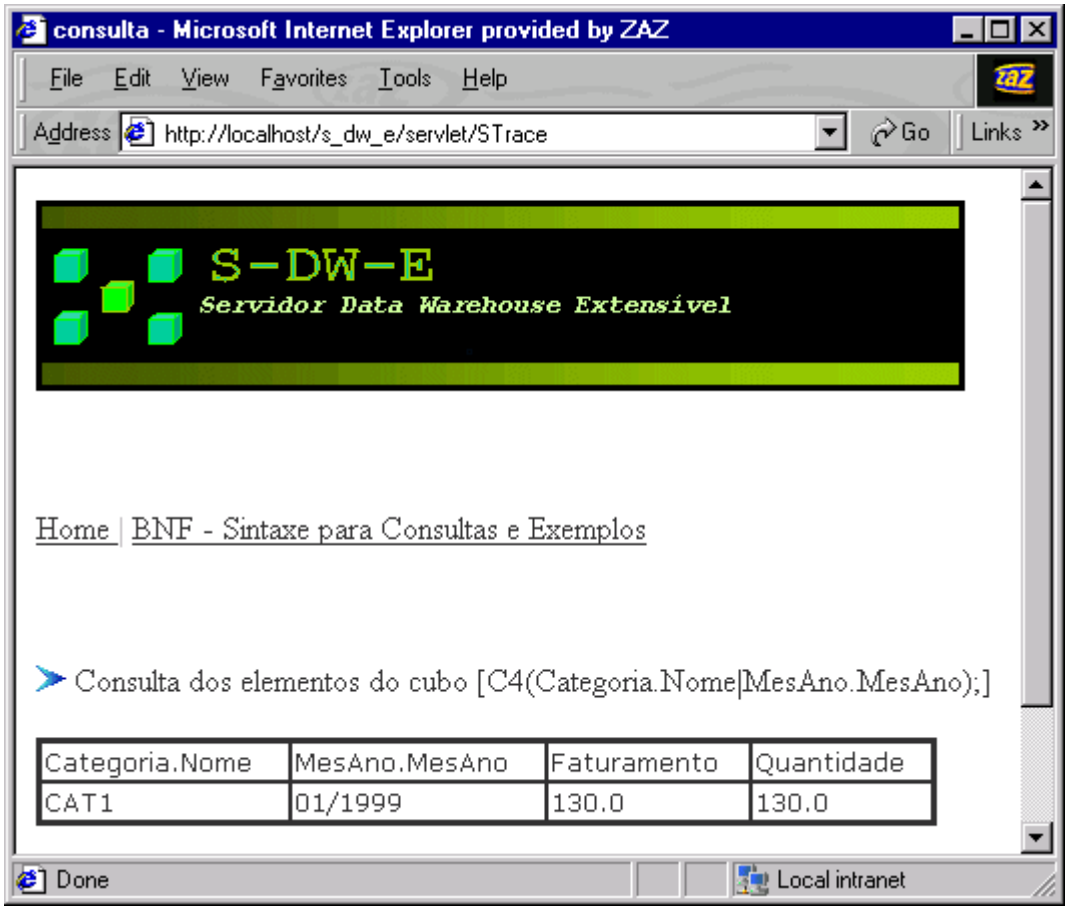


Figura 4.7: Tabela com os dados do cubo C4.

É importante observar que em nível de interface os quatro passos para a execução da consulta são os mesmos para os dois tipos de SGBD. Para o usuário final a interface é a mesma, ficando as alterações no âmbito interno, em que podemos ter um servidor operando com mais performance ou com mais recursos funcionais nas agregações.

CAPÍTULO V

O FRAMEWORK S-DW-E: PROJETO E IMPLEMENTAÇÃO

Iniciamos este capítulo apresentando o projeto arquitetural. Seguimos pormenorizando o projeto de baixo nível. Por fim, é apresentado um roteiro com os passos para a construção de novas extensões (novas tecnologias de gerencias de dados acoplado no S-DW-E).

5.1 Projeto Arquitetural

Para o nosso trabalho a construção do projeto arquitetural proporcionou uma visão da solução computacional em alto nível e a redução da complexidade do problema de construir um servidor OLAP extensível. A passagem por esta fase foi providencial, pois o S-DW-E era muito grande e complexo para ser desenvolvido como um sistema com um único módulo.

A arquitetura do S-DW-E — figura 5.1 — é implementada em três camadas (Interface, Aplicação e Dados) para atender aos requisitos não-funcionais enumerados e à arquitetura genérica definida no Capítulo IV seção 4.1.3.

A *camada de interface* é a camada que o usuário final usa para acessar e manipular o S-DW-E. Ela foi projetada para ser totalmente desacoplada da camada de aplicação, como rege o requisito *"[S1.1] 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 (Anexo III).

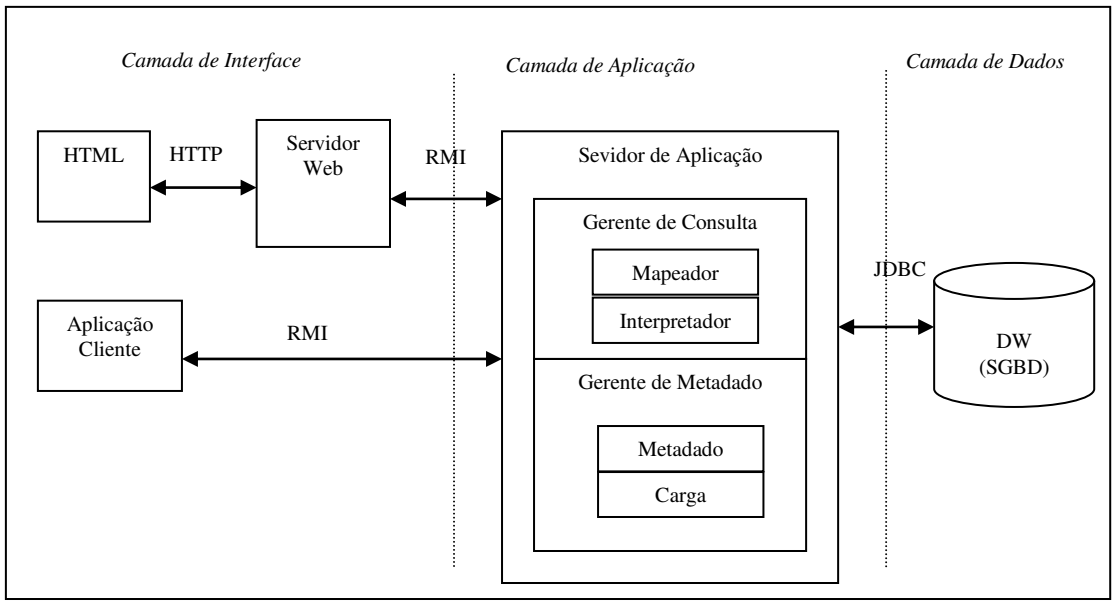


Figura 5.1: Projeto Arquitetural do S-DW-E.

Já a *camada de aplicação* contém todos os objetos com as regras de negócio e as operações para o funcionamento do S-DW-E. É nela que é feita a gerência do repositório de metadados. Devido à complexidade desta camada, esta foi fragmentada em dois módulos, *Gerente de Consulta e Gerente de Metadados*. Segue-se a descrição dos módulos (tabela 5.1).

Gerente de Consulta [Mapeador]	Módulo responsável por interpretar consultas OLAP no modelo conceitual (BNF das operações no Anexo I) e mapeá-las para a linguagem do SGBD concernente, acoplado no S-DW-E. Por exemplo, se o SGBD for o Microsoft SQLServer, então as operações OLAP são traduzidas para a linguagem SQL do SQLServer, de conformidade com os metadados que descrevem esse esquema lógico. No Anexo II, encontram-se alguns "templates" de mapeamento de operações OLAP.

Gerente de Consulta [Interpretador]	Módulo responsável pela a análise léxica e sintática das consultas efetuadas ao Servidor. Este módulo auxilia o módulo de Mapeador interpretando comandos texto ad hoc em componentes lógicos. Exemplo: Para o comando C2=Restrict(C1,Tempo.MesAno,01/1999,02/1999). São gerados os seguintes componente lógicos. • Cubo Entrada = C1; • Operação = Restrict • Cubo Saída = C2 • Dimensão Restrict = Tempo.MesAno • Filtro = "01/1999,02/1999".
Gerente de Metadado [Metadado]	Módulo que mantém as estruturas relativas aos metadados nos níveis: • Conceitual (DWs, Cubos, Dimensões, Elementos de Cubo, Elementos de Dimensão); • Lógico (Conexões com SGBDs, Esquemas de BD); • Físico (Índices, "Tablespace", Estatísticas sobre tabelas, etc) ; • Dependências entre os níveis conceitual, lógico e físico. 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).
Gerente de Metadado [Carga]	Módulo responsável por `povoar' as classes de definição de metadados nos níveis conceitual, lógico e físico. Os metadados são descritos em XML. A descrição dos arquivos DTD e XML encontram-se no Anexo IV.

Tabela 5.1: Módulos da Camada de Aplicação do Projeto Arquitetural do S-DW-E.

Por fim, na *camada de dados* estão os SGBDs onde os DWs correntes estão implementados. 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/performance.

5.1.1 Projeto Arquitetural Detalhado

Esta seção apresenta, também, o projeto arquitetural, mas com um nível de detalhe maior. Estão dispostos, além dos módulo já detalhados anteriormente, os pontos de extensão do framework S-DW-E.

A ilustração da figura 5.2 apresenta a relação existente entre os módulos do S-DW-E.

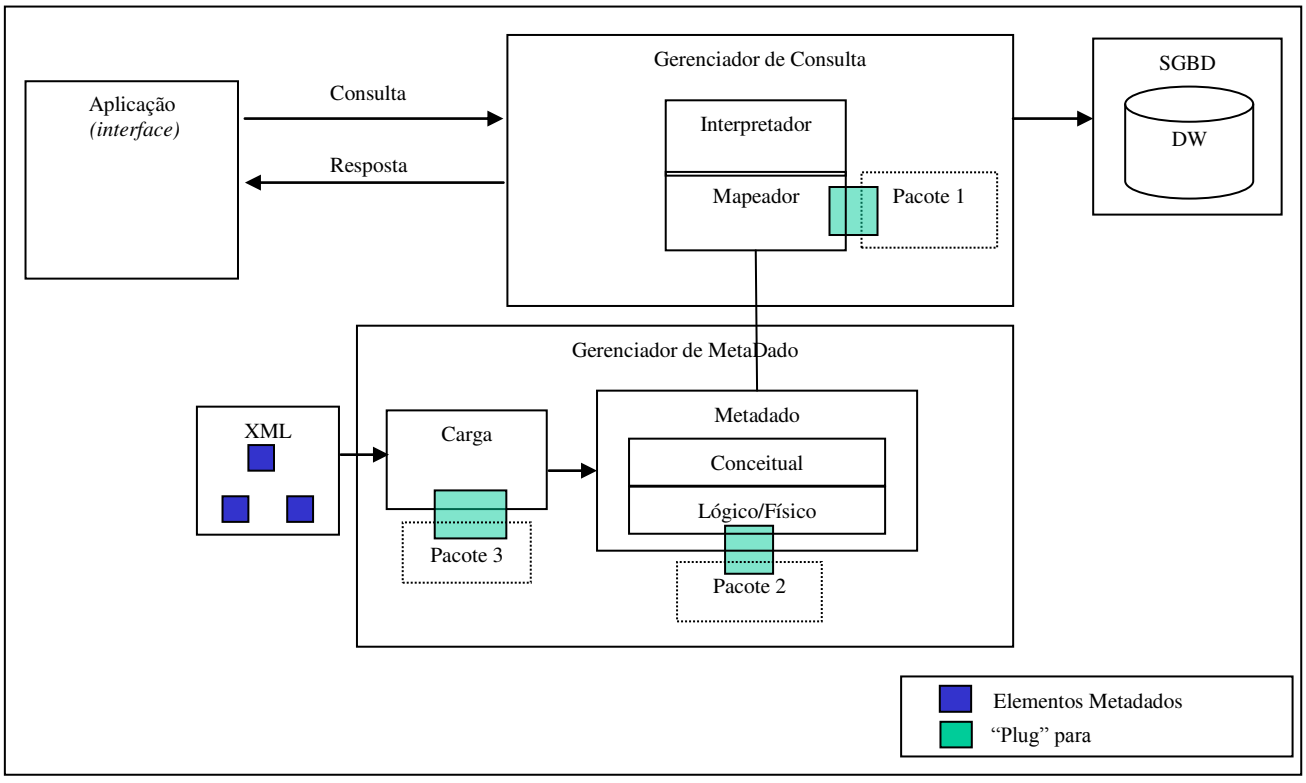


Figura 5.2: Projeto Arquitetural detalhado do S-DW-E.

O módulo de Gerente de Consulta é o módulo responsável por coordenar todas operações do servidor. Ele encapsula dois sub-módulos Mapeador e Interpretador que possuem um nível de acoplamento maior em relação aos módulos externos.

Internamente o Mapeador colabora com o Interpretador para interpretação das operações R. Agrawal (análise léxica e sintática). O módulo de Gerente de Consulta relaciona-se com o módulo Gerente de Metadado para consultar e gravar informações nos metadados conceitual, lógico e físico.

O módulo Gerente de Metadado encapsula todas funcionalidades relacionadas com a gerencia do metadados e é composto pelo sub-módulo de Metadados. Este módulo, possui ainda um sub-módulo de Carga com a função de ler arquivos XML (Anexo IV) com os metadados e carregá-los no módulo de Metadados.

Na figura 5.2 os *Pacotes* para extensão do framework estão representados por retângulos pontilhados. Os retângulos verdes são os blocos para encaixe ("plugs") das extensões. Podemos dividir o framework do S-DW-E em três frameworks menores: Mapeador, Metadado e Carga.

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 S-DW-E. Nos *Pacotes* estão um conjunto de classes que derivam classes e implementam funcionalidades de classes predefinidas do framework S-DW-E, estendendo assim, suas funcionalidades. Este processo visa acoplar no servidor uma determinada tecnologia de armazenamento.

Por exemplo, a criação de um tipo de S-DW-E para trabalhar com a tecnologia ROLAP e manipular o DW implementado no SqlServer Relacional deve ser capaz de mapear as operações escritas com base no esquema conceitual definido no S-DW-E para SQL. Para isso, serão utilizadas as informações do esquema lógico e o pacote Mapeador SqlServer Relacional.

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

- Extensão R-Microsoft SqlServer
    - *Pacote 1 - Mapeador Sqlserver R*
    - *Pacote 2 – MetaDado Lógico e Físico para Aplicações R*
    - *Pacote 3 – Carga Aplicações R*

- Extensão OR-Oracle
    - *Pacote 1 - Mapeador Oracle OR*
    - *Pacote 2 – MetaDado Lógico e Físico para Aplicações OR*
    - *Pacote 3 – Carga Aplicações OR*

Na próxima seção, descrevemos em detalhes o projeto do S-DW-E, particularmente o Gerente de Metadados e o Gerente de Consulta.

5.2 O Projeto de Baixo Nível

O projeto de baixo nível é composto por diversos diagramas que são definidos com o intuito de criar um modelo computacional extensível para a realização de consultas conceituais OLAP, segundo o modelo de Agrawal. Por questões didáticas, a apresentação do projeto de baixo nível do S-DW-E é apenas parcial.

O foco são os módulos Gerente de Metadados e o Gerente de Consultas. Iniciamos com a apresentação dos diagramas de classes dos dois módulos. Na seqüência, duas extensões do S-DW-E são descritas, correspondendo às implementações das operações OLAP no SQLServer e no Oracle8i, respectivamente. Por fim, exemplificamos o funcionamento interno do S-DW-E através de um estudo de caso de resolução de uma operação OLAP.

5.2.1 O Gerente de Metadados

No projeto de metadados temos os modelos com a solução computacional para a representação dos esquemas DW. Este projeto foi desenvolvido com base em quatro premissas (requisitos do metadados).

(r1) Captura de todas as necessidades de informações de projeto, para construção e operação de DW;

(r2) Integração entre os diferentes esquemas do metadados, incluindo interdependências;

(r3) Uso de uma representação uniforme baseada em UML;

(r4) Desenvolvimento de uma arquitetura base para o framework (pacotes de classes Java) baseado em herança para sua "customização";

O projeto do framework para os modelos de metadados é dividido em: conceitual, lógico e físico e suas interdependências. Nesse modelo ainda temos as extensões, ou seja, representação do modelo lógico e físico do servidor para uma determinada tecnologia OLAP (R-OLAP, OR-OLAP). No fim desta seção, detalhamos exemplos de duas extensões para o projeto de metadados: (f1) Metadado-R SqlServer 7; (f2) Metadado-OR Oracle 8i.

O diagrama de classe da figura 5.3 descreve o modelo de metadados conceitual do framework. Descrevemos os principais elementos do modelo.

1. A classe *Servidor* é uma classe que só possui uma instância ("singleton/façade"). Ela tem o objetivo de proporcionar uma interface com os subsistemas da camada de interface. Sua função principal é "encapsular" a navegação do modelo de metadados conceitual, isolando a complexidade para os subsistemas que a acessam.

   O objeto do tipo *Servidor* possui uma coleção de *DW_C*, que representa os diversos esquemas de DW implementados no servidor. Este objeto também contém uma coleção de funções que são usadas nas operações de "Merge" (soma() ,maximavenda(int mes1,int mes2), etc).

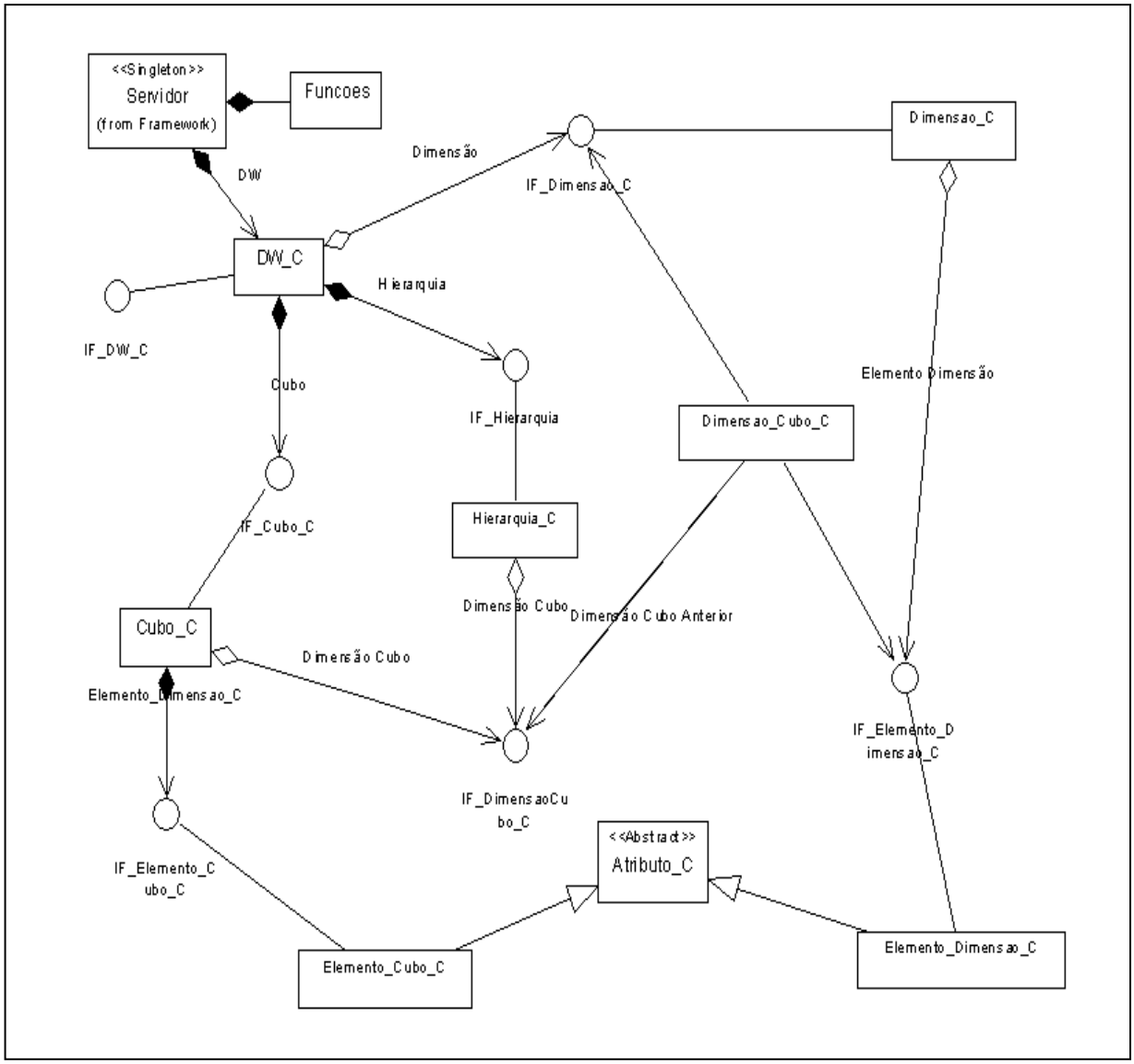


Figura 5.3: Diagrama de classes UML do modelo de metadados conceitual do framework.

2. 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 no capítulo 2.

3. 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.*

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

Como foi definido na premissa (r2) existe uma forte relação entre os modelos conceitual e lógico e lógico e físico. Veremos como é feita a ligação entre os modelos.

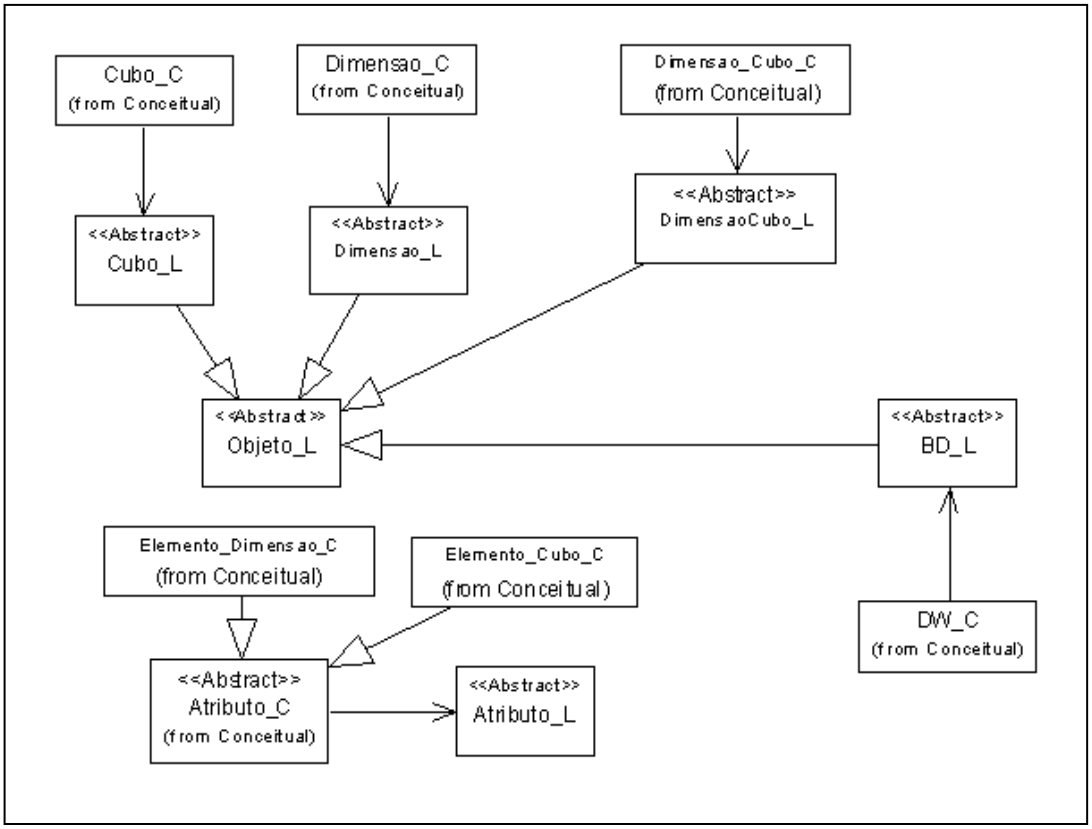


Figura 5.4: Diagrama de classes UML do modelo de metadados lógico do framework.

No diagrama da figura 5.4 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 os `ganchos' do framework para o acoplamento das futuras extensões para o metadados lógico.

No mapeamento do modelo lógico com o modelo físico, as classes do modelo lógico herdam da superclasse *L-Object* que tem um coleção de objetos do tipo *P-Object* como ilustra

a figura 5.5. A classe abstrata *P-Object* é também um `gancho´ do framework e será especificada de acordo com a aplicação que estende o framework.

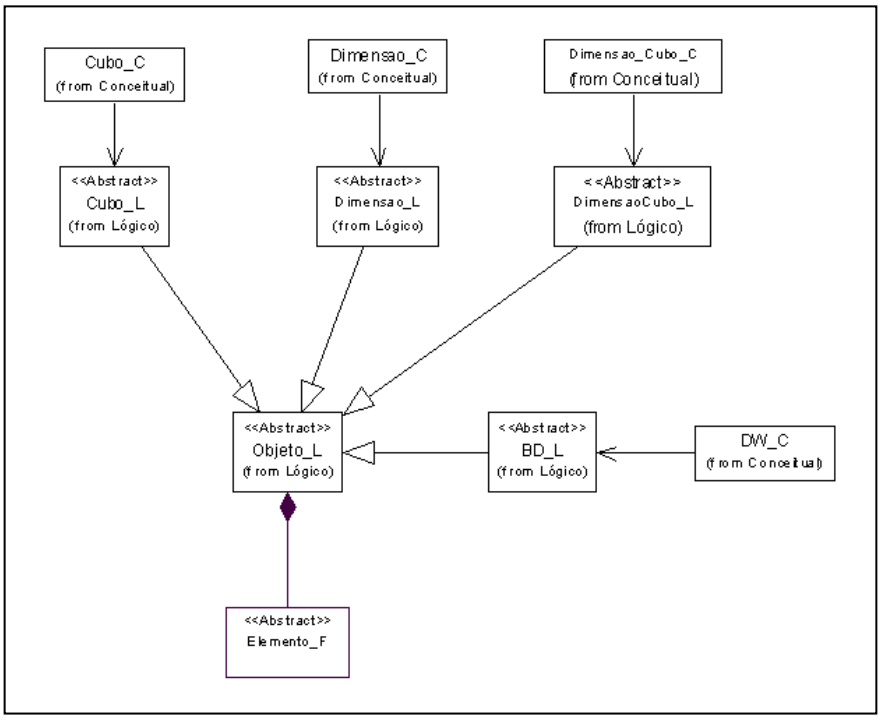


Figura 5.5: Diagrama de classes UML do modelo de metadados físico do framework.

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

(f1) Metadado-R SqlServer 7

O diagrama de classes da figura 5.6 apresenta um exemplo de extensão do modelo de metadados para representar esquemas DW em estrela relacional acolpado S-DW-E para o gerenciador SqlServer 7.

Conforme o diagrama da figura 5.6 são implementadas as classes *BDOdbc, CuboR, DimensaoR, DimensaoCuboR e AtributoR* que estendem respectivamente as classes *L-DW, L-Cubo, L-Dimensão e L-Atributo* do modelo lógico do framework.

As classes da extensão, pintadas na figura 5.6, 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 *P-Objetct*.

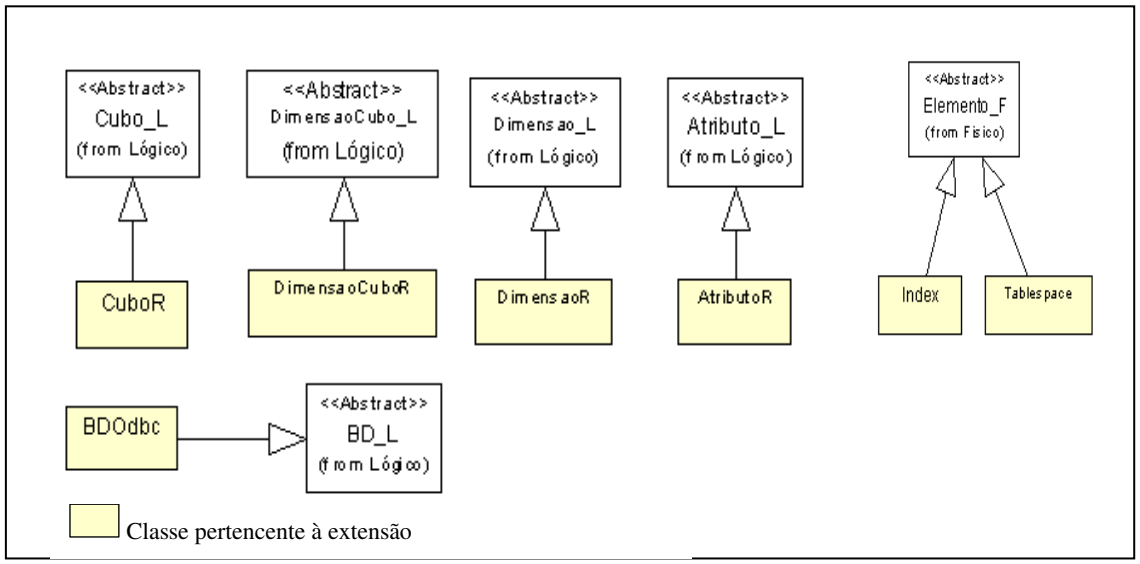


Figura 5.6: Diagrama de classes UML do modelo de metadados lógico e físico da extensão Metadado-R SqlServer 7.

(f2) Metadado-OR Oracle 8i

O diagrama de classes da figura 5.7 é muito semelhante ao da figura 5.6, com a diferença de ser mais simplificado. O motivo da simplificação deve-se a representação do esquema em estrela OR implementado no Oracle 8i. No nosso trabalho ele foi 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 fisco do framework.

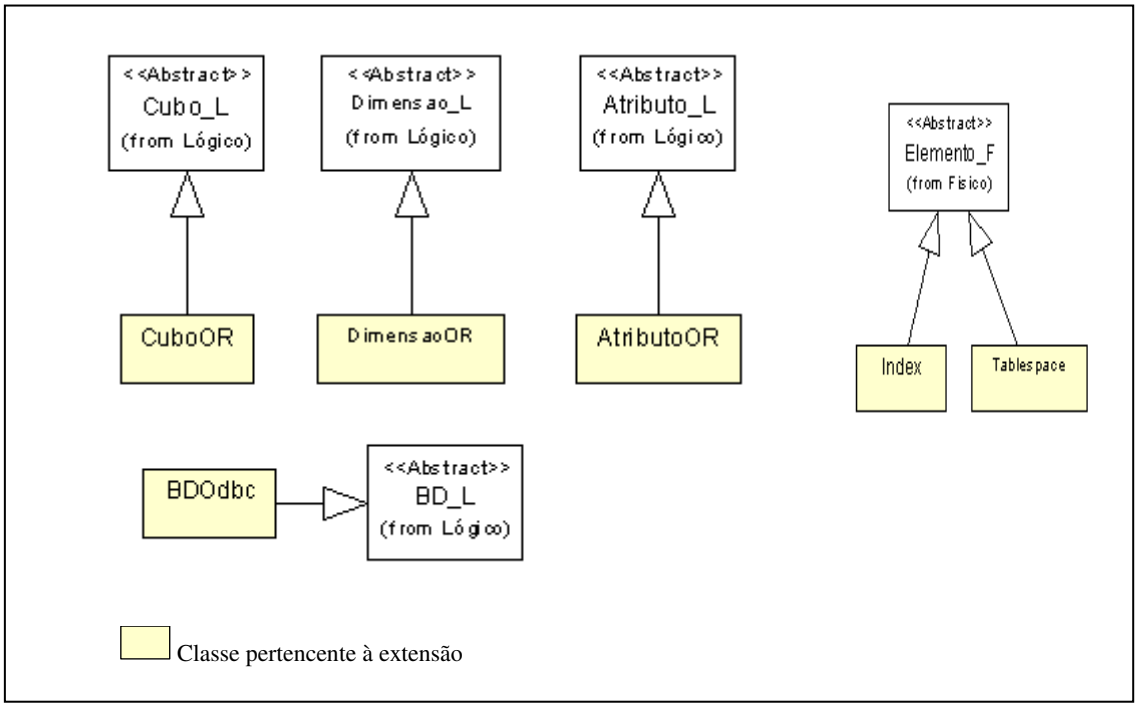


Figura 5.7: Diagrama de classes UML do modelo de metadados lógico e físico da extensão Metadado-OR Oracle 8i.

5.2.2 Gerente de Consulta

O Gerente de Consulta é o módulo que coordena todas as atividades do servidor. No processamento de uma consulta ele responsabiliza-se por gerenciar os diferentes módulos para as operações de: interpretar as operações Agrawal, gravar a definição dos novos cubos de dados no metadados e a criação física das visões dos novos *cubos de dados*. Além disso, é responsável pela clonagem da definição dos cubos no procedimento de geração dos cubos resposta ─ cubo de saída.

O diagrama de classes UML do modelo do mapeador do framework é ilustrado na figura 5.8.

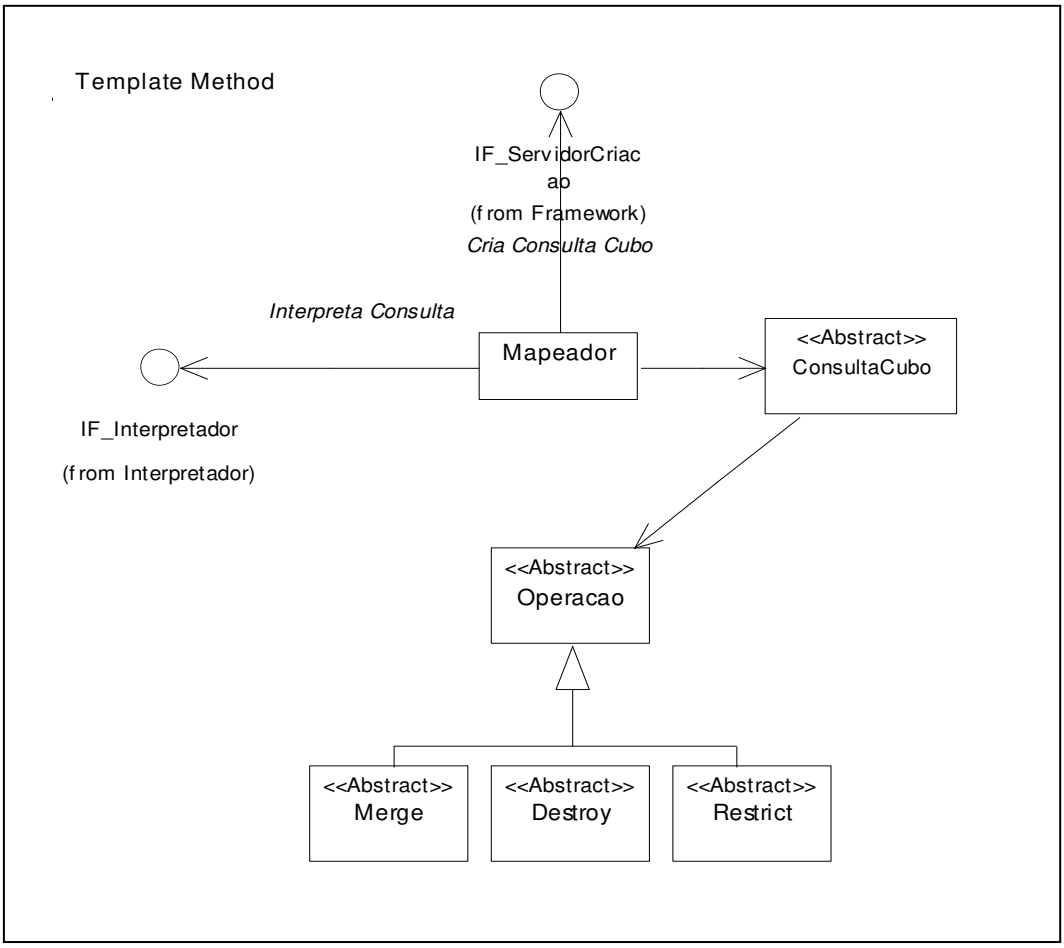


Figura 5.8: Diagrama de classes UML do modelo do Mapeador do framework.

Neste diagrama a classe *Mapeador* faz referência a uma interface *IF_Interpretador*. Esta interface é realizada no módulo Interpretador e possui a funcionalidade de interpretar comandos Agrawal. A classe *Mapeador* também faz referência a interface *IF_ServidorCriacao*. Este relacionamento possibilita que o Mapeador solicite ao módulo Metadado que crie a definição de novos *cubos de dados.*

A classe *Mapeador* coordena o procedimento de transformação das operações Agrawal na linguagem do gerenciador de banco de dados que o servidor está acolpado (referência à classe *ConsultaCubo*). Assim, procede-se a criação física do cubo de dados.

O procedimento para a construção física dos novos *cubos de dados* é parcialmente feito pela classe abstrata *ConsultaCubo*. Ela baseia-se no padrão de projeto "template method", explicitado no Capítulo III.

A classe “templateAbstrat” é a classe *ConsultaCubo* que define o esqueleto do algoritmo para construção de novos cubos de dados, transferindo alguns passos desta operação para a classe *Opercao*. A classe *Operacao* será implementada futuramente pelas extensões do Mapeador. Dependendo da operação a ser mapeada a classe *Operacao* pode ser de três tipos “Merge”, “Destroy” e “Restrict”, como mostra a figura 5.9.

O esqueleto do algoritmo da superclasse *ConsultaCubo* é estruturado da seguinte forma:

1. A definição do cubo de entrada é copiado (“clonado”) ;
2. A definição do cubo copiado (“clonado”) é transformada conforme operação, gerando toda a definição do cubo de saída;
3. A definição do cubo de saída é armazenada no metadados;
4. O método `template´ abstrato *runOperacao()* é acionado;

No algoritmo o método `template´ *runOperacao()* é implementado pelas subclasses das futuras extensões da classe *Operacao* . A figura 5.9 exemplifica um diagrama de classes com o modelo da extensão para Mapeador OR.

As classes *RestrictOR, DestroyOR e MergeOR* são os TemplatesConcretos e implementam o método definido na superclasse *Operacao*.

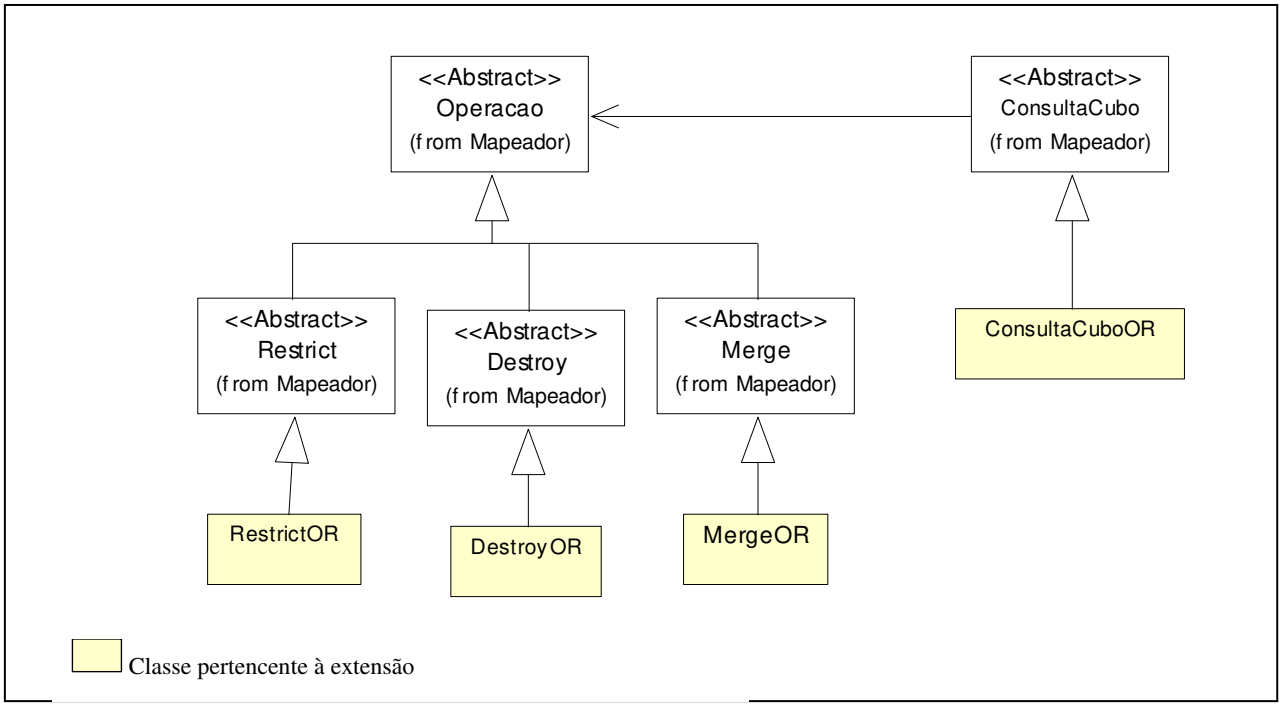


Figura 5.9: Diagrama de classes UML do modelo de Mapeador da extensão Mapeador-OR Oracle 8i.

O objetivo do método template *runOperacao()* é montar um "script" para ser executado no gerenciador acolpado. Cria-se, assim, fisicamente a visão do cubo de dados. Segue um exemplo de template para a classe *MergeOR* implementada no Oracle 8i.

1. *Create Type* [CUBO_SAIDA_TYPE] as *Object(* [REF_DIMENSAO] + [ELEMNTOS_CUBO] *);*
2. *Create Table* [CUBO_SAIDA] of [CUBO_SAIDA_TYPE];
3. *Insert into* [CUBO_SAIDA] *Select distinct* [REF_ELEMENTOS_DIMENSAO] + [ELEMNTOS_CUBO_SUM] FROM [CUBO_ENTRADA] *Group by* [REF_DIMENSOES_GROUPBY];

Os elementos entre colchetes são obtidos a partir do metadado lógico do cubo de saída. O Anexo II contém a descrição completa dos três "mapeadores" implementados no nosso trabalho.

Por fim, temos ainda a classe *ConsultaCuboOR* que estende a classe abstrata *ConsultaCubo* do framework. Essa estrutura também é uma representação do padrão

“template method”. E tem como funcionalidade criar o mecanismo de execução do “script” no banco de dados.

5.3 Estudo de Caso: Resolução de Uma Operação OLAP

Nesta seção apresentamos o funcionamento interno do S-DW-E para a resolução da operação de “*Destroy”*. Ilustramos como os módulos se comunicam e cooperam entre si através da realização da operação A operação será executada no esquema DW Vendas apresentado no Capítulo IV seção 4.7.

A figura 5.10 ilustra a interface *ad hoc* utilizada pelo usuário para a realização da consulta.

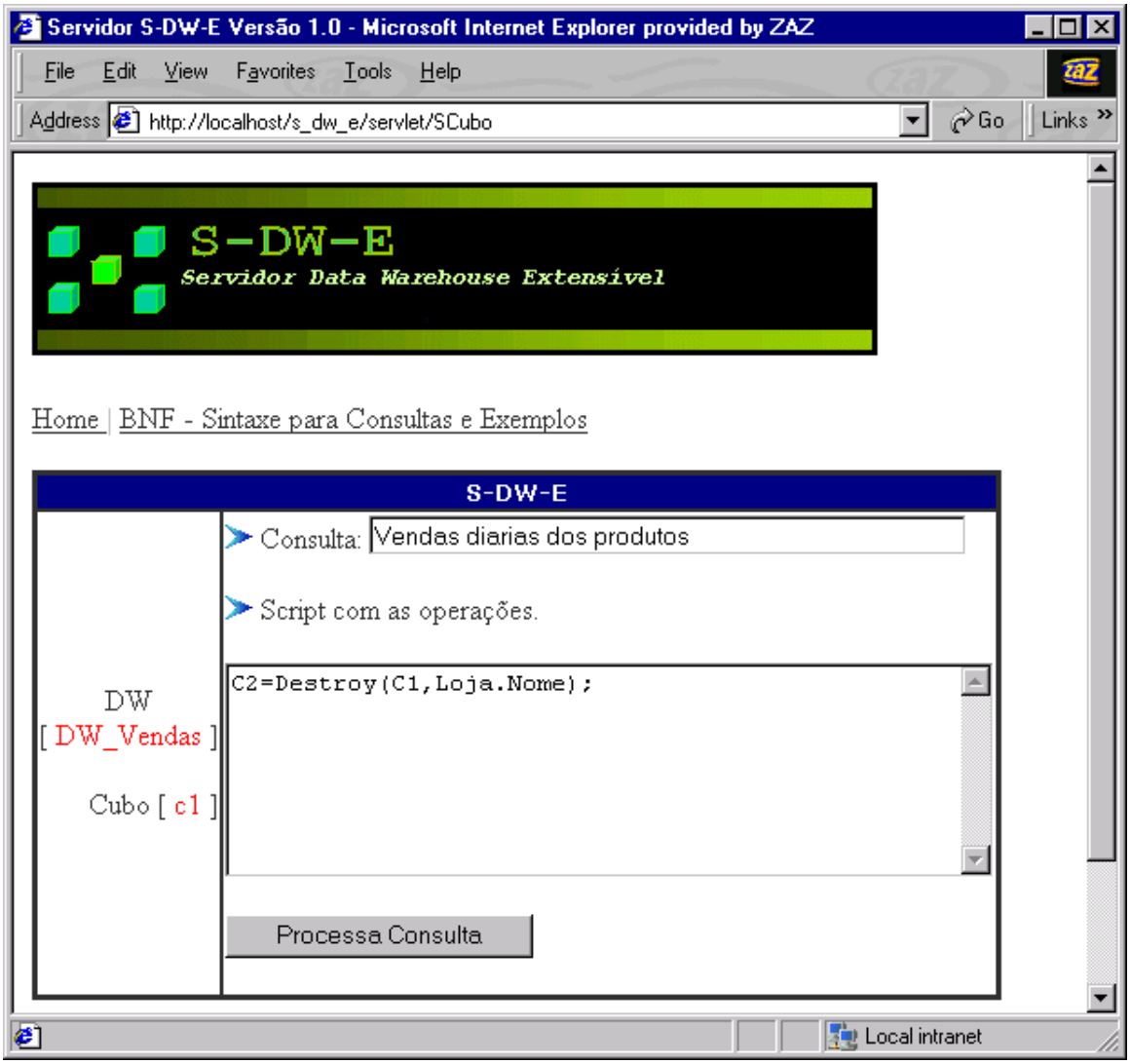


Figura 5.10: Interface do S-DW-E para o processamento de consultas "*ad hoc*".

A figura 5.11 mostra um esquema gráfico do processo do funcionamento interno do S-DW-E para essa operação. Por questões de simplificação, não tratamos das situações de exceção.

Os 5 passos do processo são detalhados em seguida.

(1) O módulo Interface capta do usuário as informações necessárias para a resolução da operação:

- DW (DW_Vendas);
- Consulta (Vendas diárias dos produtos.);
- C2=Destroy(C1,Loja.Nome);

Os dados captados são repassados para o módulo Mapeador.

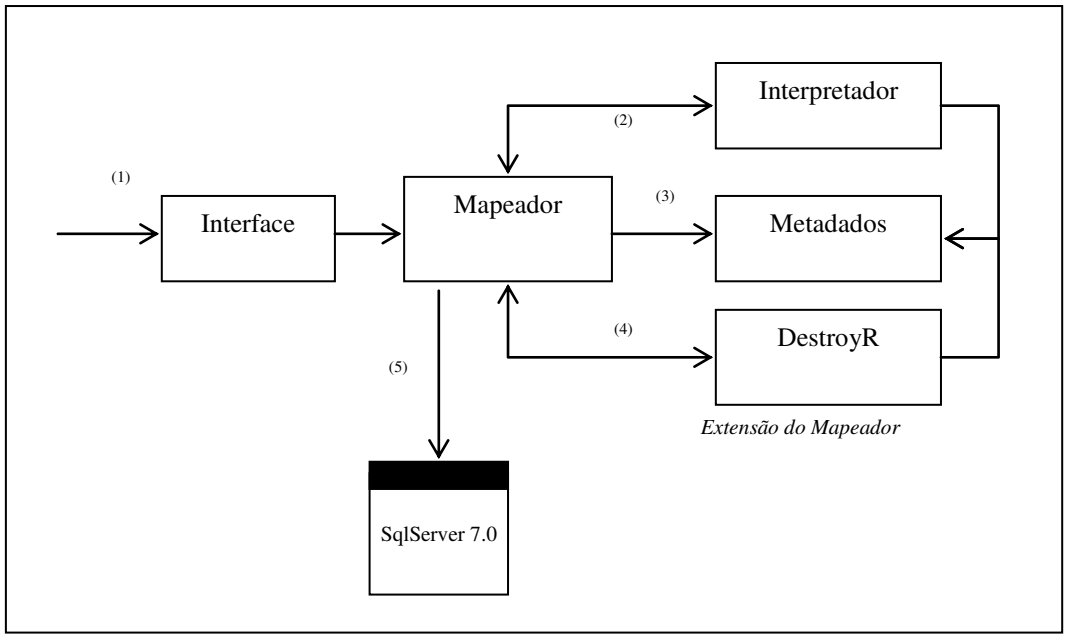


Figura 5.11: Esquema de Funcionamento do S-DW-E: Resolução de uma operação “Destroy”.

(2) O módulo Mapeador solicita uma verificação léxica e sintática da consulta ao módulo Interpretador de comandos. O Interpretador consulta o esquema de metadados no nível conceitual, que contém informações relativas ao cubo C1. Por fim, ele retorna para o Mapeador o comando em questão “particionado” nos seguintes componentes l:

- Cubo Entrada = C1;
- Operação = Destroy
- Cubo Saída = C2
- Dimensão Destroy = Loja

(3) O Mapeador cria a estrutura de definição do cubo de saída, C2, e solicita aos Metadados para gravá-la no repositório de metadados. A figura 5.12 ilustra esse processo. O Mapeador gera um novo C1’ — estrutura pertencente ao metadados e não possui dados — que possui a mesma estrutura (Dimensões e Elementos) do cubo C1. Ou seja, um clone de todos os objetos que compõem o cubo C1. A operação “Destroy” é então executada sobre o clone C1’ que é

transformado no cubo C2. Após essa tarefa, são adicionadas ao esquema de objetos do metadados informações sobre o cubo C2 para os esquemas: Conceitual, Lógico e Físico, conforme o Gerente de Metadados descrito neste Capítulo. Este processo se dá pela mudança do estado dos objetos que compõem o cubo C2.

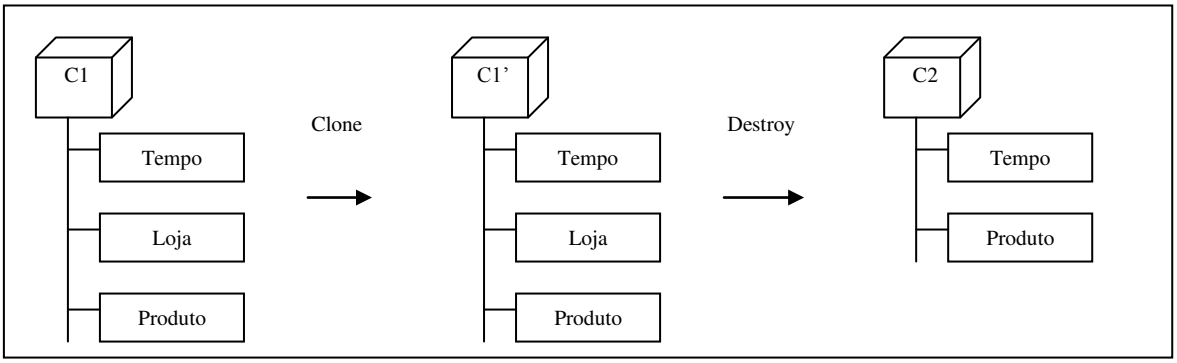


Figura 5.12: Processo de criação do metadados do cubo C2

Após o processo o metadados lógico do Cubo C2 tem a seguinte descrição:
"Tabela de Fato de Nome: C2consulta102
Atributos: ID_Tempo,ID_Produto, Quantidade, Faturamento "

(4) Mapeador aciona a extensão DestroyR, que está acoplada no servidor com o "template" (Anexo II) para mapear operações "Destroy" em comandos SQL. São passados para o classe DestroyR todos os componentes lógicos interpretados no passo (3). O módulo em questão consulta o esquema de metadados lógico do cubo C2 para o mapeamento da operação "Destroy". Por fim, é repassado para o Mapeador o script com os comandos SQL. Segue o comando gerado pelo mapeador:

"select dwvendas.dbo.Tempo.id as ID_Tempo,dwvendas.dbo.Produto.id as ID_Produto,sum(distinct faturamento) as faturamento,sum(distinct quantidade) as quantidade into TempDB.dbo.C2consulta102 From dwVendas.dbo.Vendas,dwvendas.dbo.Tempo,dwvendas.dbo.Produto Where dwVendas.dbo.Vendas.ID_Tempo=dwVendas.dbo.Tempo.ID and

dwVendas.dbo.Vendas.ID_Produto=dwVendas.dbo.Produto.ID GROUP BY dwvendas.dbo.Tempo.id,dwvendas.dbo.Produto.id"

(5) Ao final de todo o processo, o Mapeador executa o script SQL no SGBD relacionado com o DW Vendas informado pelo usuário no passo (1) ─ DW Vendas associado com SqlServer. O gerenciador processa o script, criando fisicamente uma visão dos dados que o usuário deseja consultar.

5.4 Passos para a Extensão do Framework

Para extensão do framework o desenvolvedor necessita conhecer o projeto conceitual do framework e sua API (Anexo III). Ressalta-se que o desenvolvedor não necessita conhecer nenhuma implementação interna do framework. Listamos os passos para extensão do S-DW-E.

1. Compreender o Modelo de Agrawal.

   Compreender o Modelo Multidimensional de Agrawal nas suas estruturas (Cubo, Dimensão, etc), bem como para seus operadores ("Restrict","Destroy","Merge")

2. Compreender a análise e o projeto do Framework através dos seguintes elementos da documentação:
   - Objetivo do Sistema;
   - Requisitos funcionais e Diagrama de Caso de Uso;
   - Requisitos não-funcionais;
   - Projeto Arquitetural (Macro e Detalhado);
   - Diagrama de Classes fase de Análise;
   - Diagrama de Classes fase de Projeto;
3. Projetar e implementar o Diagrama de Classe para o Modelo Lógico:

   Todas as classes do Modelo Lógico devem possuir atributos e métodos que possibilitem obter informações para fazer o mapeamento para a extensão definida. Deve-se definir:

   - Uma classe que faça o mapeamento do DW Conceitual para o Gerenciador acoplado no Framework. Para isso, esta classe deve estender a classe *BD_L*;

- Métodos que devem ser implementados:
  - public void setConexao() throws Exception; (Método responsável por fazer a conexão com o Banco).
  - public Connection getConexao();(Método que retorna o objeto Conn que contém a conexão com Banco de Dados).

- Uma classe que faça o mapeamento da Dimensão Conceitual para uma Dimensão Lógica. Para isso, esta classe deve estender a classe *Dimensão_L*;
  - Métodos a serem ser implementados:
    - public String getDescricaoDimensao_L(); (Método que retorna a descrição da Dimensão lógica)
- Uma classe que faça o mapeamento do Cubo Conceitual para um Cubo Lógico. Para isso, esta classe deve estender a Classe *Cubo_L*;
  - Métodos a serem ser implementados:
    - public Object clone(); (Método que retorna um objeto igual ao objeto em questão)
    - public String getDescricaoCubo_L();(Método que retorna a descrição do Cubo lógico)
- Uma classe, se necessário, que represente a relação entre o Cubo e a Dimensão Conceitual logicamente. Para isso, esta classe deve estender a Classe *DimensaoCubo_L*;
  - Métodos a serem implementados:
    - public Object clone(); (Método que retorna um objeto igual ao objeto em questão)
- Uma classe que faça o mapeamento do Atributo Conceitual num Atributo Lógico. Para isso, esta classe deve estender a classe *Atributo_L*;
  - Métodos a serem implementados:
    - public Object clone(); (Método que retorna um objeto igual ao objeto em questão)
    - public String getDescAtributo_L();(Método que retorna a descrição do Atributo lógico)

4. Projetar e implementar um Diagrama de Classe para o Modelo Físico.

As classes do modelo lógico que vão representar a Dimensão, Cubo e a relação do Cubo com a Dimensão possuem uma coleção de objetos físicos. Assim, neste pacote você deve criar todos as classes que representarão fisicamente os objetos lógicos. Para isso, estas classes devem estender a classe *Objeto_L*;

5. Projetar e implementar um Diagrama de Classe para o pacote Mapeador
   Este pacote terá a função de mapear as funções Agrawal na linguagem de consulta do gerenciador que será acoplado no framework. Deve-se definir:
   - Uma classe que receberá as declarações das operações mapeadas, e as processará no servidor de Banco de Dados acoplado. Além disso, esta classe conterá métodos que possibilitem a apresentação das informações dos Cubos processados. Esta classe deve estender a classe *ConsultaCubo* do framework(Pacote Mapeador).
     - Métodos a serem implementados:
       - public ConsultaCuboR(String nome)throws Exception (O construtor deve ser implementado e passar o nome para a Classe pai. Exemplo: super(nome))
       - public Operacao criaObjRestrict() (Método responsável por criar um objeto “restrict” da tecnologia “plugada”. Exemplo: return new RestrictR();)
       - public Operacao criaObjDestroy() (Método responsável por criar um objeto “destroy” da tecnologia “plugada”. Exemplo: return new DestroyR();)
       - public Operacao criaObjMerge()(Método responsável por criar um objeto “Merge” da tecnologia “plugada”. Exemplo: return new MergeR();)
       - public void processaCubo()throws Exception (Método que processa os comandos na linguagem do gerenciador acoplado. Este método será chamado pelas classe de operação (“Restrict”, ”Destroy”, “Merge”)
       - public Iterator getConsulta(Cubo_C c)throws Exception (Método que dado um Cubo Conceitual é retornado as informações contidas nesse cubo na forma de tabela.)

- Uma classe que fará a operação de “Restricty” para qualquer consulta. Esta classe conterá um “template” do comando necessário para executar uma restrição de forma genérica. Esta classe deve estender a classe *Restrict* do framework(Pacote Mapeador).
    - Métodos a serem implementados:
        - public Cubo_L runOperacao()throws Exception (Método que contém um “template” para fazer a operação de “restrict”)
- Uma classe que fará a operação de “Destroy” para qualquer consulta. Esta classe conterá um “template” do comando necessário para executar redução das dimensões do Cubo de forma genérica. Esta classe deve estender a classe *Destroy* do framework(Pacote Mapeador).
    - Métodos a serem implementados:
        - public Cubo_L runDestroyExtensao()throws Exception (Método que contém um “template” para fazer a operação de “destroy”)
- Uma classe que fará a operação de “Merge” para qualquer consulta. Esta classe conterá um “template” do comando necessário para executar uma troca de dimensões do Cubo de forma genérica (mudança de granularidade). Esta classe deve estender a classe Merge do Framework(Pacote Mapeador).
- Métodos a serem implementados:
    - public IF_MetaCubo runOperacao()throws Exception (Método que contém um “template” para fazer a operação de “merge”)

Como instrumento auxiliar, pode-se consultar também o projeto das duas extensões implementadas no nosso trabalho.

CAPÍTULO VI

CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Neste capítulo, apresentamos as considerações sobre como desenvolver um ambiente de DWing extensível à luz da técnica de projeto de software, Framework. Iniciamos apresentando os resultados e as contribuições obtidos com a nossa pesquisa. Seguimos com a análise crítica sobre alguns resultados obtidos. Por fim, indicamos algumas sugestões de trabalhos futuros.

O estudo ao longo do nosso trabalho tratou da pesquisa de um ambiente de DWing extensível. O desafio foi construir um servidor OLAP que, através de um gerente de metadados formal e extensível, estivesse apto a ser, facilmente, adaptado a qualquer tecnologia de SGBDs.

Levamos em conta também a inexistência de um modelo de gerência de metadados para sistemas de DWing que seja aceito verdadeiramente como padrão.

A nossa pesquisa parece inovadora, uma vez que trabalhos relacionados limitam-se a projetos de ferramentas OLAP com modelos de metadados acoplados a uma determinada tecnologia, ou seja, ferramentas não extensíveis.

Como primeiro resultado, temos a definição de uma arquitetura genérica para ambiente de DWing extensível. A arquitetura apresenta todos os elementos que compõem esse ambiente, suas relações e responsabilidades.

Devido ao tamanho e à complexidade de projetar uma solução completa para a arquitetura, delimitamos o escopo do projeto. O foco foi a construção de um servidor que resolve consultas OLAP em DWs implementados nas tecnologias ROLAP e OR-OLAP.

O coração do servidor é o seu repositório de metadados extensível baseado num modelo conceitual formal, modelo de R. Agrawal [ARG99]. Através do modelo formal geramos uma interface comum para servidores OLAP. Portanto, o servidor tem como principais características ser um software flexível e reutilizável.

A construção de uma solução computacional com essas características nos fez pesquisar uma solução adequada para tratar este tipo de problema. Projetar um software reutilizável é uma missão dura. Assim, utilizamos a técnica de projeto de software, Framework, associada com padrões de projeto e orientação a objetos como a principal ferramenta para a definição e construção de servidores de DW, que batisamos de S-DW-E.

A captura das funcionalidades comuns a todos os tipos de servidores de DW foram encapsuladas no framework S-DW-E. A condição para a sua viabilidade é que a interface OLAP seja padronizada, o padrão sendo o modelo conceitual formal de R. Agrawal.

As extensões acopladas ao framework são responsáveis apenas por adaptar o S-DW-E a trabalhar com diferentes tecnologias de DW (R-OLAP, OR-OLAP, MOLAP, etc). O desenvolvedor que desejar criar um novo DW no S-DW-E deverá somente concentrar-se nos aspectos específicos da sua extensão. Todas as funcionalidades presentes no S-DW-E são reutilizadas.

No processo de construção do S-DW-E criamos duas aplicações (R-OLAP SqlServer e OR-OLAP Oracle). As aplicações validaram o framework S-DW-E e, com base nelas, pudemos verificar o percentual de reutilização alcançado. Em nível de código, a reutilização alcançada chega a atingir aproximadamente 70%, ou seja, as extensões representam somente

30% do código. Foram reutilizadas também as documentações da análise e do projeto do framework S-DW-E.

A construção das extensões (aplicações) proporcionaram também uma experiência prática para o uso da tecnologia objeto-relacional em uma ferramenta OLAP. Podemos atestar que os recursos objeto-relacionais aplicados no servidor S-DW-E facilitaram a implementação de um DW, principalmente na resolução de consultas complexas. No nosso trabalho não foi feito nenhum estudo comparativo de desempenho entre tecnologias distintas.

Por fim, destacamos 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. Detalhamos os pontos relevantes desse processo e expomos exemplos vivenciados na prática no processo de construção do S-DW-E.

Apesar da abordagem de framework ter ajudado a resolver o nosso problema inicial, foram encontradas algumas dificuldades. Enumeramo-las:

- Mesmo com todas as técnicas utilizadas, desenvolver uma aplicação reutilizável ainda é uma tarefa árdua. No entanto, é compensatório quando pensamos na reutilização alcançada.
- O custo em desenvolver um framework pode ser muito alto, visto que, no nosso projeto foi necessário desenvolver duas aplicações antes de iniciar o projeto do framework.
- Para o desenvolvedor, a criação de novas extensões no S-DW-E demanda um esforço e um conhecimento muito grande do projeto do framework. Esta dificuldade pode ser minorada com o uso de framework “Black Box”[ROB00].
- Apesar da reutilização alcançada, o S-DW-E é muito dependente da solução implementada nas extensões. Portanto, se a extensão não for implementada corretamente toda a solução pode ficar comprometida.

Para dar continuidade a este trabalho são sugeridos os seguintes estudos:

- A criação de um gerente de metadados para a gerência da persistência. Na solução atual, os metadados estão em arquivos XML que são carregados na

inicialização do S-DW-E. Existe a necessidade de projetar uma solução que utilize um SGBD para fazer a persistência das informações dos metadados, ou seja, gerenciamento de informações dos modelos, esquemas e interdependências. Deve-se tratar também das representações de informações de extração, consistência, carga, entre outras não tratadas no nosso servidor.

- Construção de novas extensões com base em tecnologias OO-OLAP. A criação de uma extensão do S-DW-E com tecnologia OO permitirá fazer uma avaliação dos SGBDOO para DWing. Os SGBDs OO são relativamente pouco amadurecidos, devendo-se fazer pesquisas para comprovação das vantagens em utilizar esses tipos de SGBDs para DWing.
- Construção de uma interface amigável é um dos requisitos do ambiente de DWing. Os usuários (gerentes) devem ser capazes de realizar, de forma fácil e intuitiva, consultas para a tomada de decisão. Assim, faz-se necessário a construção de uma ferramenta de "front-end" que utilize interfaces gráficas com recursos de "click" e "drag & drop". Na versão atual do S-DW-E é disponibilizada somente uma interface "adhoc". Esta ferramenta deverá utilizar a API do S-DW-E (Anexo III) para acoplar-se ao servidor S-DW-E.

ANEXO I

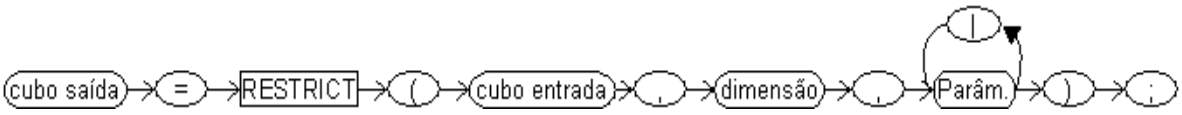
BNF S-DW-E DOS OPERADORES OLAP

Este anexo apresenta a definição formal dos comandos R. Agrawal para realização de consultas OLAP no servidor S-DW-E. O padrão gráfico da BNF do S-DW-E segue a mesma estrutura da SGBD Oracle. Por fim, é apresentada a resolução de uma consulta exemplo.

1. Sintaxe das Operação do S-DW-E (BNF)

Restrict

O operador opera sobre uma dimensão de um cubo, removendo os valores da dimensão que não satisfazem a condição estabelecida.

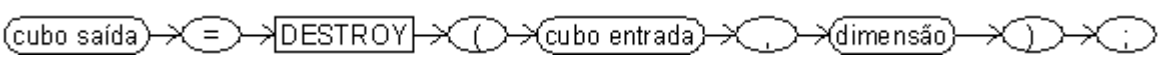


Exemplo:

C2 = RESTRICT(C1,Tempo.Ano,1999|2000);

Destroy

Operador usado para reduzir o número de dimensões.

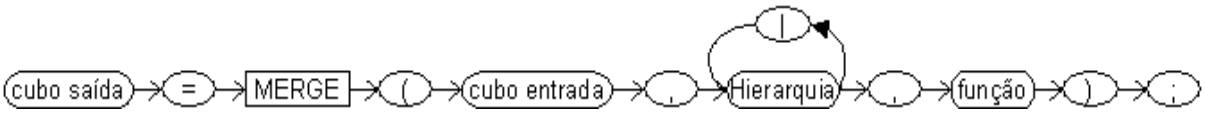


Exemplo:

C5 = DESTROY(C2,Produto.Nome);

Merge

A operação de agregação do Cubo. Está operação promove uma mudança de "granularidade" do Cubo conforme as hierarquias indicadas. Os elementos do cubo serão transformados de acordo com a função também indicada.

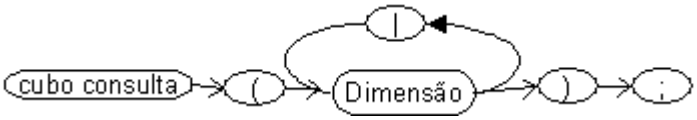


Exemplo

C3=MERGE(c1,Produto_Cat.Produto.Nome|Data_MesAno.Tempo.Data,sum);

Consulta Cubo

Esta operação permite ao usuário consultar informações de um Cubo.



Exemplo

C1(tempo.Data | Produto.Nome | Loja.Nome | Loja.Estado);

Regras da BNF

Elementos da Consulta	**Tipo**	**Restrições**	**Exemplo**

Cubo Entrada	Literal	Para um cubo ser referenciado como de entrada ele anteriormente já deve ter sido criado.	Faturamento_Loja
Cubo Saída	Literal	O cubo de saída não pode ter o mesmo nome que um cubo já existente na consulta.	Crescimento_Percentual
Dimensão	Literal	As dimensões referenciadas devem pertencer ao cubo de entrada.	Tempo.Ano | Loja.Nome | ...
Parâmetro	Literal	O conjunto de valores que se deseja passar como parâmetro deve ter o mesmo domínio da dimensão.	1999 | 2000
Função	Literal	Função deve existir na biblioteca disponibilizada no Servidor. O parâmetro passado deve ter o mesmo domínio da especificação da função.	CrescimentoP|1999|2000

2. Exemplo de Uma Consulta Para o Esquema Demonstração DW Vendas

Qual o faturamento e a quantidade das vendas mensais para os produtos relacionados com à Categoria 'Cat1'?

Resposta:

Consulta: Consulta_Demo

C2=DESTROY(C1,LOJA.NOME);
C3=MERGE(C2,PRODUTO_CAT.PRODUTO.NOME|DATA_MESANO.TE
MPO.DATA,SUM);
C4=RESTRICT(C3,CATEGORIA.NOME,CAT1);

Consultado Resultado

C4(Categoria.Nome|MesAno.MesAno);

ANEXO II

TEMPLATES DE MAPEADORES DAS EXTENSÕES S-DW-E

Segue a descrição dos "templates" para as extensões do Gerente de Consultas do S-DW-E. Na versão atual do servidor foram desenvolvidos três "templates" para construção dos *Mapeadores* das extensões do S-DW-E:

- Template para Mapeador Relacional_SQLServer 7
- Template pare Mapeador Relacional_Oracle 8i
- Template para Mapeador Objeto-Relacional_Oracle 8i

"Templates" para o Mapeador Relacional_SQLServer 7	
Operação	Template
Destroy	*Select* [ELEMENTOS_DIMENSAO_MENOS_A_DIMENSAO_REMOVIDA] + [ELEMENTOS_CUBO_SUM] *into* [CUBO_SAIDA] *From* [CUBO_ENTRADA] *Where* [JUNCAO_DAS_TABELAS_DE_DIMENSAO_FATO] *Group by* [DIMENSOES_GROUPBY];
Restrict	*Select* [ELEMENTOS_DIMENSAO] + [ELEMENTOS_CUBO_SUM] *into* [CUBO_SAIDA] *From* [CUBO_ENTRADA] *Where* [JUNCAO_DAS_TABELAS_DE_DIMENSAO_FATO] *Group by* [DIMENSOES_GROUPBY];
Merge (sum)	*Select* [ELEMENTOS_DIMENSAO] + [ELEMENTOS_CUBO_SUM] *into* [CUBO_SAIDA] *From* [CUBO_ENTRADA] *Where* [JUNCAO_DAS_TABELAS_DE_DIMENSAO_FATO] *Group by* [DIMENSOES_GROUPBY];

"Templates" para o Mapeador Relacional_Oracle 8i	
Operação	Template
Destroy	*Create Table* [CUBO_SAIDA] *as Select* [ELEMENTOS_DIMENSAO_MENOS_A_DIMENSAO_REMOVIDA] + [ELEMNTOS_CUBO_SUM] *From* [CUBO_ENTRADA] *Where* [JUNCAO_DAS_TABELAS_DE_DIMENSAO_FATO] *Group by* [DIMENSOES_GROUPBY];
Restrict	*Create Table* [CUBO_SAIDA] *as Select* [ELEMENTOS_DIMENSAO] + [ELEMENTOS_CUBO] *From* [CUBO_ENTRADA] *Where* [ELEMENTO_DIMENSAO_FILTRO] *and* [JUNCAO_DAS_TABELAS_DE_DIMENSAO_FATO];
Merge	*Create Table* [CUBO_SAIDA] *as Select* [ELEMENTOS_DIMENSAO] + [ELEMNTOS_CUBO_SUM] *From* [CUBO_ENTRADA] *Where* [JUNCAO_DAS_TABELAS_DE_DIMENSAO_FATO] *Group by* [DIMENSOES_GROUPBY];

"Templates" para o Mapeador Objeto-Relacional_Oracle 8i	
Operação	Template
Destroy	*1. Create Type* [CUBO_SAIDA_TYPE] as *Object(* [REF_DIMENSAO_MENOS_A_DIMENSAO_REMOVIDA] + [ELEMNTOS_CUBO] *);* *2. Crete Table* [CUBO_SAIDA] *of* [CUBO_SAIDA_TYPE]; 3. *Insert into* [CUBO_SAIDA] *Select distinct* [REF_ELEMENTOS_DIMENSAO_MENOS_A_DIMENSAO_REMOVIDA] + [ELEMNTOS_CUBO_SUM] *From* [CUBO_ENTRADA] *Group by* [REF_DIMENSOES_GROUPBY]
Restrict	*1. Create Type* [CUBO_SAIDA_TYPE] as *Object(* [REF_DIMENSAO] + [ELEMNTOS_CUBO] *);*

	2. Crete Table [CUBO_SAIDA] *of* [CUBO_SAIDA_TYPE]; 3. *Insert into* [CUBO_SAIDA] *Select distinct* [REF_ELEMENTOS_DIMENSAO] + [ELEMNTOS_CUBO] *From* [CUBO_ENTRADA] *Where* [REF_ELEMENTO_DIMENSAO_FILTRO] ;
Merge (sum)	*1. Create Type* [CUBO_SAIDA_TYPE] as *Object(* [REF_DIMENSAO] + [ELEMNTOS_CUBO] *);* 2. *Crete Table* [CUBO_SAIDA] of [CUBO_SAIDA_TYPE]; 3. *Insert into* [CUBO_SAIDA] *Select distinct* [REF_ELEMENTOS_DIMENSAO] + [ELEMNTOS_CUBO_SUM] FROM [CUBO_ENTRADA] *Group by* [REF_DIMENSOES_GROUPBY];

ANEXO III

API DE ACESSO REMOTO AO S-DW-E

Segue a descrição da API com os métodos de acesso remoto a camada de aplicação do servidor. O objetivo dessa API é disponibilizar um conjunto de métodos para as aplicações da camada de interface interagirem com o S-DW-E. A API é representada pela interface IF_Servidor_RMI. Detalhamos cada método da API.

- ✓ *public ArrayList getLDW_C_IT() throws RemoteException;*
  Método que retorna uma lista de nomes do tipo *String* dos DWs disponíveis no servidor em questão.

- ✓ *public ArrayList getLCubo_C_IT(String nomeDW) throws Exception,RemoteException;*
  Método que retorna uma lista de nomes do tipo *String* dos Cubos de um DW passado como parâmetro (descrição).

- ✓ *public void processaConsulta(String nomeDW, String cubo, String texto, ConsultaCubo consultacubo) throws ExceptionCampos,Exception;*
  Método que valida e processa uma consulta no servidor. Para a execução da consulta são passados como parâmetro o nome do DW e do Cubo. Além disso, é passado como parâmetro o texto contendo as linhas com os comandos da consulta "adhoc" (baseados na BNF anexo II). O último parâmetro é passado o objeto *consultacubo* que está diretamente relacionado com a extensão "plugada" no servidor.
  Exemplo:
  S-DW-E SqlServer R
  servidor.processaConsulta(nome_dw, nome_cubo, texto_consulta, new Relacional.SqlServer.ConsultaCuboR (nome_consulta)));

S-DW-E Oracle OR

servidor.processaConsulta(nome_dw, nome_cubo, texto_consulta, new ObjetoRelacional.OracleOR.ConsultaCuboOR (nome_consulta)));

Obs.: Caso exista algum erro léxico ou sintático no texto da consulta é lançada a exceção *ExceptionCampos.*

Descrevemos a interface do classe *ExceptionCampos*

public ArrayList getItens()

Método que retorna uma referência para a lista de campos com problemas.

- ✓ *public ArrayList listaParamCubo(String nomeDW, String nomeConsulta, String Cubo) throws Exception,RemoteException;*

  Método que retorna uma lista com objetos do tipo *String* com a descrição das dimensões e dos elementos do cubo (metadados). Este método é usado para a apresentação da consulta dos dados de um cubo. São passados como parâmetro o nome do DW e da consulta. O último parâmetro é o texto como o nome do cubo e das dimensões a serem consultadas conforme o exemplo:

  Cubo =”C4(Categoria.Nome|MesAno.MesAno);”

  Obs: Caso exista algum erro léxico ou sintático no texto Cubo é lançada uma exceção do tipo *Exception*

- ✓ *public ArrayList listaLinhaCubo(String nomeDW,String nomeConsulta,String nomeCubo) throws Exception,RemoteException;*

  Método que retorna uma lista com objetos do tipo *String* que são as células dos dados da consulta de um cubo. São passados como parâmetro o nome do DW e da consulta. O último parâmetro é o texto como nome do cubo e das suas dimensões a serem consultadas conforme o exemplo:

  ”C4(Categoria.Nome|MesAno.MesAno);”

  Obs: Caso exista algum erro léxico ou sintático no texto Cubo é lançada uma exceção do tipo *Exception.*

- ✓ *public ArrayList listaLCuboConsulta(String nomeDW,String nomeConsulta) throws Exception,RemoteException;*

Método que localiza a lista com todas as descrições de “trace” gerados por uma consulta. O função deste método é permitir ao usuário visualizar os comandos (SQL ou outros) gerados para a apresentação do “trace” da consulta. Para a localização são passados como parâmetros o nome do DW e da consulta.

ANEXO IV

METADADOS S-DW-E: DTD e XML

Este anexo apresenta a definição dos arquivos DTD (estrutura) e XML (dados) que compõem o metadados do S-DW-E. Cada DW no S-DW-E é representado por um arquivos 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 para o esquema DW Vendas apresentado no Capítulo 4 seção 4.5.

1.1O 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>
```

BIBLIOGRAFIA E REFERÊNCIAS BIBLIOGRÁFICAS

[AGR 99] AGRAWAL R. & GUPTA A. & SARAWAGI S., Modeling Multidimensional Databases, IBM, 1999.

[ANS98] ANSTEY D. A., High performance Oracle8 object-oriented design, The Coriolis Group, 1998.

[BOB98] BOBROWSKI S., Oracle8 Architecture, McGraw-Hill, 1998.

[DAT98] DATE C.J. & DARWEN H., Foundation For Object/Relational Databases - The Third Manifesto, Addison-Wesle, 1998.

[FIR97] FIRESTONE J. M., Object-Oriented Data Warehousing Information Systems, Inc., White Paper, 1997.

[FOW99] FOWLER M., Refactoring: improving the Design of Existing Cod., Addisson-Wesley,1999.

[GAM94] GAMMA E. & HELM R. & JOHNSON R. & VLISSIDES J., Design Patterns Elements of Reusable Object-Oriented Software,. Addison-Wesley, 1994.

[GRA98] GRAND M., Patterns in Java: a catalog of reusable design patterns illustrated with UML, Volume I, John Wiley & Sons, 1998.

[HOR99] HORSTMANN & CORNELL, Core Java 2: Volume I - Fundamentals, Prentice-Hall, 1999.

[IBM 99] IBM, Building Object-Oriented Frameworks, White Paper, 1999.

[JAC98] JACOBSON I. & BOOCH G. & RUMBAUGH J., The Unified Software Development Processs, Addison-Wesley,1998.

[KIM96] KIMBALL, R., The Data Warehouse Toolkit, John Wiley & Sons, 1996.

[KIM98] KIMBALL, R., The Data Warehouse Lifecicle Toolkit : Expert Methods for Designing, Developing, Deploying Data Warehouses, John Wiley, 1998.

[KLE99a] 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.

[KLE99b] KLEIN L. Z., A Tecnologia Objeto-Relacional em Ambientes de Data Warehouses, Dissertação de Mestrado - IME/RJ, 1999.

[LAN95] LANDIN N & NIKLASSON A., Development of Object-Oriented Framework, Conden:Lutedx,1995.

[LAR00] LARMAN C., Utilizando UML e padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos, Bookman, 2000.

[MAR95] MARTIN J. & ODELL J., Análise e Projeto Orientados a Objeto, Makron Books,1995.

[PIR96] PIRES P. F & MATTOSO M. L. Q., Aspectos de Interoperabilidade na Arquitetura Heterogênea HIMPAR, COPPE/UFRJ, 1996.

[ROB00] ROBERTS D. & JOHNSON R, Evolving Frameworks: A Pattern Language for Developing Object-Oriented Frameworks, University of Illinois,2000.

[ROG97] ROGERS G. F., Framework-Based Software Development in C++, Prentice Hall, 1997.

[SAM00]. SAMPAIO M. C. & JORGE E. F.& BAPTISTA C. S., Metadata for an Extensible Data Warehouse Server, UFPB-COPIN/2001.

[SAM98] SAMPAIO M. C., Semana de Informática da UFBa (Seminfo'98) : Curso: Data Warehousing, UFPB-COPIN/1998.

[SAN98] SANTOS D. V., Transformação de Esquemas de Objetos para um Gerenciador Relacional Estendido, considerando o Padrão ODMG, UFPB-COPIN, Setembro/98.

[SAU00] SAUVÉ J. P., Notas de Aula da Disciplina Métodos Avançados de Programação,2000.

[STÖ99] STÖHR, T., MÜLLER, R., RAHM, E., An Integrative and Uniform Model for Metadata Management in Data Warehousing Environments. Proc. of the Int. Workshop on Design and Management of Data Warehouses (DMDW'99), pages (12-1)-(12-16), 1999.

[STO96] STONEBRAKER, M., Object/Relational DBMSs: the Next Great Wave., Morgan Kaufmann, 1996.

[TAL93] TALIGENT, Inc, Leveraging Object-Oriented Frameworks, A Taligent White Paper, 1993.

[UCH99] UCHÔA E. M. A. & MELO R. N., Integração de Sistemas de Banco de Dados Heterogêneos Usando Frameworks, PUC/Rio de Janeiro, 1999.

[ULL97] ULLMAN J. D. & WIDOM J., A First Course in Database Systems, Prentice-Hall, 1997.

[VET 00] 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.

[WEI88] WEINAND, E. GAMMA, R Marty, ET++ - An Object+Oriented Application Framework in C++, Proceedings of the OOPSLA'88, 1988 .