Pós-Graduação em Ciência da Computação

"Minerando Exceções em Cubos OLAP"

Por

Fábio Moura Pereira

Dissertação de Mestrado

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

Recife, fevereiro/2003

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Fábio Moura Pereira

Minerando Exceções em Cubos OLAP

Dissertação submetida ao Programa de Pós-graduação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. PhD Jacques Robin

Recife, fevereiro/2003

Pereira, Fábio Moura
Minerando exceções em cubos OLAP / Fábio Moura Pereira. - Recife : O Autor, 2003.
163 folhas : il., fig., tab.

Dissertação (mestrado) - Universidade Federal de Pernambuco. CIn. Ciência da Computação, 2003.

Inclui bibliografia e apêndices.

1. Banco de dados – Processamento Analítico On-line (OLAP) – Mineração de dados. 2. Inteligência artificial – Mineração de dados (OLAM). 3. Mineração de dados (Inteligência artificial) – Mineração de exceções. I. Título.

004.89 CDU (2.ed.) UFPE
006.331 CDD (21.ed.) BC2004-383

A Thiago e Tatyana

AGRADECIMENTOS

Em primeiro lugar, agradeço a minha mãe que, mesmo sem se preocupar em entender exatamente o que estava fazendo, em muito me apoiou. Agradeço ao meu pai (em memória) que me deu uma profissão a qual aprendi a amar. Agradeço a Dayse, que me apoiou e me "suportou" nos momentos mais difíceis. Agradeço a meus irmãos, Paulo e Danilo.

Também agradeço imensamente: a amiga Corina Chagas, pelo apoio incondicional; ao professor Jacques Robin – "*just do it*"; ao professor Benedito Acioly, que me mostrou "a luz no fim do túneo"; aos professores Waldenor Pereira e Francisco Tenório.

Agradeço a todos aqueles que, direta ou indiretamente, contribuíram para a conclusão deste trabalho.

RESUMO

OLAP e Mineração de Dados têm emergido separadamente como duas das técnicas de maior sucesso para suporte à decisão, independentes de domínio. Elas têm poderes e limitações complementares: OLAP fornece um serviço rápido e flexível para visualização *guiada pelo usuário* de dados multidimensionais e de multi-granularidade, enquanto a mineração de dados fornece um serviço *automático* para descoberta de padrões e dados. Assim como OLAP pode representar um passo de pré-processamento e seleção de dados eficiente, flexível e iterativo para a mineração de dados, por sua vez, a mineração de dados pode representar um guia automático que acelera em muito a procura por *insigths* nos dados OLAP.

Esta dissertação investiga a integração sinérgica entre essas duas tecnologias, que recentemente passou a ser chamada de OLAM (*On-Line Analytical Mining*), com a finalidade de minerar desvios em um data warehouse multidimensional e de multi-granularidade. Construído a partir de um trabalho teórico previamente realizado neste tema, a dissertação lida com aspectos práticos da integração de alguns algoritmos propostos para mineração de exceções em dados OLAP dentro de JODI/OCCOM: um sistema cliente-servidor, de código aberto, independente de plataforma, multi-usuário e com uma interface gráfica amigável. O sistema é o primeiro do gênero e pode ser prontamente incorporado a ambientes de descoberta de conhecimento baseados em Java ou orientação a objetos. A combinação de JODI/OCCOM com o previamente desenvolvido gerador de hipertexto em linguagem natural HYSSOP, permite a geração automática de resumos, no formato de páginas web, de células de dados que são estatisticamente consideradas exceções em um contexto multidimensional e multi-granular de um data warehouse OLAP.

Palavras-chave: OLAP, mineração de dados, OLAM, mineração de exceções.

ABSTRACT

OLAP and Data Mining, have independently emerged as two of the most successful domain-independent decision support techniques. They have complementary strengths and limitations: OLAP provides fast and flexible, *user-guided* multidimensional and multi-granular data aggregation and visualization services, while data mining provides *automated* data and pattern discovery services. Whereas OLAP can provide an efficient, flexible, iterative, data selection pre-processing step for data mining, data mining can in turn provide automated guidance that greatly speeds-up insight searches of OLAP data.

This dissertation investigates the synergetic integration of both, recently coined OLAM (On-Line Analytical Mining), for the task of mining outliers in a multidimensional and multi-granular data warehouse. Building on previous theoretical work on this topic, the dissertation tackles the practical aspects of integrating several proposed OLAP data outlier mining algorithms inside JODI/OCCOM, an open-source, platform-independent, multi-user, client-server system with a user-friendly graphical interface. The system is the first of its kind and can be readily incorporated in Java-based or multi-language, object-oriented, knowledge discovery frameworks. The combination of JODI/OCCOM together with the previously developed automatic hypertext natural language generator HYSSOP allows the automatic summarizing (as simple, two-web page, textual brief) of data cells that are statistical outliers in the multidimensional and multi-granular context of an OLAP data warehouse.

Key-words: OLAP, data mining, OLAM, outlier mining.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O processamento analítico on-line (OLAP, do inglês *On-Line Analytical Processing*) (CODD, et.al., 1993) e a mineração de dados têm emergido, de forma independente, como duas das tecnologias mais populares e de maior sucesso de apoio ao processo de tomada de decisão dentro das organizações. Esta dissertação surgiu do intuito de integração destas duas tecnologias. Apresentaremos neste capítulo, de forma resumida, o contexto em que esta dissertação está inserida, a sua motivação e os seus objetivos.

1.1 Data Warehousing e OLAP

Nos dias atuais, analistas de negócios estão submetidos a uma quantidade muito grande de informações, tornando cada vez mais complexa a tarefa de gerenciamento das organizações. Uma arquitetura de banco de dados que emergiu recentemente é a de *data warehouse*, "um repositório de múltiplas e heterogêneas fontes de dados, organizados dentro de um esquema unificado em um mesmo local, de maneira a facilitar o processo de tomada de decisão" (HAN; KAMBER, 2001). A construção de *data warehouses* é uma tecnologia que tem por objetivo converter uma grande quantidade de dados em informação que possa ser utilizada de maneira estratégica.

Diversos motivos levaram à popularização desta tecnologia:

- Crescimento exponencial dos dados dentro das organizações;
- Em vez de dados acumulados, usuários querem informação;
- Usuários dominam negócios e não computadores;
- Decisões precisam ser tomadas rapidamente e de maneira correta, utilizando toda informação disponível;
- A competição está aquecendo áreas de inteligência de negócio e dando cada vez mais valor à informação;
- A adoção da tecnologia de *data warehouse* melhora a produtividade da empresa e a qualidade de seus serviços;
- Eficiência não é mais a chave para o sucesso: a flexibilidade tomou esse lugar.

A construção de um *data warehouse* passa por diversas etapas:

1. Limpeza dos dados – remoção de dados inconsistentes e ruídos;

2. Integração dos dados – combinação de dados de múltiplas fontes;
3. Seleção dos dados – os dados relevantes para o processo de análise são recuperados do banco de dados;
4. Transformação dos dados – os dados são transformados ou consolidados dentro de um formato apropriado para o processamento analítico.

Após o processo de construção, os dados dentro de um *data warehouse* se encontram disponíveis para uso. Com a finalidade de facilitar a realização de consultas analíticas a dados armazenados em um *data warehouse* surgiu a tecnologia de processamento analítico on-line (OLAP), técnicas de análise que incluem funcionalidades para agregação dos dados, permitindo visões por diferentes ângulos.

O processamento analítico on-line possibilita a realização de consultas, retornando agregações de dados quantitativos ao longo de várias dimensões analíticas categóricas, com vários níveis de granularidade. Por exemplo, é possível analisar o volume de vendas de uma empresa em função da região, família de produto e ano; caso o analista quisesse dados mais detalhados, poderia refinar esta consulta para cidade, nome do produto e mês das vendas.

OLAP requer o uso de bancos de dados analíticos especializados, que são derivados a partir de bancos de dados transacionais, mas que apresentam novas características:

- Novos modelos de dados (multidimensional, multi-granular) – fornecidos pelos *data warehouses*;
- Novas linguagens de consulta – que permitam acesso a dados armazenados em um formato multidimensional;
- Servidores especializados – que forneçam um modelo para construção de cubos OLAP multidimensionais, devendo incluir um formato de armazenamento multidimensional especializado para os dados dos cubos.

Os dados em um servidor OLAP são sumarizados e armazenados em um cubo de dados multidimensional. Um cubo OLAP (ou cubo de dados) consiste em um conjunto de medidas numéricas para análise e um conjunto de dimensões que provêm o contexto das medidas. Uma medida é definida por uma função de agregação numérica que pode ser avaliada para cada célula do cubo (soma, quantidade, média). Uma dimensão é descrita por um conjunto de atributos e pode possuir uma hierarquia conceitual, como em Produto (família do produto, departamento e nome) ou em tempo (ano, mês, dia).

Esta organização fornece ao usuário flexibilidade para visualizar os dados a partir de diferentes perspectivas. Operações em cubos de dados OLAP existem a fim de materializar estas diferentes visões, permitindo busca e análise interativa do material em mãos. Portanto,

OLAP provê um ambiente amigável para análise interativa dos dados. As principais operações OLAP existentes são:

- *Drill-down* – navega de um nível menos detalhado para um nível mais detalhado dos dados;
- *Roll-Up* – realiza uma agregação no cubo de dados, tanto subindo um nível na hierarquia de uma dimensão como pela diminuição no número de dimensões. É o inverso da operação de *drill-down*;
- *Slice* e *dice* – realiza uma seleção de parte(s) de um cubo para visualização;
- *Pivot* – realiza uma rotação no cubo (dados apresentados em linhas passam a ser mostrados em colunas e vice-versa).

As operações mostradas acima permitem o que conhecemos por descoberta *dirigida pelo analista*, onde o usuário navega pelo cubo de dados a procura de tendências gerais ou dados atípicos, gerando *insights* sobre a atividade modelada no banco de dados. Porém, este método de análise dos dados possui várias limitações, dentre elas o fato de que a explosão combinatória das consultas OLAP possíveis, usando linguagens expressivas para analisar um domínio com muitas dimensões e muitos níveis de granularidade, torna praticamente impossível o processo de descoberta de informações interessantes por parte do usuário. Daí a necessidade de funcionalidade de um guia automático que possa sugerir consultas promissoras para o analista.

1.2 Mineração de Dados e Descoberta de Conhecimento em Bancos de Dados

O termo Descoberta de Conhecimento em Bancos de Dados (*KDD – Knowledge Discovery in Databases*) refere-se ao processo abrangente de se abstrair conhecimento a partir dos dados de uma grande base. KDD é um processo de identificação de padrões previamente desconhecidos, que consiste na aplicação de algoritmos de aprendizagem de máquina ou estatística, que produz um conjunto de padrões, não triviais, interpretáveis e válidos, e potencialmente úteis, em meio aos dados (FSS, 1996). O processo de KDD é dividido em várias etapas:

- Definição do objetivo: qual tipo de conhecimento se deseja extrair e como representá-lo;
- Seleção dos dados: conjunto de dados alvo onde será realizado o processo de descoberta;

- Pré-processamento e limpeza dos dados: preparação dos dados para processamento, remoção de ruídos e re-configuração dos dados;
- Transformação: os dados são colocados em um formato que possa ser utilizado pelos algoritmos de aprendizagem;
- Mineração de dados: escolha e aplicação de um método particular (algoritmo de aprendizagem de máquina) para extração dos padrões de comportamento dos dados;
- Interpretação e avaliação: análise dos padrões identificados a fim de descobrir se os mesmos representam informação nova ou relevante, também envolve a apresentação dos padrões e modelos extraídos.

Como vimos, a Mineração de Dados é uma etapa do processo de KDD, e consiste no uso de algoritmos estatísticos ou de aprendizagem de máquina escaláveis com a finalidade de descoberta *automática* de tendências gerais ou de dados atípicos, fornecendo *insights* sobre a atividade modelada no banco de dados que possam auxiliar no processo de tomada de decisão. Uma limitação dos algoritmos de mineração de dados é o fato das primeiras iterações serem ingênuas, geralmente trazendo tendências óbvias ou incompreensíveis. A descoberta de *insights* para a tomada de decisão requer um processo exploratório em espiral, com várias iterações através das etapas de limpeza, re-formatação, seleção e mineração, antes do surgimento de informação relevante. Se faz necessário um ambiente integrado que permita a realização de iterações rápidas e acelere o processo de descoberta de conhecimento.

Assim como OLAP, o processo de KDD requer o uso de bancos de dados analíticos especializados, derivados a partir de bancos de dados transacionais, mas com dados previamente limpos, re-formatados e selecionados, para que possam ser processados com êxito pelos algoritmos de mineração. Em outras palavras, após a construção do *data warehouse* (que envolve as fases de seleção e pré-processamento), existem duas maneiras de extrair *insights* decisórios dos dados: OLAP e mineração de dados.

1.3 Foco da Dissertação

A dissertação tem como foco principal a integração sinérgica entre as tecnologias de OLAP e Mineração de Dados. No meio de diferentes paradigmas e arquiteturas de sistemas de mineração de dados, OLAM (do inglês *On-Line Analytical Mining*) (HAN et.al., 1997) que tem por finalidade a mineração de conhecimento em bancos de dados multidimensionais, é

particularmente importante pela seguinte razão: complementaridade de OLAP e mineração de dados. Enquanto OLAP utiliza dados de alta qualidade (limpos, integrados, consistentes) necessários para a aplicação de ferramentas de análise, a mineração de dados, por sua vez, automatiza o processo de descoberta de padrões interessantes, análise e exploração do grande volume de dados disponíveis nos sistemas OLAP.

Apresentamos, como resultado do trabalho desenvolvido, um componente Java para mineração de exceções em cubos OLAP: OCCOM (*OLAP Cuboid Cell Outlier Miner*) guia usuários a explorarem cubos OLAP de maneira eficiente, possuindo as seguintes características:

- O seu objetivo é a mineração de valores extremos (*outlier mining*) em cubos de dados OLAP;
- Minera dados atípicos em cubóides OLAP por meio de análise estatística multidimensional e multi-granular;
- Apresenta aspectos de uma ferramenta prática, *open-source*, portável e baseada em tecnologia de baixo custo;
- Reaproveita algoritmos propostos em pesquisas teóricas (SARAWAGI, et.al., 1998; CHEN, 1999), que constituem uma base sólida para o seu desenvolvimento;
- Integra os algoritmos de mineração em uma ferramenta Java, com interface gráfica amigável;
- O seu protótipo funciona através do uso de consultas OLAP em MDX (*Multidimensional Expressions*) utilizando o Microsoft SQL Server;
- Suas camadas de interface e mineração são reutilizáveis para uso com OLAPI e ORACLE9i;
- Preenche a principal lacuna na arquitetura de descoberta de conhecimento em bancos de dados MATRIKS – projeto do Centro de Informática da Universidade Federal de Pernambuco (CIn/UFPE) (FAVERO, 2000);

O desenvolvimento de OCCOM foi motivado pelo projeto MATRIKS, que propõe a construção de um ambiente abrangente e aberto para descoberta de conhecimento em bancos de dados.

O projeto MATRIKS, que tem como alvo a integração de tecnologias de *data warehousing*, bancos de dados multidimensionais, mineração de dados e geração automática de hipertextos em linguagem natural, se baseia no principal gargalo dos ambientes atuais de KDD: a falta de integração dos vários serviços computacionais necessários no processo exploratório, iterativo e interativo de KDD.

1.4 Estrutura da Dissertação

A dissertação está organizada da seguinte maneira:

No Capítulo 2 apresentamos o contexto em que a pesquisa está inserida, o processo de descoberta de conhecimento em bancos de dados, o conceito de *data warehouse*, OLAP, OLAM e o ambiente MATRIKS.

No Capítulo 3 são relatados os trabalhos relacionados ao tema estudado.

No Capítulo 4 está documentado o desenvolvimento da API JODI, que implementa o modelo ODCI.

No Capítulo 5 relatamos o desenvolvimento do componente de mineração de dados OCCOM, sua arquitetura, modelo, implementação e testes.

Os aspectos de desenvolvimento da interface gráfica de OCCOM são apresentados no Capítulo 6.

E finalmente, no Capítulo 7, realizamos uma conclusão do trabalho, destacando suas contribuições e trabalhos futuros.

2 CONTEXTO DA PESQUISA

Organizações modernas para se manterem competitivas, necessitam responder rapidamente às mudanças do mercado. Para tal, elas necessitam de rápido acesso às informações relevantes antes de tomarem qualquer decisão de negócio. Por outro lado, o crescimento explosivo de bases de dados empresariais, governamentais e científicas têm ultrapassado o poder das tecnologias convencionais de banco de dados para interpretar e digerir estes dados, criando a necessidade de uma nova geração de ferramentas e técnicas para automatizar e dar mais inteligência aos processos analíticos em bases de dados (FSS, 1996). Neste contexto, o campo de Sistemas de Suporte à Decisão (*Decision Support Systems - DSS*) tem crescido rapidamente. Sistemas de Suporte à Decisão possuem a finalidade de prover automaticamente, para os responsáveis pelo processo de tomada de decisão dentro das organizações, *insights* descobertos e informações agregadas de valor, necessárias para se fazer as melhores escolhas e de forma mais rápida (CAMPOS; FILHO, 2000).

Neste cenário de apoio à tomada de decisão, onde há a necessidade de automação do processo de obtenção, organização e tratamento dos dados para análise, surgem os Sistemas de Suporte à Decisão, envolvendo as seguintes tecnologias de Banco de Dados e Inteligência Artificial: *Data Warehouse*, OLAP e Mineração de Dados.

Nas próximas seções são mostradas estas tecnologias, além do projeto MATRIKS, onde este trabalho está inserido.

2.1 Data Warehouse

Sistemas de Suporte à Decisão (DSS) dão lugar a diferentes requisitos de tecnologias de banco de dados daqueles exigidos em aplicações de processamento transacional tradicional. Em DSS, o ambiente de dados é fundamentalmente diferente do ambiente convencional de processamento de transações. Em um ambiente convencional, os dados são organizados de forma a otimizar as operações transacionais, já em um ambiente de DSS, os dados são organizados para fins de exploração e análise.

Desta forma, as aplicações típicas de uma empresa podem ser classificadas em: aplicações do negócio, que garantem a operação da empresa (sistemas de produção); e aplicações sobre o

negócio, que analisam o negócio (sistemas de suporte à decisão e sistemas de informações executivas) (CAMPOS; FILHO, 2000).

Para dar suporte a esses dois tipos de aplicações, uma arquitetura de dados adequada deve possuir: bancos de dados operacionais, para dar suporte às aplicações do negócio; e os bancos de dados de apoio à decisão, para dar suporte às aplicações sobre o negócio. Desta forma, normalmente o *data warehouse* é mantido separadamente da base de dados operacional da organização, existindo várias razões para tal. O *data warehouse* serve como base para o processamento analítico *on-line* (OLAP) e a mineração de dados, onde os requisitos funcionais e de performance são bem diferentes das aplicações de processamento transacional *on-line* (OLTP), cujos bancos de dados tradicionais dão suporte.

Aplicações de OLTP normalmente automatizam as operações do dia a dia das organizações, tais como entrada de pedidos e transações bancárias. Estas transações tem as seguintes características:

- São estruturadas e repetitivas, e consistem em transações pequenas, isoladas e atômicas;
- Requerem atualização de dados constante, e relativamente fazem poucas leituras aos mesmos. As atualizações são feitas de acordo com a mudança de estado do objeto a que ele se refere, não sendo guardado histórico destas atualizações.
- Consistência e recuperação da base de dados são questões críticas;
- A maximização do desempenho transacional é a principal métrica de desempenho;
- Os dados são organizados orientados por atividades/operações funcionais. A base de dados é projetada para refletir a semântica operacional de aplicações conhecidas, e minimizar conflitos de concorrência.

Os *data warehouses*, em contraste, visam suporte à decisão, e possuem as seguintes características:

- Dados históricos, resumidos e consolidados são mais importantes do que registros individuais detalhados. Sistemas de Suporte à Decisão requerem dados que podem ser perdidos no ambiente operacional, por exemplo, a análise de tendências e previsões requerem dados históricos, entretanto, bases de dados operacionais armazenam apenas os dados correntes;
- Dados integrados de diversas fontes operacionais: *data warehouses* podem conter dados consolidados de diversas bases de dados operacionais heterogêneas. As diferentes fontes podem conter dados de qualidade variável ou usar representações, códigos, ou formatos inconsistentes. Desta forma, é importante integrar os dados (nomes, unidades de medidas),

de forma que sejam transformados até um estado uniforme. Por exemplo, o dado sexo: uma aplicação pode codificá-lo como M/F, outra como H/M ou ainda 0/1. No *data warehouse*, estes dados são convertidos para um estado uniforme, codificando-o de uma única forma;

- Possuem grande volume de dados: visto que *data warehouses* podem conter dados consolidados de diversas bases de dados operacionais, e potencialmente sobre longos períodos de tempo, eles tendem a ser ordens de magnitude maiores do que as bases de dados operacionais;
- Orientados por assunto: caracteriza o fato do *data warehouse* armazenar informações a respeito de assuntos importantes para o negócio da empresa para fins de análise. Por exemplo, produtos, lojas, clientes. Enquanto que, em ambientes operacionais os dados estão organizados voltados para atividades, isto é, estão modelados e armazenados voltados para as operações transacionais do dia a dia da empresa, onde normalmente são acessados por suas chaves primárias;
- Otimizados fisicamente para consultas analíticas (OLAP): a carga de trabalho em *data warehouses* é de consultas intensivas, complexas e *ad-hoc*, que podem acessar milhões de registros, e executar várias junções e agregações. Um *data warehouse* contém dados extraídos do ambiente de produção da empresa, selecionados, depurados, e principalmente otimizados para o processamento de consultas, e não para o processamento de transações, como são os bancos de dados destinados aos sistemas de produção;
- Modelados logicamente para consultas analíticas (OLAP): o desempenho de consultas e tempos de resposta das mesmas são mais importantes do que o desempenho de transações. Para tal, os dados de um *data warehouse* são modelados especialmente voltados para análise;
- Não voláteis: os dados armazenados em um *data warehouse* são apenas para leitura. Significando que, em um *data warehouse*, após os dados serem integrados e transformados, é dada a carga inicial dos dados e posteriormente apenas consultas podem ser realizadas sobre esses dados;
- Limpeza de dados: considerando que *data warehouses* são usados para tomada de decisão, é importante que seus dados estejam corretos. Em grandes volumes de dados provenientes de diversas fontes, existe uma grande probabilidade de existirem erros e anomalias nos dados. Dessa forma, antes de serem consolidados em um *data warehouse*, os dados são limpos, para que não sejam armazenados ruídos (dados inconsistentes, em branco, ou

anômalos; tamanhos de campos inconsistentes, descrições inconsistentes). Tais ruídos nos dados podem deteriorar a qualidade dos resultados dos algoritmos de *data mining*, que poderão ser posteriormente aplicados sobre os dados;

- Fornecem uma visão simples e concisa em torno de alguns assuntos: dados que não são úteis ao processo de tomada de decisão devem ser excluídos;
- Possuem um horizonte de tempo significantemente maior que em sistemas operacionais: fornece informações a partir de uma perspectiva histórica (5 a 10 anos).

Visto essas características, temos então a seguinte definição de *Data Warehouse* segundo Inmon (1996): um *Data Warehouse* é uma coleção de dados orientada por assuntos, integrada, variante no tempo e não volátil, que tem por objetivo dar suporte ao processo de tomada de decisão.

As diversas etapas do processo para construção de um *data warehouse* pode ser melhor visualizado através da Figura 2.1.

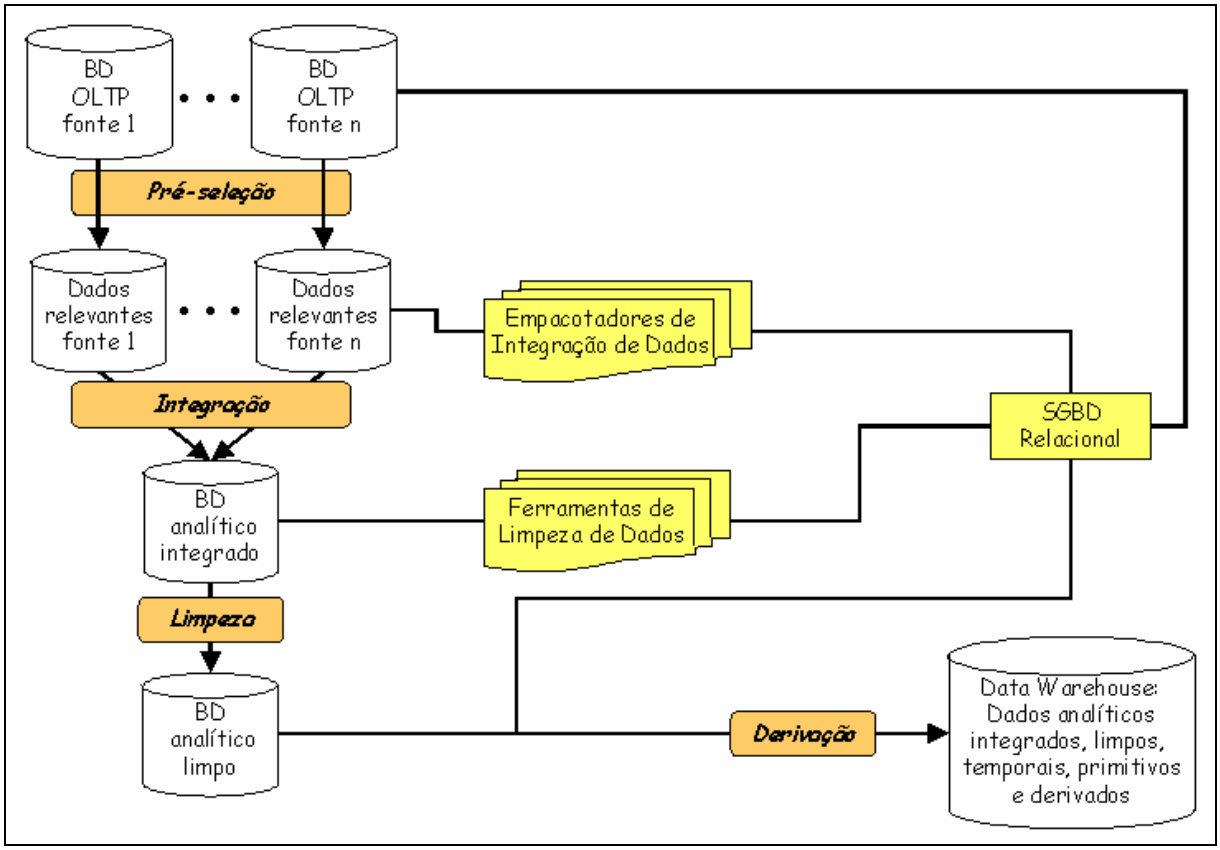


Figura 2.1 – Processo de construção de um *Data Warehouse*.

Dados de diversas fontes OLTP são pré-selecionados e integrados através de empacotadores de integração de dados, formando um banco de dados analítico integrado. Posteriormente ferramentas de limpeza de dados são utilizadas a fim de gerar um banco de dados analítico limpo, que através de um processo de derivação pode gerar novas informações, formando

assim o *data warehouse*, um conjunto de dados analíticos integrados, limpos, temporais, primitivos e derivados, armazenados em um banco de dados relacional.

Considerando que bases de dados operacionais são otimizadas para suportar a carga de trabalho de OLTP, tentar executar consultas complexas de OLAP nestas bases de dados pode resultar em desempenhos não aceitáveis (maiores detalhes de OLAP serão abordados na próxima seção). Por essas razões é que os *data warehouses* são, geralmente, implementados separadamente das bases de dados operacionais.

Para facilitar as consultas, análises complexas e visualização, os dados em um *data warehouse* são organizados em torno de pontos principais. Por exemplo, em um *data warehouse* de vendas, data da compra, cliente, vendedor e produto podem ser algumas das dimensões representadas. Geralmente estas dimensões possuem hierarquias: a data da compra, por exemplo, pode ser organizada em uma hierarquia de dia-mês-semestre-ano, produto em uma hierarquia de produto-categoria-indústria, e assim por diante. Operações típicas de OLAP, como veremos na próxima seção, são realizadas sobre dimensões e hierarquias.

2.2 OLAP

OLAP (CODD, et.al., 1993; CHAUDHURI; DAYAL, 1996) é uma categoria de software específica para realizar processamento analítico dos dados de um *data warehouse*, de forma que este processamento deve: (1) ocorrer com alto desempenho e interatividade, e (2) auxiliar à tomada de decisão em uma organização, por meio da interpretação desses dados em uma variedade de visões multidimensionais. A Figura 2.2 mostra uma arquitetura OLAP típica.

Visões multidimensionais dos dados em um *data warehouse E-R* são um modelo conceitual de dados que influencia as ferramentas de visualização, o projeto do *data warehouse*, e os mecanismos de consulta. Em um modelo de dados multidimensional existe um conjunto de *medidas* numéricas que são os objetos da análise. Exemplos de medidas são: valor das vendas, quantidade vendida, lucros, inventário. Cada medida depende de um conjunto de *dimensões* que fornecem o contexto da medida. Por exemplo, as dimensões associadas com a quantidade de vendas podem ser cidade, produto, e data da venda. As dimensões juntas determinam unicamente a medida. Desta forma, uma medida é um valor no espaço multidimensional de dimensões. Cada dimensão é descrita por um conjunto de atributos. Por exemplo, a dimensão produto pode ser descrita pelos seguintes atributos: categoria da indústria, a indústria que

fabrica o produto, ano em que foi introduzido no mercado e média de lucro. Os atributos de uma dimensão podem estar relacionados através de uma hierarquia. Ainda no exemplo de vendas, a dimensão produto pode conter a seguinte hierarquia entre seus atributos: indústria → categoria → produto.

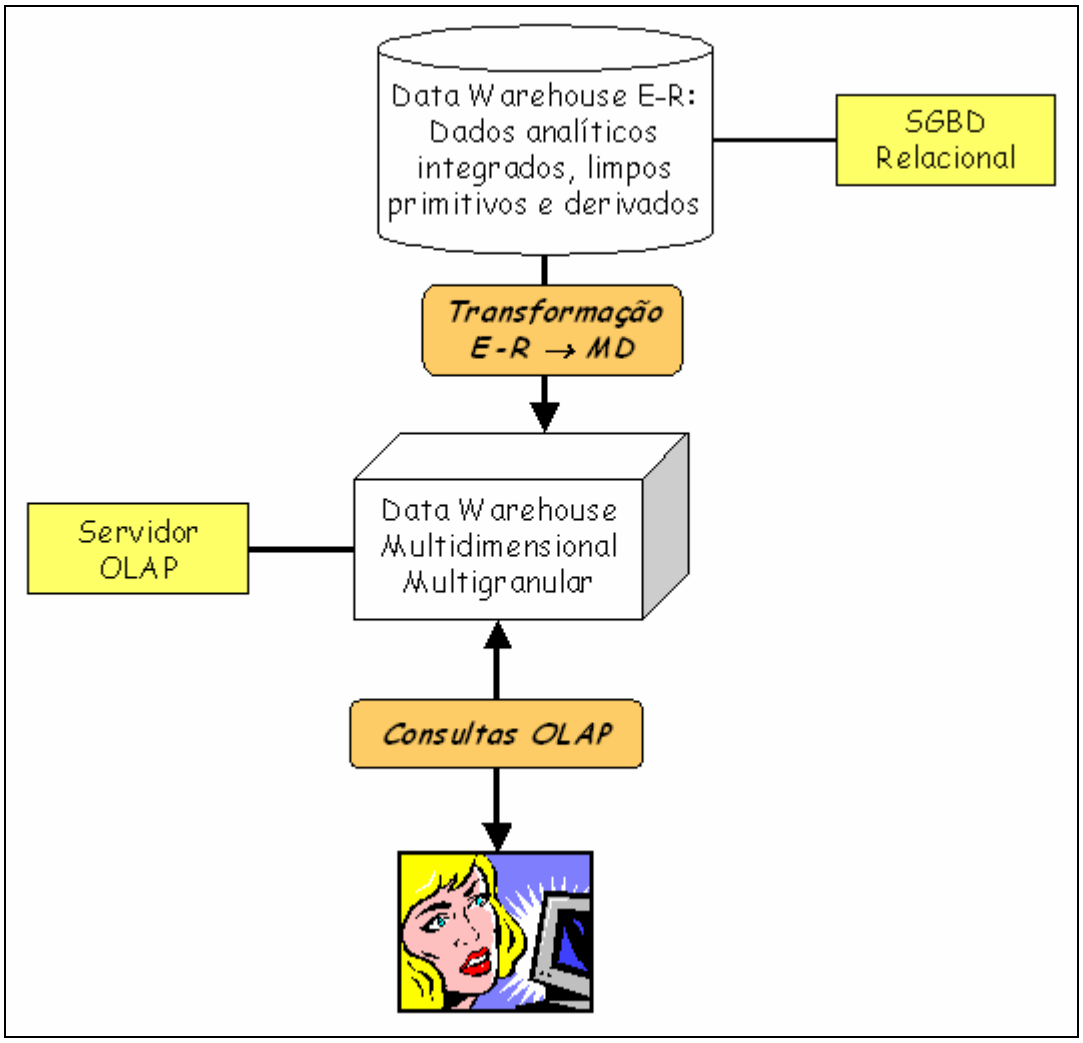


Figura 2.2 – Arquitetura OLAP típica.

Entre os tipos de processamento executados em uma base de dados de uma organização, vimos, na seção anterior, que o processamento analítico (OLAP) defini-se em contraste ao processamento transacional (OLTP), que é destinado a armazenar e analisar dados de suporte operacional, ou seja, dados essencialmente manipulados para permitir análise de registros atômicos com eficiência. Em OLAP, os dados estão otimizados para o processamento analítico, enquanto que em OLTP, os dados estão otimizados para o processamento transacional. Na Figura 2.3, vemos um resumo das diferenças entre aplicações de OLAP e OLTP.

Visões multidimensionais sobre os dados em OLAP provêm do fato de que questões típicas de análise de negócios organizacionais geralmente requerem a visualização dos dados

segundo diferentes perspectivas. Por exemplo, suponha-se que uma grande rede de lojas departamentais necessita analisar seu negócio para ter a capacidade de planejar e reagir, rapidamente às mudanças nas condições de seus negócios e, desta forma, tomar decisões para se posicionar de forma competitiva no mercado. Para tal, é necessário analisar o histórico dos dados da empresa sobre as vendas, sobre o estoque de produtos. Uma análise deste tipo requer uma visão histórica por meio de diversas perspectivas, como por exemplo: totais de vendas por produto, totais de vendas por região, e totais de vendas por período de tempo. A análise de dados sob diversas perspectivas permite fazer previsões sobre o negócio em questão, como por exemplo, no caso da rede de lojas departamentais, responder perguntas do tipo: *"Qual a tendência do volume de vendas, de um determinado produto, para um determinado período de tempo, em uma determinada região de atuação da rede de lojas departamentais?"*.

	OLTP	**OLAP**
Objetivos	Análise diária dos dados do negócio	Analise histórica sobre os dados do negócio
Visão dos dados	Relacional	Multidimensional
Operações com os dados	Inclusão, Alteração, Exclusão e Consulta.	Carga e Consulta
Atualização	Contínua (tempo real)	Periódica (em lote)
Número de usuários	Centenas	Dezenas
Tipo de usuário	Operacional	Gerencial
Interação com o usuário	Predominantemente pré-definida	Predominantemente ad-hoc
Granularidade dos dados	Detalhada	Detalhada e resumida
Redundância	Não existe	Existe
Volume de armazenamento	Megabytes – Gigabytes	Gigabytes – Terabytes
Histórico dos dados	Não mantém	Mantém
Acesso aos registros na ordem de	Dezenas	Milhares

Figura 2.3 – OLAP x OLTP (LINO, 2000)

Desta forma, é a análise de dados sob diferentes perspectivas, que dá origem ao aspecto multidimensional da tecnologia OLAP. Discutiremos agora, alguns conceitos pertinentes a tecnologia OLAP, relacionados a este aspecto multidimensional, por meio do exemplo da Figura 2.4.

Conceitos:

- Dimensões: São as diferentes perspectivas envolvidas. Dimensões geralmente correspondem a campos não numéricos em um *data warehouse* e fornecem informações descritivas. No caso da Figura 2.4, as dimensões são: Tempo (*Time*), Localização (*Location*), e Item (*Item*).

- Medidas: Disponibilizam as informações quantitativas que se deseja consultar e analisar, isto é, campos numéricos em um banco de dados. No caso da Figura 2.4, a medida é o total de vendas de produtos.
- Cubos ou hipercubos: Os dados que são extraídos do *data warehouse E-R* são organizados e armazenados em estruturas multidimensionais chamadas de cubo[1]. No caso da Figura 2.4, temos um cubo com três dimensões: Tempo (*Time*), Localização (*Location*), e Item (*Item*).

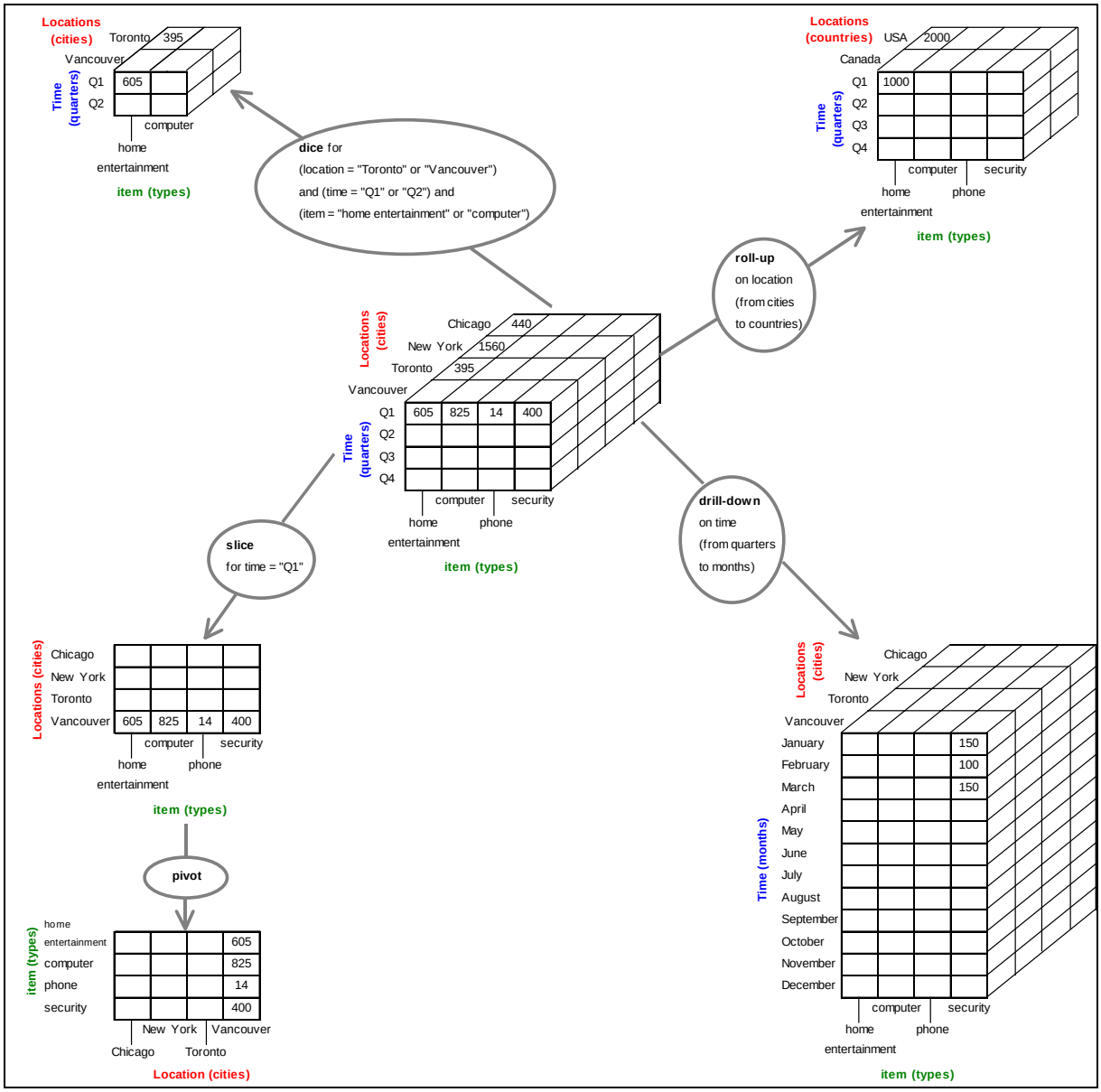


Figura 2.4 - Cubo de dados de vendas por trimestre, local e tipo de item (HAN; KAMBER, 2001)

[1] Um cubo pode conter n dimensões, e é chamado de n-dimensional.

- Funções de Agregação: Avaliam-se agregações das medidas segundo as dimensões do cubo por meio de funções de agregação. Alguns exemplos de funções de agregação são: soma, média, desvio padrão, mínimo, máximo. No caso da Figura 2.4, a função de agregação utilizada é soma (*sum*), onde são apresentadas as somas dos totais das vendas segundo as dimensões, isto é, a soma de vendas de computadores (*computer*), no primeiro trimestre (*Q1*), na cidade de Toronto, por exemplo.
- Membros: São os elementos de uma dimensão. No caso da Figura 2.4, os membros da dimensão Item são *home entertainment*, *computer*, *phone* e *security*, enquanto que os membros da dimensão Localização (*Location*) são Vancouver, Toronto, New York e Chicago.
- Hierarquias: Os membros de uma dimensão são organizados (agregados) em níveis de hierarquias. Por exemplo, a dimensão Tempo (*Time*) da Figura 2.4 poderia organizar seus membros em níveis de granularidade, como Ano, Trimestre, Mês, Dia; ou seja, em uma hierarquia temporal. Em uma hierarquia, a granularidade de um membro de nível inferior é sempre menor do que a de um nível superior. Por exemplo, a granularidade de Dia é menor que a de Ano. A disposição dos membros de uma dimensão, nos níveis de uma hierarquia desta dimensão, deve ser feita respeitando-se o grau de agregação dos mesmos, pois um membro de menor granularidade sempre deve estar imediatamente abaixo do seu membro de granularidade maior seguindo-se a hierarquia.

OLAP também caracteriza-se por possuir uma série de operações específicas para a manipulação/navegação dos dados multidimensionais. A seguir são discutidas brevemente algumas das operações mais populares, ilustradas com exemplos do cubo de vendas da Figura 2.4:

- *Drill-down*: desagregação/detalhamento dos dados para um nível com menor granularidade. Supondo-se que exista a hierarquia *Tempo*, definida como Ano→ Trimestre→Mês→Dia no exemplo da Figura 2.4, então, aplicando-se a operação *drill-down* sobre a dimensão *Tempo*, obter-se-ia uma nova visualização dos dados, com a dimensão data disposta segundo o mês do ano da compra, conforme pode ser observado. Neste caso, mês (*months*) é um nível imediatamente abaixo de trimestre (*quarters*), na hierarquia definida de tempo;
- *Roll-up* ou *Drill-up*: agregação dos dados para um nível com maior granularidade. A operação de *roll-up* é exatamente o inverso da operação de *drill-down*, isto é, aplicando-se operações de *roll-up*, partindo-se de visualizações de dados em níveis de granularidade inferiores, chega-se a níveis de granularidade superiores. Na Figura 2.4, realizando-se a

operação de *roll-up* na dimensão Localização (*Location*), que se encontra no nível de hierarquia *Cidades* (*cities*), esta passaria para o nível imediatamente acima, que neste caso é representado por *Países* (*countries*);

- *Slice/Dice*: seleção de parte de um cubo (fatiamento do cubo). Por exemplo, na Figura 2.4, uma operação de *slice* pode ser aplicada ao cubo, de modo que sejam selecionadas as células do cubo que fazem parte da dimensão Localização e Item para o primeiro trimestre (*Q1*) (fatia do cubo). A operação *dice* define um sub-cubo pela seleção de duas ou mais dimensões;
- *Pivoting/Rotate*: inversão/rotação dos eixos do cubo para visualização dos resultados de uma consulta. Na Figura 2.4, o cubo está sendo visualizado, dispondo-se a dimensão Item no eixo-x e a dimensão Localização no eixo-y. Por meio da operação de *pivoting* pode-se mudar a perspectiva de visualização, invertendo-se os eixos, isto é, dispondo-se a dimensão Localização no eixo-x e a dimensão Item no eixo-y, por exemplo;
- *Drill-across* - desagregação/detalhamento dos dados através de múltiplos níveis de dimensões diferentes. Por exemplo, na Figura 2.4, uma operação de *drill-across* aplicada ao cubo *Países* (*countries*) x *Trimestres* (*quarters*) x *Tipos* (*types*), poderia levar diretamente ao cubo *Cidades* (*cities*) x *Meses* (*months*) x *Tipos* sem passar pelo cubo intermediário *Cidades* x *Trimestres* x *Tipos*.

Apesar de aplicações de *OLAP* apresentarem os dados em visões lógicas multidimensionais, estes não necessariamente estão armazenados fisicamente em estruturas multidimensionais[2]. Estruturas relacionais podem ser usadas para a representação e armazenamento de dados multidimensionais. De acordo com a forma como os dados estão armazenados no *data warehouse* que dá suporte aos sistemas OLAP, três tipos de arquitetura caracterizam as ferramentas OLAP (CHAUDHURI; DAYAL ,1996; PENDSE , 2000):

- ROLAP (*Relational* OLAP): Realiza seu processamento analítico em um *data warehouse* com estrutura física relacional, e modelado dimensionalmente por meio de técnicas de modelagem chamadas Esquema Estrela, Esquema Floco de Neve[3] (KINBALL, 1996; INMON, 1996; CAMPOS; FILHO, 2000) , ou Esquema Constelação[4];

[2] SGBD Multidimensionais.

[3] Em Esquemas Floco de Neve as tabelas de dimensões estão normalizadas.

[4] Em Esquemas Constelação são usadas estruturas mais complexas, nas quais múltiplas tabelas de fatos compartilham tabelas de dimensões.

- MOLAP: (*Multidimensional* OLAP): Realiza seu processamento analítico em um *data warehouse* cujo armazenamento físico usa tecnologia de banco de dados multidimensionais (GYSSENS; LAKSHMANAN, 1997) com matrizes n-dimensionais;
- HOLAP (*Hybrid* OLAP)- Integra as características funcionais da ROLAP e MOLAP em uma única arquitetura híbrida. O armazenamento físico dos dados do *data warehouse* é feito em tabelas relacionais, entretanto para implementar eficientemente as consultas, um cache dos níveis de agregação mais comuns é guardado na memória como uma matriz n-dimensional.

O ambiente OLAP permite aos usuários facilmente sumarizar e acessar dados, mas possuem algumas limitações na construção e manutenção de modelos analíticos complexos dos dados organizacionais. Para aumentar a sua eficácia, sistemas OLAP devem dar suporte a linguagens de consulta, possíveis em diferentes níveis de abstração (TORLONE; CABIBBO, 1998).

Consultas OLAP não podem ser previstas e são muito dinâmicas. Os tipos de informações requisitadas cobrem todo o escopo dos dados disponíveis. Consultas devem possuir a habilidade de tirar vantagem dos relacionamentos representados no banco de dados.

Uma das estratégias para responder a consultas OLAP rapidamente é a de computação prévia de consultas complexas envolvendo múltiplas agregações dependentes em múltiplas granularidades. Estes cubos pré-computados são muito úteis na prática, já que muitas consultas complexas podem ser respondidas sem um aumento significante no custo computacional, em comparação com consultas simples a cubos de dados padrões.

Assim como o padrão SQL é utilizado para acesso aos dados em um banco de dados relacional, expressões multidimensionais (MDX - *Multidimensional Expressions*) vêm crescendo como um padrão de fato para bancos de dados multidimensionais (MLC, 2000).

A seguir apresentaremos as principais características das expressões multidimensionais.

2.2.1 Expressões Multidimensionais – MDX

Padrão introduzido pela Microsoft, expressões multidimensionais permitem que programadores OLAP acessem funções orientadas a conjunto e hierarquia, e especifiquem objetos como eixos, medidas, dimensões, e níveis utilizados em consultas OLAP. Embora seja similar a SQL na sintaxe, MDX é uma linguagem independente.

A seguir veremos a sintaxe utilizada por instruções MDX através do uso de exemplos.

Apesar dos dados estarem armazenados nos servidores OLAP de uma forma multidimensional, para a realização de consultas MDX os dados devem estar dispostos em apenas 2 eixos: coluna (eixo obrigatório) e linha (eixo opcional). A forma mais simples de uma expressão multidimensional obedece à estrutura:

```
SELECT especificação_do_eixo ON COLUMNS,        (obrigatório)
especificação_do_eixo ON ROWS                   (opcional)
FROM nome_do_cubo                               (obrigatório)
WHERE especificação_slice                       (opcional)
```

No exemplo da Figura 2.4, um cubo formado de apenas uma dimensão (*Locations*) poderia ser gerado através da instrução

```
SELECT [Locations].[Countries].MEMBERS ON COLUMNS
FROM [Sales]
```

Algumas observações podem ser feitas a partir deste exemplo:

- A palavra-chave ".MEMBERS" refere-se aos membros de um determinado nível de hierarquia (todos os países da dimensão *Location*), se por outro lado, o usuário quisesse referenciar as cidades (filhos) de um determinado país (membro) deveria utilizar a palavra chave ".CHILDREN" (ex. [Locations].[Countries].[USA].CHILDREN → Chicago e New York);
- Como não indicamos uma medida para os dados, os resultados apresentados se referem à medida definida como padrão no momento da criação do cubo de dados, para indicar uma outra medida, deveríamos utilizar a cláusula WHERE (ex. WHERE [Measures].[Sales Average]).

A seguir mostramos a instrução para o cubo de duas dimensões (*Location* e *Item*) gerados a partir da operação de *slice* do cubo central da Figura 2.4:

```
SELECT [Item].[Types].MEMBERS ON COLUMNS,
[Locations].[Cities].MEMBERS ON ROWS
FROM [Sales]
WHERE ([Measures].[Unit Sales], [Time].[Quarters].[Q1])
```

Para a realização de consultas MDX envolvendo mais de duas dimensões, deveremos utilizar a função CROSSJOIN, que produz todas as combinações entre dois conjuntos. Como exemplo tomemos o cubo central da Figura 2.4:

```
SELECT [Locations].[Cities].MEMBERS ON COLUMNS,
CROSSJOIN( {[Time].[Quarters].Members}, {[Item].[Types].Members}) ON ROWS
FROM [Sales]
WHERE [Measures].[Sales Average]
```

Para finalizar esta breve apresentação sobre consultas MDX, já que se trata de uma linguagem poderosa, com muitos recursos e inúmeras funções (que podem inclusive ser extendidas),

mostraremos o uso da cláusula WITH, que permite ao usuário criar suas próprias medidas e membros, tomando por base os dados já existentes.

Supondo que a nossa hierarquia da dimensão Tempo esteja organizada da seguinte maneira: Ano→Trimestre→Mês, e que quiséssemos apresentar os resultados das vendas por semestre, poderíamos utilizar a instrução WITH para criar os novos membros:

```
WITH
   MEMBER [Time].[S1] AS '[Time].[Quarters].[Q1] + [Time].[Quarters].[Q2]'
   MEMBER [Time].[S2] AS '[Time].[Quarters].[Q3] + [Time].[Quarters].[Q4]'
SELECT
   {[Time].[S1], [Time].[S2]} ON COLUMNS,
   [Locations].[Cities].MEMBERS ON ROWS
FROM [Sales]
```

A seguir apresentamos um exemplo de criação de uma nova medida (percentual de lucro) a partir de medidas existentes:

```
WITH
   MEMBER Measures.ProfitPercent AS
   '(Measures.[Store Sales] – Measures.[Store Cost]) / (Measures.[Store Cost])',
   FORMAT_STRING = '#.00%'
SELECT
   [Locations].[Cities].MEMBERS ON COLUMNS,
   [Time].[Months].MEMBERS ON ROWS
FROM [Sales]
WHERE Measures.ProfitPercent
```

2.3 Descoberta de Conhecimento em Bancos de Dados

A descoberta de conhecimento em bancos de dados (*KDD – Knowledge Discovery in Databases*) é um processo de identificação de padrões previamente desconhecidos, que consiste na aplicação de algoritmos de aprendizagem de máquina ou estatística, e que produz um conjunto de padrões, não triviais, interpretáveis e válidos, e potencialmente úteis, em meio aos dados (FSS, 1996). No processo de KDD está inclusa a etapa de Mineração de Dados.

Considera-se aqui *dados* como uma série de fatos, como por exemplo, registros em uma base de dados; e *padrões* como uma expressão em alguma linguagem, descrevendo um subconjunto dos dados ou um modelo que se aplica ao subconjunto. Desta forma, extrair padrões significa ajustar um modelo para os dados, encontrar alguma estrutura nos dados, ou de forma geral, fazer uma descrição de alto nível de um conjunto de dados. O termo *processo* é usado já que KDD é composto de várias etapas. Já *não-trivial* significa que busca e inferência está envolvida no processo, isto é, não é um processo de cálculo de quantidades

predefinidas, como o cálculo de uma média por exemplo. Os padrões descobertos devem ser *válidos* e possuir um certo grau de certeza. É desejável também que os padrões sejam novos e *potencialmente úteis*, trazendo algum benefício para o usuário ou para a tarefa de descoberta de conhecimento. E finalmente os padrões devem ser *compreensíveis*.

Como foi dito, o processo de KDD é interativo e iterativo em espiral, envolvendo várias etapas, nas quais muitas delas necessitam da participação direta do usuário. As primeiras iterações são ingênuas, muitas vezes trazendo resultados óbvios ou incompreensíveis. A Figura 2.5 fornece uma visão prática do processo, enfatizando a natureza interativa do mesmo.

Considerando que as etapas de seleção, limpeza, integração e transformação dos dados do processo de KDD coincidem com os passos para construção de um *data warehouse* para o processamento analítico on-line. As organizações podem então, após a construção do *data warehouse*, escolher qual método a ser adotado para análise dos dados: OLAP e/ou Mineração de Dados.

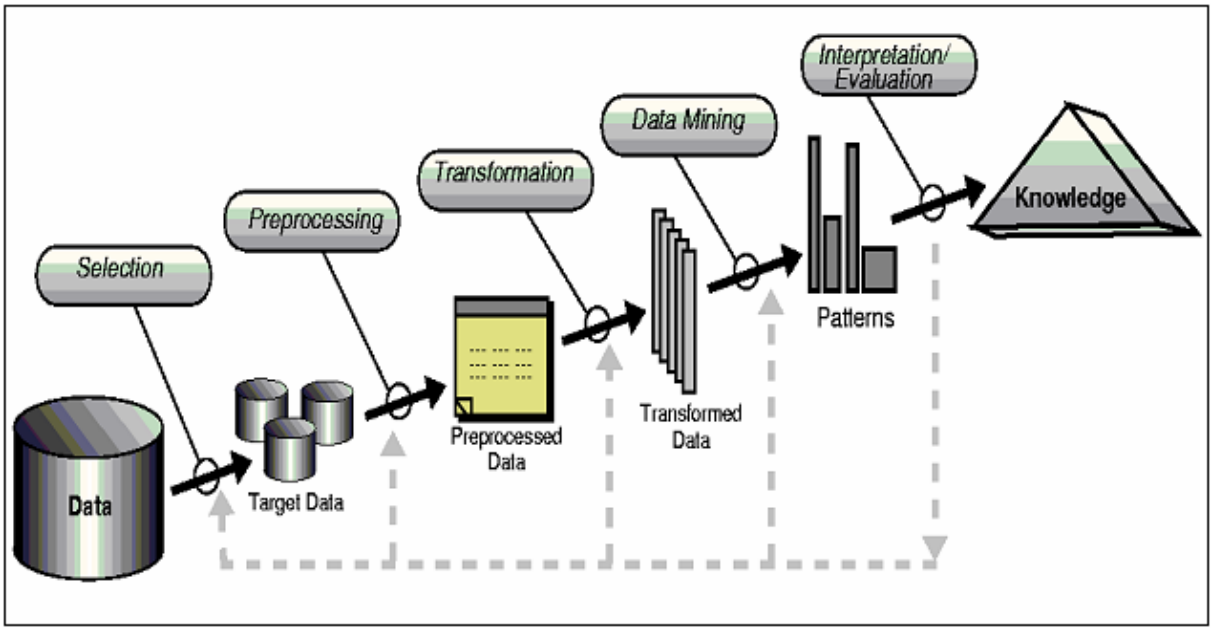


Figura 2.5 – Os passos que constituem o processo de KDD (FSS, 1996)

Antes de se iniciar o processo de KDD, a primeira coisa a ser feita é desenvolver um entendimento do domínio da aplicação e do conhecimento prévio relevante, identificando os objetivos do processo de KDD do ponto de vista do cliente. Entendemos por cliente, neste contexto, as pessoas responsáveis pelo processo de tomada de decisão dentro da organização em foco.

Discutiremos a seguir as etapas do processo de KDD, ilustrando com exemplos de uma aplicação de fraude de cartão de crédito, cujo objetivo é determinar quando usuários estão comprando itens com um cartão de crédito roubado. Para este exemplo, o propósito da análise

será identificar os clientes com padrões de uso do cartão de crédito que diferem dos padrões de uso previamente estabelecidos.

As etapas do processo de KDD são:

- Definição do objetivo da mineração: devem ser respondidas as seguintes questões: Qual tipo de conhecimento está sendo procurado para influenciar qual tipo de decisão? Qual grau de abstração ele deve possuir? Qual confiabilidade estatística? Em qual formalismo representá-lo? No nosso exemplo estamos procurando um padrão de comportamento para clientes de cartão de crédito, como saída obteríamos a resposta se determinado tipo de compra representa ou não uma fraude e como conseqüência, a compra seria ou não autorizada pela administradora do cartão.
- Seleção: diz respeito à compreensão do domínio de aplicação, do conhecimento prévio relevante, e dos objetivos do processo de KDD do ponto de vista do usuário final, resultando em um conjunto de dados alvo onde será realizado o processo de descoberta. Envolve o que os estatiscistas chamam de análise exploratória dos dados. Recorrendo-se ao exemplo do cartão de crédito, temos que grandes companhias de cartão de crédito normalmente possuem vários *sites* de processamento que são responsáveis por áreas geográficas específicas. Desta forma, um subconjunto dos dados deve ser selecionado da base de dados, pois seria desnecessário analisar todos os dados.
- Pré-processamento: na fase de pré-processamento e limpeza de dados são realizadas operações básicas de remoção de ruídos, coleta de informações necessárias para modelar ou explicar ruído nos dados, decisão de que estratégias usar para tratar campos de dados em branco, e tratar informações temporais seqüenciais assim como mudanças conhecidas. Nesta fase, pode também ser feita re-configuração de dados, para assegurar formatos consistentes e integrados. Freqüentemente dados possuem erros introduzidos em seus processos de entrada, e como os dados alvos podem ser selecionados de várias fontes, os mesmos estão sujeitos a inconsistências de vários tipos (em termos de modelo de dados, semântica dos atributos, modos de representação). No nosso exemplo, é possível que o mesmo cliente seja representado de duas maneiras diferentes, em *sites* diferentes: em um, o campo nome do cliente pode conter o primeiro nome seguido pelo último, em outro pode conter apenas o último nome. A fase de pré-processamento deve identificar essas diferenças e deixar os dados consistentes e limpos.
- Transformação: nesta fase são feitas redução e projeção de dados. Isto é alcançado encontrando-se características úteis para representá-los de acordo com o objetivo da tarefa

de mineração de dados que será usada na fase seguinte. Com os métodos de transformação e redução aplicados nesta fase, o número efetivo de variáveis em consideração pode ser reduzido, bem como representações não variantes dos dados podem ser encontradas, deixando os dados em formatos utilizáveis. Ilustrando com o exemplo do cartão de crédito, temos que os dados podem ser transformados para uso em diferentes técnicas de análise: um conjunto de tabelas individuais podem ser agrupadas em uma única tabela, um atributo que está representado de diferentes formas (data escrita como 18-03-1999 e 3/18/99) devem ser transformados para um formato comum, ou ainda um dado que está representado como texto deve ser transformado para um formato numérico, se for requerido por uma método de mineração de dados no processo.

- Mineração dos dados: consiste na escolha e aplicação de um método particular de mineração de dados, isto é, um algoritmo de aprendizagem de máquina, para a extração dos padrões de comportamento dos dados. Dentre exemplos de tipos de métodos de mineração de dados que podem ser aplicados encontram-se: sumarização, classificação, regressão, agrupamento, entre outros (FSS, 1996). No nosso exemplo, pode ser escolhido um algoritmo que irá automaticamente procurar por agrupamentos no comportamento dos dados. Este tipo de algoritmo pode encontrar, por exemplo, um conjunto de clientes que fazem relativamente pequena quantidade de compras, um conjunto de clientes que fazem grande número de compras, e um conjunto de clientes que fazem um grande número de compras em períodos muito curtos de tempo. Estes comportamentos podem ser analisados posteriormente para determinar se alguns dos padrões são representativos do comportamento de fraude de cartão. Desta forma, o algoritmo de mineração escolhido poderia caracterizar um comportamento de fraude por quantidade de compras em um curto período de tempo, ou por áreas geográficas diferentes de onde o cliente normalmente realiza compras.
- Interpretação e Avaliação: nesta etapa é feita a interpretação dos resultados, os padrões minerados, com possível retorno aos passos anteriores. Esta fase também envolve visualização dos padrões e modelos extraídos, ou dos dados de acordo com modelos descobertos. Quando um padrão é identificado, ele deve ser examinado para determinar se o mesmo é novo, relevante, e correto, por algum padrão de medida. Esta etapa pode requerer mais interação com o usuário, pois este é quem mais pode fazer determinações de relevância. Quando o padrão é considerado relevante e útil, ele pode ser considerado conhecimento, e normalmente é colocado na base de conhecimento para ser usado em

iterações subseqüentes. A base de conhecimento pode ser vista como um mecanismo de armazenamento similar a uma base de dados, usada para armazenar os conhecimentos descobertos e também informações prévias sobre o domínio.

Na Figura 2.6 mostramos os passos necessários para a realização da etapa de mineração de dados em um ambiente de *data warehouse*: dados analíticos previamente integrados, limpos, primitivos e derivados (armazenados em um *data warehouse*) são selecionados em tabelas que, após sofrerem um processo de transformação, se encontram prontas para aplicação de algoritmos de mineração. Ferramentas de mineração são então utilizadas, gerando o conhecimento minerado, que será posteriormente apresentado ao usuário.

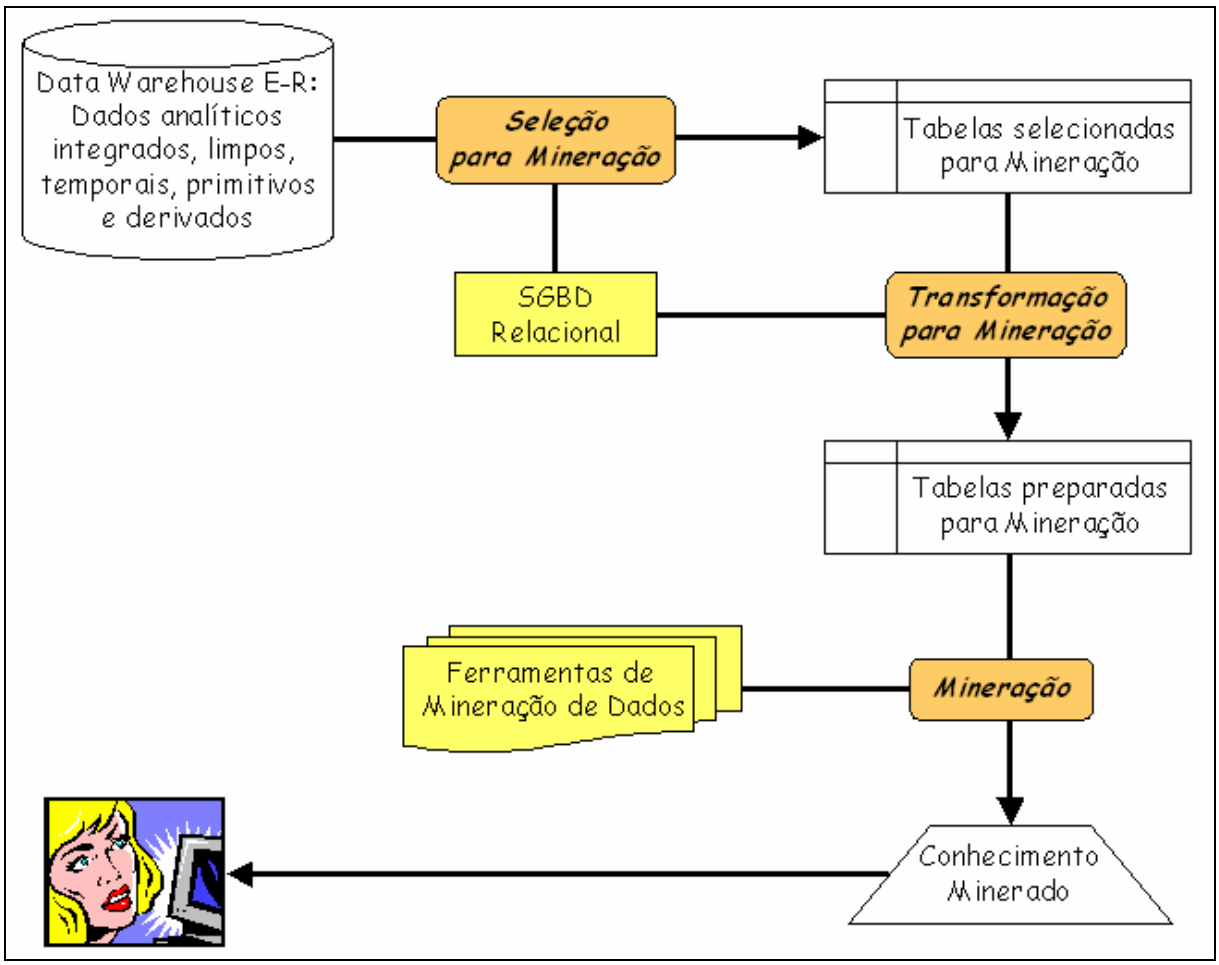


Figura 2.6 –Processo de Mineração de Dados.

Os algoritmos de mineração de dados, de acordo com o conhecimento gerado, podem ser classificados em dois tipos:

- Descritivos – descrevem conceitos ou padrões relevantes no conjunto de dados; ou
- Preditivos – baseados nos dados em análise, constroem modelos para o banco de dados e fazem previsão de tendências e de dados desconhecidos.

Os resultados obtidos através do processo de mineração de dados podem ser apresentados em diversos formatos, a depender do algoritmo de mineração empregado. A seguir mostramos as principais classes de saída de mineração de dados:

- Descrição conceitual: caracteriza e discrimina os dados – algoritmo para generalização, sumarização e descoberta de contrastes nas características dos dados (ex.: características de regiões secas e úmidas);
- Associação: identificam correlação e causalidade nos dados. Podem descobrir associações multi-dimensionais – Idade (X, “20..29”) ^ Renda (X, “20..29mil”) → Compra (X, “PC”) [suporte = 2%, confidência = 60%], ou uni-dimensionais – Comprou (T, “Computador”) → Comprou (x, “Software”) [1%, 75%];
- Classificação e predição: encontram modelos que distinguem e descrevem classes ou conceitos para previsão futura (ex.: classificação de países baseado no clima ou classificação de carros baseada no consumo médio). Os resultados são apresentados na forma de árvore de decisão, regra de classificação ou rede neural. Podem ser utilizados para prever valores desconhecidos ou em falta;
- Agrupamento: algoritmo aplicado quando se deseja agrupar dados sem conhecer os nomes das classes. Os dados são agrupados de maneira a formar novas classes (ex.: agrupamento de casas para encontrar padrões de distribuição). Baseia-se no princípio de maximizar a similaridade intra-classes e minimizar a similaridade entre classes;
- Análise de valores extremos (*outliers*): encontra objetos que não estão de acordo com o comportamento geral dos dados – valores atípicos. Útil na detecção de fraudes e análise de eventos raros.

Processos de KDD e Mineração de Dados têm sido aplicados em diversos domínios científicos e empresariais. Em ciência, uma das principais áreas de aplicação é astronomia; em negócios, dentre as principais áreas de aplicação destacam-se *marketing*, finanças (especialmente investimentos financeiros), detecção de fraudes, telecomunicações. Exemplos de aplicações de *marketing* são sistemas de análise de compras, onde é possível encontrarmos padrões (que podem ser valiosos para varejistas) do seguinte tipo: “Se o cliente compra X, então ele deverá comprar Y e Z”.

Tanto o projeto MATRIKS como OCCOM se focalizam em uma classe de saída do processo de KDD: a mineração de valores extremos (*outliers*).

2.4 OLAM

Atualmente existe um campo emergente de pesquisa que se foca na integração das tecnologias de OLAP e Mineração de Dados. O termo Mineração Analítica On-Line (*On-Line Analytical Mining* – OLAM) para se referir à integração destas duas tecnologias complementares em um ambiente de KDD foi introduzido por HAN (1997).

A mineração de conhecimento em bancos de dados multidimensionais é particularmente importante devido à complementaridade de OLAP e mineração de dados: enquanto OLAP fornece dados de alta qualidade (limpos, integrados, consistentes) necessários para a aplicação de ferramentas de análise e consultas OLAP, a mineração de dados, por sua vez, automatiza o processo de descoberta de padrões interessantes, análise e exploração do grande volume de dados disponíveis nos sistemas OLAP.

A arquitetura apresentada na Figura 2.6 pode ser adaptada para ambientes OLAM como é mostrado na Figura 2.7. Em um ambiente OLAM, as tabelas para mineração são selecionadas através de consultas a bases de dados OLAP, sobre as quais são aplicadas as ferramentas de mineração de dados. O usuário poderá então visualizar tanto os dados originais, através da realização de consultas OLAP, como o conhecimento gerado através do processo de mineração.

Existem vários desafios nesta integração, principalmente porque as ferramentas de Mineração de Dados atuais devem ser refeitas pensando-se em lidar com a representação de dados OLAP. Alguns resultados neste sentido podem ser encontrados no sistema *DBMiner* (HAN, 1997), e em pesquisas sobre métodos de Mineração de Dados no contexto OLAP em AAD (1996).

Esta proposta de integração entre OLAP e Mineração de Dados é bastante promissora porque se beneficia das principais vantagens da representação, organização e consultas multidimensionais de dados OLAP: visão histórica e multidimensional dos dados, interatividade, alto desempenho, uso de operações específicas para navegação nos dados; e da análise inteligente de dados proporcionado pela tecnologia de Mineração de Dados. A principal característica desta integração é a capacidade de minerar subconjuntos de dados em múltiplos níveis de abstração e granularidade que OLAP dá suporte.

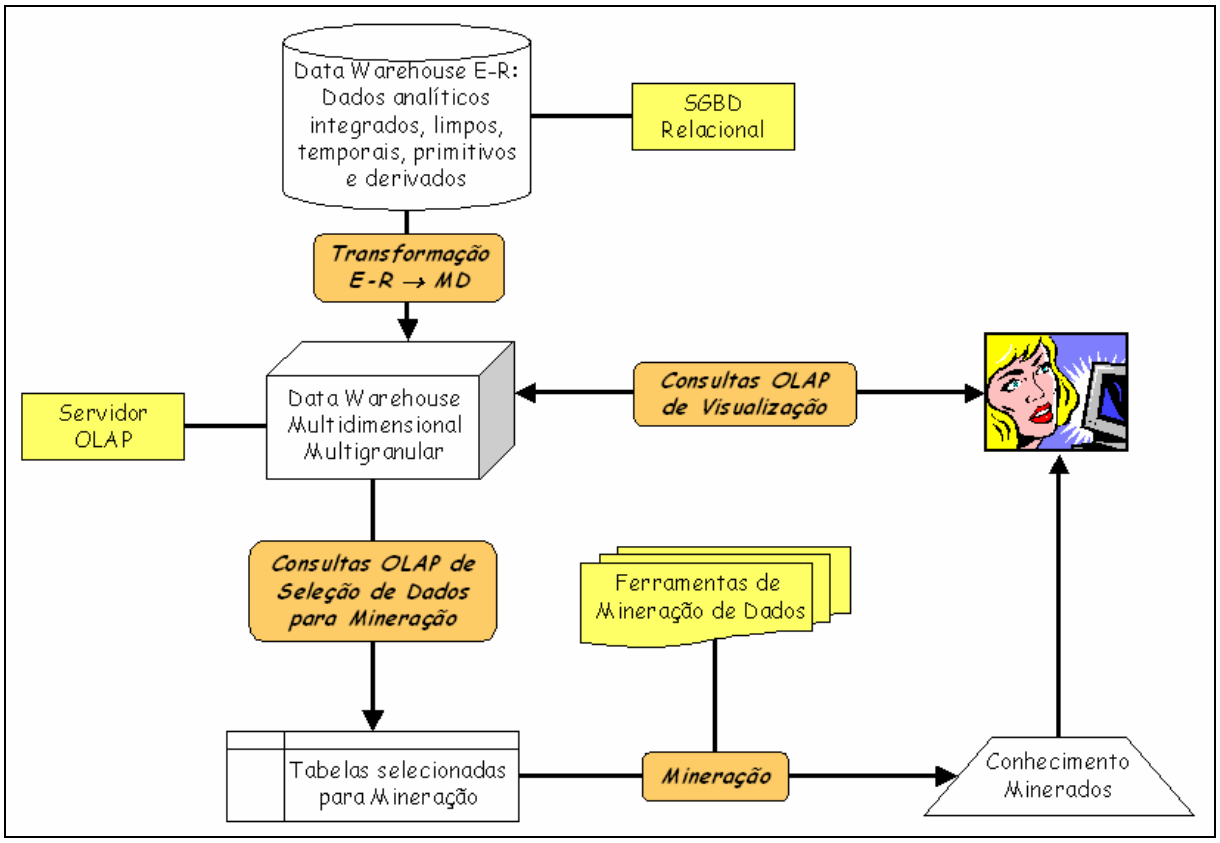


Figura 2.7 – OLAM – Integração OLAP x Mineração de Dados.

2.5 O Ambiente MATRIKS

Como visto anteriormente, o processo de KDD é inerentemente exploratório, interativo e iterativo, com cada uma das etapas obrigatoriamente revisitadas e redefinidas várias vezes com os benefícios dos novos *insights* parciais gerados pela iteração precedente. É também um processo intensamente cognitivo, cooperativo e demorado e, conseqüentemente, requer o uso de ferramentas de representação e gerenciamento do conhecimento.

Todas essas características apontam para sistemas de KDD com arquitetura de software aberta e altamente evolutiva, baseada no encapsulamento de serviços específicos em componentes reutilizáveis e sua livre e imediata conexão via um vasto leque de APIs (*Application Program Interface*) e padrões de comunicação.

No entanto, os desenvolvedores de software na área persistem na direção oposta. Eles continuam a oferecer ferramentas de mineração especializadas, sem API, como se a tarefa de mineração de dados fosse auto-contida e não parte do complexo e multi-facetado processo de KDD; além disso, os ambientes de descoberta de conhecimento são monolíticos e fechados, o

que impede sua personalização por parte do usuário, e sua extensão com novas funcionalidades.

Na Figura 2.8 apresentamos uma arquitetura ideal para um ambiente de KDD, formada pelas seguintes camadas:

- Camada de dados: formada basicamente por bancos de dados transacionais e outras fontes de dados como arquivos Web, HTML e XML, que, após passar pelos passos de limpeza, integração e transformação dos dados, irão constituir os bancos de dados multidimensionais (*data warehouses*) que dão suporte ao processamento analítico on-line (OLAP) e a ferramentas de mineração de dados. Note que o conhecimento produzido volta para a camada de dados para formar o que chamamos de Base de Conhecimento.
- Camada de processamento: conjunto de ferramentas responsáveis pelas etapas de pré-processamento (integração, limpeza e transformação dos dados) e pela descoberta do conhecimento em si (ferramentas de mineração).
- Camada de interface: formada pela API integrada para software externo, que permite a integração com outras ferramentas, contribuindo para a expansão do sistema; e pela interface responsável pela interação do usuário com o ambiente de KDD, por onde ele poderá acompanhar as etapas desenvolvidas e visualizar os resultados obtidos.

MATRIKS (*Multidimensional Analysis and Textual Summarizing for Insight Knowledge Search*) é um projeto do CIn/UFPE que visa o desenvolvimento de um ambiente abrangente e aberto para KDDW (*Knowledge Discovery in Data Warehouses*) (FAVERO, 2000; ROBIN; FAVERO, 2000) e que pretende melhorar o estado da arte em sistemas de suporte à decisão, baseado nos seguintes princípios de desenvolvimento:

- Integração em um único ambiente de uma diversidade maior de sistemas computacionais do que os sistemas atualmente existentes, através do desenvolvimento segundo uma arquitetura de software moderna, aberta e extensível por meio de encapsulamento de serviços em componentes, comunicando-se através de uma interface de software aplicativo (API – *Application Program Interface*);
- Integração de ferramentas de mineração de dados e OLAP, resultando em OLAM (HAN; KAMBER, 2001);
- Integração das tecnologias de *data warehousing*, bancos de dados multidimensionais, mineração de dados e geração automática de hipertextos em linguagem natural.

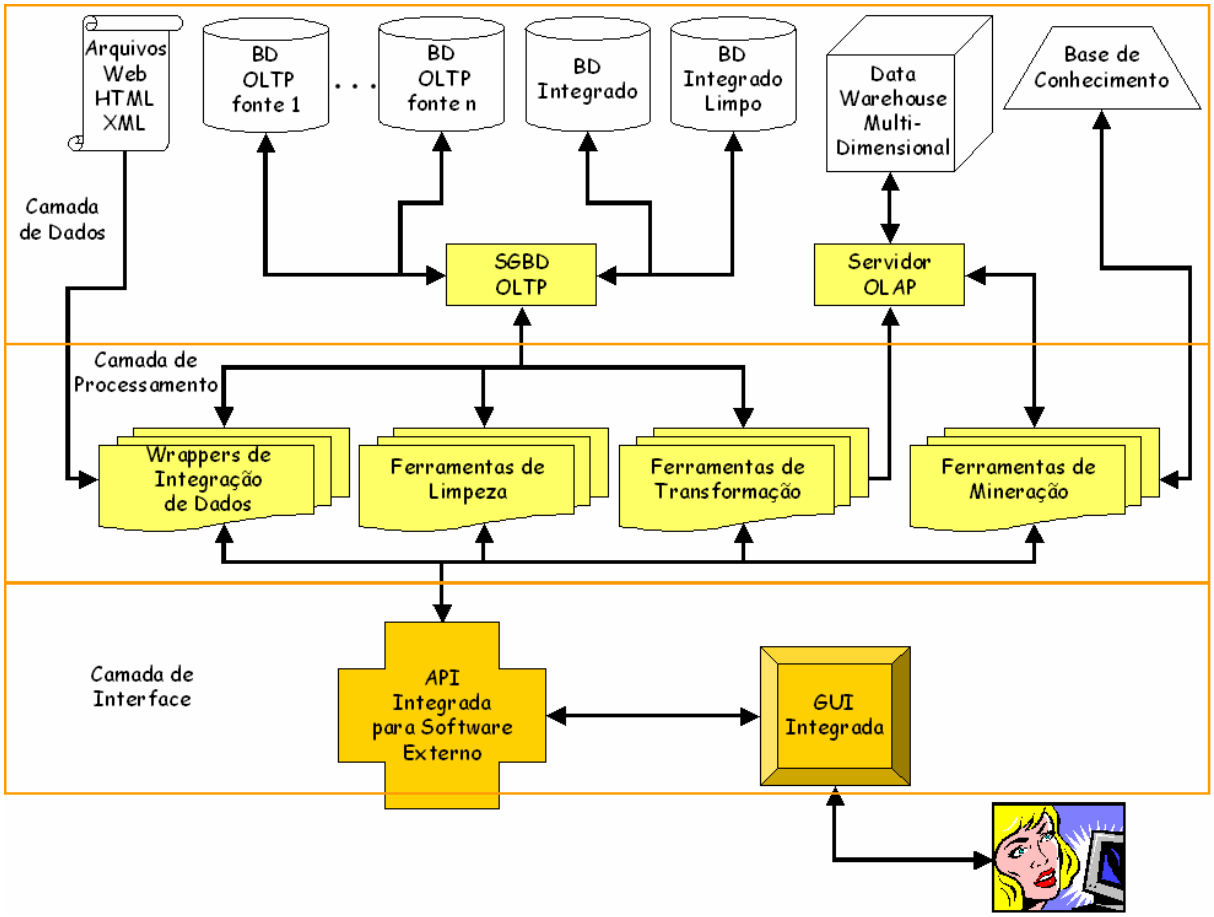


Figura 2.8 – Arquitetura ideal de um ambiente de KDDW

A arquitetura dos sistemas atuais de suporte à decisão é monolítica, o que impede, tanto sua extensão com novos serviços pelo usuário, como sua comunicação com software externo em um sistema maior. A falta de integração dos vários serviços computacionais necessários no processo exploratório, iterativo e interativo de KDD constitui hoje o maior gargalo nesse processo.

A arquitetura dos componentes de software do projeto MATRIKS pode ser vista na Figura 2.9.

Na arquitetura, o papel principal de OCCOM (Minerador de Exceções) é localizar anomalias em todos os níveis de agregação em cubos de dados OLAP, fornecendo objetos enriquecidos por mineração que auxiliarão usuários a explorarem cubos OLAP eficientemente.

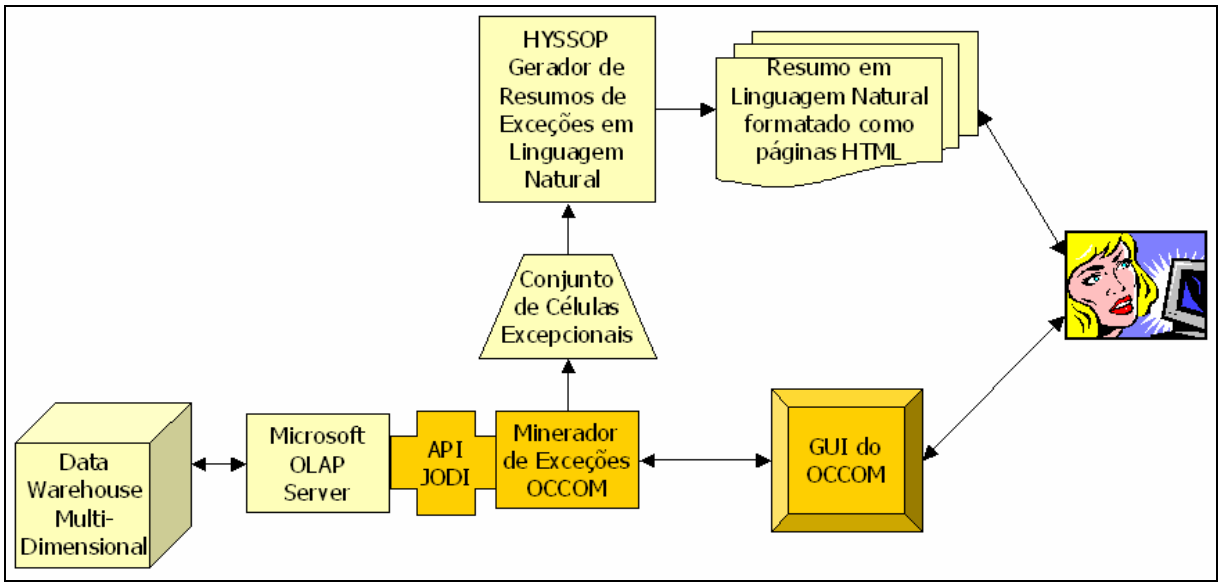


Figura 2.9 – Arquitetura do projeto MATRIKS

2.5.1 HYSSOP

HYSSOP (*Hypertext Summary System for OLAP*) é um sistema que gera resumos de *insights* decisórios obtidos por KDD no formato de um hipertexto em linguagem natural. Constitui-se da sinergia entre duas tecnologias: Geração de Linguagem Natural (*NLG – Natural Language Generation*) e Descoberta de Conhecimento em Bancos de Dados (KDD) (FAVERO, 2000; ROBIN; FAVERO, 2001).

Nenhum gerador de texto em linguagem natural anterior ao projeto MATRIKS era capaz de gerar resumos em formato hipertexto. Alguns geram resumos textuais lineares e outros hipertextos sem preocupação de concisão. Além do mais, a extensibilidade e portabilidade de todos era extremamente limitada.

O foco de HYSSOP é na sumarização de um único tipo de conhecimento minerado: valores excepcionais de agregações em um espaço analítico n-dimensional representados como cubóides OLAP. Um exemplo de entrada para HYSSOP pode ser visto na Figura 2.10. Esta tabela de entrada é chamada matriz de conteúdo. Cada linha da matriz corresponde a uma célula do cubo, cujo valor mostra um desvio significante do valor da média composto de todas as partições formadas pela interseção com esta célula, em todos os possíveis níveis de agregação *roll-up*. A matriz de conteúdo contém a coordenada da célula ao longo das dimensões analíticas do cubóide, o valor da medida da célula, o seu grau de excepcionalidade

(alto, médio ou baixo), e as médias agregadas para as partições do cubóide que possuem interseção com a célula.

A tarefa de HYSSOP é gerar um hipertexto que analisa e sumariza o contexto da matriz para tomada de decisão. No entanto, o contexto da matriz é apenas uma parte da entrada de HYSSOP, ele especifica o que deve ser dito no hipertexto a ser gerado, mas não diz como fazê-lo. Para esta tarefa, é utilizada uma linguagem declarativa para especificar estratégias de discursos de alto nível, que diz como HYSSOP irá sumarizar a informação da matriz de conteúdo. Através desta linguagem, chamada de DOSL (*Discourse Organization Specification Language*), o analista pode especificar como ele deseja que as exceções mineradas serão agrupadas e ordenadas por HYSSOP.

Um exemplo dos resultados apresentados por HYSSOP pode ser visto na Figura 2.11. A próxima página (Figura 2.12), é acessada através do segundo *hyperlink* desta página inicial (abaixo de 40%). Todas as outras páginas seguem esta mesma estrutura.

Uma das principais contribuições de OCCOM é gerar entradas para HYSSOP, ligando-o com *OLE DB for OLAP*[5]. Considerando que HYSSOP foi implementado na linguagem de programação LIFE[6], torna-se necessário o desenvolvimento de uma interface Java/LIFE que permita a conexão entre os dois módulos do ambiente MATRIKS.

	Analytical Dimensions				Measures	Mining Result	Roll -up means		
Cell	Product	Place	Time	Sales Variation Sign	Sales Variation Value	Exception Degree	Product	Place	Time
1c	Birch Beer	nation	Nov	-	10	low	3	2	4
2c	Jolt Cola	nation	Aug	+	6	low	0	3	-7
3c	Birch Beer	nation	Jun	-	12	low	2	5	3
4c	Birch Beer	nation	Sep	+	42	high	-2	1	1
5c	Cola	central	Aug	-	30	low	7	-5	-1
6c	Diet Soda	east	Aug	+	10	low	-5	7	-8
7c	Diet Soda	east	Sep	-	33	medium	-1	0	7
8c	Diet Soda	east	Jul	-	40	high	-1	5	8
9c	Diet Soda	south	Jul	+	19	low	1	-1	-11
10c	Diet Soda	west	Aug	-	17	low	2	4	1
11c	Cola	Colorado	Sep	-	32	medium	-2	2	2
12c	Cola	Colorado	Jul	-	40	medium	-1	4	0
13c	Cola	Wisconsin	Jul	+	11	low	0	3	7

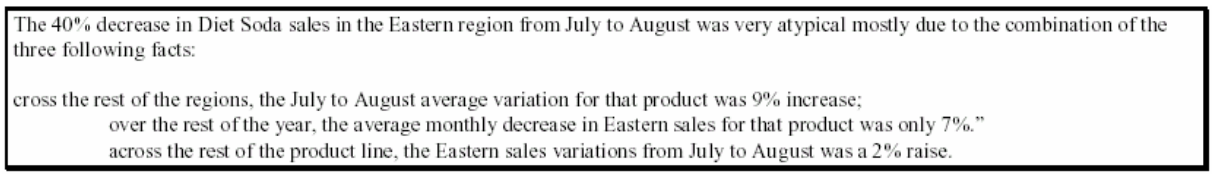
Figura 2.10 – Exemplo de um entrada para HYSSOP (ROBIN; FAVERO, 2001).

[5] Conjunto de objetos e interfaces que permite o acesso a dados multidimensionais armazenados em um servidor OLAP da Microsoft (Microsoft, 1998).

[6] LIFE é uma linguagem de programação que busca a integração de estilos das linguagens funcionais e lógicas, cujo foco principal é o processamento de linguagem natural (AÏT-KACI; LINCOLN, 1989).

Last year, the most atypical sales variations from one month to the next occurred for:
Birch Beer with a 42% national increase from September to October;
Diet Soda with a 40% decrease in the Eastern region from July to August.
At the next level of idiosyncrasy came:
Cola´s Colorado sales, falling 40% from July to August and then a further 32% from September to October;
again Diet Soda Eastern sales, falling 33% from September to October.
Less aberrant but still notably atypical were:
again nationwide Birch Beer sales' -12% from June to July and -10% from November to December;
Cola's 11% fall from July to August in the Central region and 30% dive in Wisconsin from August to September;
Diet Soda sales´ 19% increase in the Southern region from July to August, followed by its two opposite regional variations from August to September, +10% in the East but -17% in the West;
national Jolt Cola sales' +6% from August to September.
To know what makes one of these variations unusual in the context of this year's sales, click on it.

Figura 2.11 – Página inicial de saída em formato de hipertexto gerada por HYSSOP (ROBIN; FAVERO, 2001)



The 40% decrease in Diet Soda sales in the Eastern region from July to August was very atypical mostly due to the combination of the three following facts:

cross the rest of the regions, the July to August average variation for that product was 9% increase;
over the rest of the year, the average monthly decrease in Eastern sales for that product was only 7%."
across the rest of the product line, the Eastern sales variations from July to August was a 2% raise.

Figura 2.12 – Página HYSSOP acessada através de *hyperlink* da página inicial (ROBIN; FAVERO, 2001)

3 TRABALHOS RELACIONADOS

Neste capítulo serão discutidos os trabalhos relacionados ao tema desta dissertação – a descoberta de valores atípicos em cubos de dados OLAP, e a análise de modelos para acesso a *data warehouses* dimensionais.

3.1 Mineração de Exceções em Cubos OLAP

Exceções são uma pequena porção de dados que fogem radicalmente do padrão geral seguido pelos outros dados de um determinado contexto ou amostra, e que são freqüentemente tratados como ruídos. Entretanto, “um ruído para uma pessoa é um sinal para outra”. Algumas vezes eventos raros são mais importantes que eventos comuns. Analistas de negócios que estão navegando em um cubo de dados OLAP estão freqüentemente procurando por exceções, porque as exceções geralmente levam a identificação de áreas problemáticas ou novas oportunidades (CHEN, 1999). Por exemplo, um aumento nas vendas de um produto para uma determinada faixa etária em uma certa época do ano, pode representar uma oportunidade a ser explorada ou, pelo contrário, pode representar o resultado de um campanha de vendas que não alcançou o resultado esperado, mas que, de qualquer forma, foge ao comportamento tradicional das vendas daquele produto naquele período.

Quando o usuário utiliza as operações dos cubos OLAP (*drill-down*, *roll-up*, *slice, dice*), o grande volume de valores de dados isolados tornam difícil a visualização de exceções através de processo não automático. Ele precisa guiar a exploração do *data warehouse* por intuição e hipóteses próprias baseadas na sua perícia e conhecimento do domínio em análise, para detectar exceções implícitas nos dados. Para cubos com muitas dimensões e muitos níveis de hierarquia ao longo de cada dimensão, esse processo exploratório manual requer a especificação e avaliação dos resultados de um número combinatorialmente explosivo de consultas OLAP, inviabilizando o uso prático de tal abordagem.

O processo exploratório guiado pela descoberta é uma abordagem alternativa em que medidas pré-computadas que apontam exceções nos dados são utilizadas para guiar o usuário no processo de análise dos dados, em todos os níveis de agregação. Intuitivamente, uma exceção é o valor de uma célula de um cubo de dados que é significantemente diferente de um valor

antecipado, baseado em um modelo estatístico. O modelo considera variações e padrões no valor medido através de todas as dimensões a que a célula pertence. Por exemplo, se a análise das vendas de um item revela um aumento das vendas no mês de dezembro em comparação a todos os outros meses, isto poderia ser visto como uma exceção na dimensão tempo. Entretanto, isto não é uma exceção se a dimensão item é considerada, já que ocorre um aumento similar nas vendas para outros itens durante dezembro. O modelo considera exceções escondidas em todas as agregações *group-by* do cubo de dados. Indicações visuais, como a cor de fundo, são utilizadas para refletir o grau de exceção em cada célula, baseado em um indicador de exceção pré-computado. Desta forma, o indicador de exceção pode ser utilizado para guiar a descoberta de anomalias interessantes nos dados (HAN; KAMBER, 2001).

Cada célula é marcada em todas as suas possíveis agregações no cubo de dados com medidas que indicam o grau de "surpresa" que a quantidade na célula possui. Estas medidas indicam quão anômalo o valor em uma célula é em relação a outras células. Este grau de surpresa é composto por três valores descritos a seguir:

1. *SelfExp*: representa o grau de surpresa de uma célula em relação a outras células do mesmo nível de agregação.
2. *InExp*: representa o grau de surpresa encontrado em alguma célula abaixo da célula considerada se nós realizarmos um *drill-down* a partir da mesma.
3. *PathExp*: representa o grau de surpresa para cada dimensão sobre a qual executarmos um *drill-down* a partir da célula considerada.

O uso destas medidas para exploração do cubo de dados é ilustrado pelo exemplo a seguir.

Supondo que se queira analisar as vendas mensais de uma empresa em termos de diferença percentual em relação ao mês anterior. As dimensões envolvem item, tempo e região. Inicia-se o estudo pelos dados agregados sobre *todos os itens e regiões* para cada mês, como visto na Figura 3.1.

Item	all
Região	all

Soma das Vendas	Mês											
	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Total		1%	-1%	0%	1%	3%	-1%	-9%	-1%	2%	-4%	3%

Figura 3.1 – Mudança nas vendas sobre o tempo (HAN; KAMBER, 2001)

Os valores de *SelfExp* e *InExp* são traduzidos em efeitos visuais, mostrados em cada célula. A cor de fundo de cada célula é baseada no valor de *SelfExp*. Uma caixa é desenhada em torno de cada célula, onde a largura e cor da caixa são funções do seu valor *InExp*. Caixas largas indicam alto valor de *InExp*. Em ambos os casos, quanto mais escura a cor, maior o grau de

exceção. Por exemplo, a caixa escura e larga das vendas durante julho, agosto e setembro avisam ao usuário para explorar o nível mais baixo de agregação destas células através da operação de *drill-down*.

A operação de *drill-down* pode ser executada através das dimensões de agregação Item ou Região. Para saber qual dimensão seguir, ao selecionar uma célula, o valor de *PathExp* da célula irá refletir o grau de exceção de cada caminho. Vamos supor que o caminho Item contenha mais exceções.

Média de Vendas	Mês											
Item	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Sony b/w printer		9%	-8%	2%	-5%	14%	-4%	0%	41%	-13%	-15%	-11%
Sony color printer		0%	0%	3%	2%	4%	-10%	-13%	0%	4%	-6%	4%
HP b/w printer		-2%	1%	2%	3%	8%	0%	-12%	-9%	3%	-3%	6%
HP color printer		0%	0%	-2%	1%	0%	-1%	-7%	-2%	1%	-4%	1%
IBM home computer		1%	-2%	-1%	-1%	3%	3%	-10%	4%	1%	-4%	-1%
IBM laptop computer		0%	0%	-1%	3%	4%	2%	-10%	-2%	0%	-9%	3%
Toshiba home comp.		-2%	-5%	1%	1%	-1%	1%	5%	-3%	-5%	-1%	-1%
Toshiba laptop comp.		1%	0%	3%	0%	-2%	-2%	-5%	3%	2%	-1%	0%
Logitech mouse		3%	-2%	-1%	0%	4%	6%	-11%	2%	1%	-4%	0%
Ergo-way mouse		0%	0%	2%	3%	1%	-2%	-2%	-5%	0%	-5%	8%

Figura 3.2 – Mudança nas vendas para cada combinação item-tempo (HAN; KAMBER, 2001)

Uma operação *drill-down* através da dimensão Item resulta no cubo da Figura 3.2, mostrando as vendas ao longo do tempo para cada item. Considerando a diferença de vendas de 41% para "*Sony b/w printers*" em setembro, esta célula possui uma cor de fundo escura, indicando um alto valor de *SelfExp*, significando que esta célula é uma exceção.

A seguir veremos dois modelos estatísticos para computação de exceções em cubos OLAP que serviram como base para desenvolvimento deste trabalho.

3.1.1 Medidas e Algoritmo de Sarawagi, Agrawal e Meggido

Um valor em uma célula de um cubo de dados é considerado uma exceção se ele difere significantemente de um valor antecipado calculado utilizando-se um modelo que leva em consideração todas as agregações (*group-bys*) em que o valor participa.

Sarawagi et.al. (1998) observaram os seguintes fatores para a elaboração de seu modelo:

- A necessidade de considerar variações e padrões no valor da medida ao longo de todas as dimensões a que a célula pertence. Isto nos ajuda a encontrar valores excepcionais dentro do contexto de uma agregação em particular;

- A necessidade de encontrar exceções em todos as possíveis agregações do cubo, e não somente no nível mais detalhado, facilitando o entendimento do usuário final através de uma representação concisa;
- O usuário deve possuir a habilidade para interpretar a razão pela qual certos valores são marcados como exceção: um usuário típico de OLAP é um executivo de alguma organização, não necessariamente um estatístico sofisticado;
- O procedimento para encontrar exceções deve ser computacionalmente eficiente e escalável para grandes volumes de dados, comum em bases de dados OLAP.

O Modelo

Para um valor $y_{i1,i2,\ldots,in}$ em um cubo C na posição i_r da r-ésima dimensão d_r ($1 \le r \le n$), definimos um valor antecipado $\hat{y}_{i1,i2,\ldots,in}$ como uma função f de contribuição das várias agregações de níveis mais altos como:

$$\hat{y}_{i1i2\ldots in} = f\left(\gamma^{G}_{(i_r | d_r \in G)} \mid G \subset \{d_1, d_2 \,\mathrm{K}\, d_n\}\right) \text{ (valor antecipado) (3.1)}$$

Iremos nos referenciar aos γ termos como os coeficientes da equação do modelo. Veremos adiante como estes coeficientes são derivados e as diferentes formas que a função f pode tomar.

Ilustramos a seguir o caso para um cubo com três dimensões A, B e C. O valor antecipado $\hat{y}_{ijk}$ para o i-ésimo membro da dimensão A, j-ésimo membro da dimensão B e k-ésimo membro da dimensão C, é expresso como uma função de sete termos obtidos a partir de cada uma das sete agregações (*group-bys*) do cubo, como:

$$\hat{y}_{ijk} = f\left(\gamma, \gamma^{A}_{i}, \gamma^{B}_{j}, \gamma^{C}_{k}, \gamma^{AB}_{ij}, \gamma^{BC}_{jk}, \gamma^{AC}_{ik}\right) \text{ (coeficientes da equação)}$$

A Figura 3.3 mostra os termos (agregações) gerados a partir de um cubo com três dimensões, e a forma como são organizados. Por exemplo, o termo γ (*All*) pode ser obtido através das agregações γ^{A}_{i}, γ^{B}_{j} ou γ^{C}_{k}; e o termo γ^{A}_{i} pode ser obtido pelas agregações γ^{AB}_{ij} ou γ^{AC}_{ik}.

A diferença absoluta entre o valor atual, $y_{i1,i2,\ldots,in}$ e o valor antecipado $\hat{y}_{i1,i2,\ldots,in}$ é chamado valor residual do modelo. Portanto,

$$r_{i1i2\ldots in} = \left| y_{i1i2\ldots in} - \hat{y}_{i1i2\ldots in} \right| \text{ (valor residual)}$$

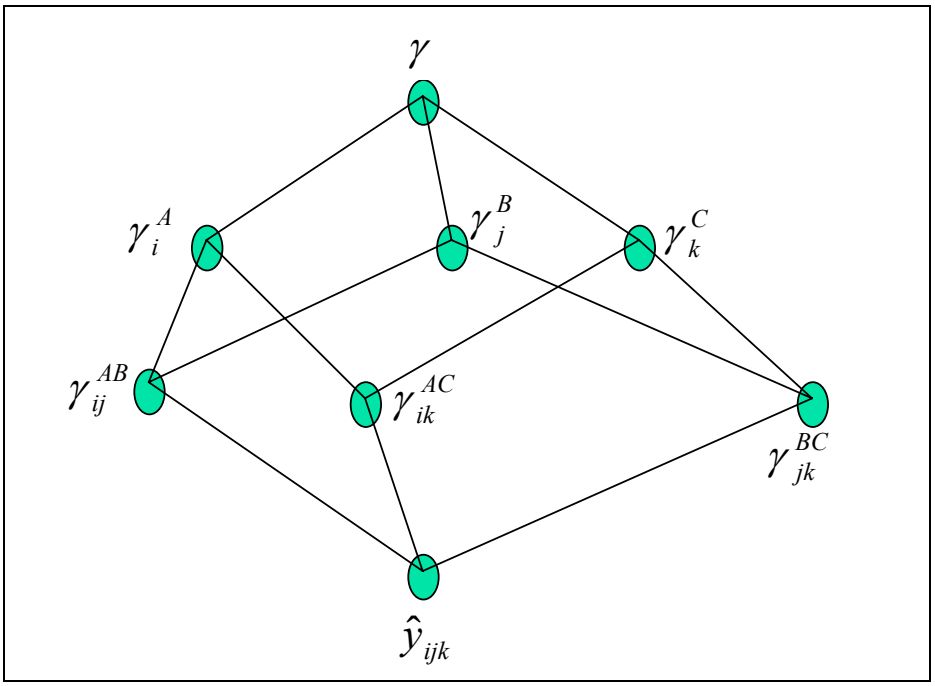


Figura 3.3 – Termos agregados para um cubo com três dimensões.

Qualquer valor com um resíduo de tamanho relativamente alto é uma exceção. Uma definição estatisticamente válida para "relativamente alto" requer que utilizemos uma escala de valores baseadas também no desvio padrão antecipado $\sigma_{i_1,i_2,\mathrm{K},i_n}$ associado ao resíduo. Então, nós consideramos $y_{i1,i2,\ldots,in}$ uma exceção se

$$s_{i1i2\ldots in} = \frac{|\, y_{i1i2\ldots in} - \hat{y}_{i1i2\ldots in} \,|}{\sigma_{i1i2\ldots in}} \quad > \quad \tau \qquad (3.2)$$

onde é predefinido algum limiar τ. A utilização de $\tau = 2.5$ corresponde a uma probabilidade de 99% em uma distribuição normal, ou seja, probabilidade de 1% de tal desvio ser devido ao acaso, sem viés causal subjacente.

Formas Funcionais de *f*

A função f na equação 3.1 pode tomar uma das seguintes formas:

- Aditiva: a função f retorna a soma de seus argumentos.
- Multiplicativa: a função f retorna o produto de seus argumentos.

Segundo Sarawagi et.al., em suas experiências com dados OLAP, a forma multiplicativa fornece um melhor enquadramento que a forma aditiva, conforme justificativa apresentada em seu artigo (SARAWAGI, et.al, 1998, p.8). Para simplificar o cálculo, transforma-se a forma multiplicativa para uma forma aditiva linear tomando-se o logaritmo dos valores dos dados originais. Gerando a fórmula:

$$\hat{l}_{i1i2...in} = \log \hat{y}_{i1i2...in} = \sum_{G \subset \{d_1, d_2, \mathrm{K}\, d_n\}} \gamma^{G}_{(i_r | d_r \in G)}$$ (coeficientes da equação) (3.3)

Para um cubo tridimensional, esta equação toma a forma:

$$\hat{l}_{ijk} = \log \hat{y}_{ijk} = \gamma + \gamma^{A}_{i} + \gamma^{B}_{j} + \gamma^{C}_{k} + \gamma^{AB}_{ij} + \gamma^{BC}_{jk} + \gamma^{AC}_{ik}$$

onde cada um dos termos é gerado a partir de operações de *roll-up* (agregação) do cubóide base.

Para ilustrar a aplicação deste modelo, a Figura 3.4 mostra um cubo de dados com três dimensões (*Gender*, *Marital Status* e *Time*). A dimensão *Time* possui três níveis de hierarquia (*Semester* → *Quarter* → *Month*). A partir dele, podem ser gerados os seguintes cubóides: *Time X Gender, Time X Marital Status, Gender X Marital Status* (com duas dimensões); *Time, Gender, Marital Status* (com uma dimensão); e o cubóide *All* (maior nível de agregação). A quantidade de cubos gerados pode ser obtida através da fórmula 2^n-1, onde *n* corresponde ao número de dimensões.

O primeiro passo a ser realizado é o cálculo de *log* $y_{i1,i2,...,in}$, conforme a Figura 3.4. A partir dos valores obtidos serão gerados cada um dos termos da equação 3.3.

		Gender	F		M	
Time		Marital Status	M	S	M	S
S1	Q1	1	5.920,00	5.012,00	5.057,00	5.639,00
		2	5.261,00	5.005,00	5.230,00	5.461,00
		3	5.609,00	6.103,00	6.024,00	5.970,00
	Q2	4	4.981,00	5.009,00	4.915,00	5.274,00
		5	5.312,00	5.224,00	4.968,00	5.577,00
		6	5.050,00	5.416,00	5.660,00	5.224,00
S2	Q3	7	5.911,00	5.935,00	5.756,00	6.161,00
		8	5.105,00	5.315,00	5.852,00	5.425,00
		9	5.150,00	5.183,00	5.041,00	5.014,00
	Q4	10	4.643,00	4.863,00	5.069,00	5.383,00
		11	6.054,00	6.226,00	6.188,00	6.762,00
		12	6.340,00	6.891,00	6.700,00	6.865,00

		Gd	F		M	
Time		MS	M	S	M	S
S1	Q1	1	8,6861	8,5196	8,5285	8,6375
		2	8,5681	8,5182	8,5622	8,6054
		3	8,6321	8,7165	8,7035	8,6945
	Q2	4	8,5134	8,5190	8,5000	8,5705
		5	8,5777	8,5610	8,5108	8,6264
		6	8,5271	8,5971	8,6412	8,5610
S2	Q3	7	8,6846	8,6886	8,6580	8,7260
		8	8,5380	8,5783	8,6745	8,5988
		9	8,5468	8,5531	8,5254	8,5200
	Q4	10	8,4431	8,4894	8,5309	8,5910
		11	8,7085	8,7365	8,7304	8,8191
		12	8,7546	8,8380	8,8099	8,8342

$\hat{l}_{i1i2...in} = \log \hat{y}_{i1i2...in}$

Figura 3.4 – Cubóide base com 3 dimensões para cálculo de exceções.

Estimativas dos Coeficientes do Modelo

Os coeficientes da equação do modelo (3.3) são estimados com base na média: para derivação destas estimativas, assume-se que os logaritmos dos valores são distribuídos de forma normal com a mesma variância. A abordagem a seguir produz os mínimos quadrados estimados neste caso:

- $\gamma = \lambda_{+,,+}$ que é correspondente a média. Observe que um "+" no *i*-ésimo índice denota uma agregação ao longo da dimensão *i*.
- $\gamma_{i_r}^{A_r} = \lambda_{+..+i_r+..+} - \gamma$ onde $\lambda_{+..+i_r+..+}$ é a média sobre todos os valores ao longo do *i*-ésimo membro da dimensão A_r. Então, $\gamma_{i_r}^{A_r}$ denota o quanto a média dos valores ao longo do *i*-ésimo membro da dimensão A_r difere da média geral.
- $(\gamma)_{i_r i_s}^{A_r A_s} = \lambda_{+,,+i_r+,,+i_s+,,+} - \gamma_{i_r}^{A_r} - \gamma_{i_s}^{A_s} - \gamma$

Geralmente, os coeficientes correspondentes a qualquer agregação *G* são obtidos pela subtração do valor da média λ da agregação *G* de todos os coeficientes de nível de agregação mais alto (*roll-up*). Intuitivamente, os coeficientes refletem um ajuste da média da agregação correspondente após todos os ajustes dos níveis mais altos terem sido considerados.

A abordagem dos coeficientes estimados com base na média apresenta como problema o fato de não ser robusta na presença de valores extremamente grandes (*outliers*). Embora vários métodos tenham sido propostos, Sarawagi et.al. utilizam a média cortada a 75%, onde 25% dos valores extremos são cortados e a média é calculada com base nos 75% dos números centrais. Pela remoção de 25% dos valores extremos, o método torna-se robusto a *outliers*.

Para a computação do cubo utiliza-se um método conhecido como *Up-Down*, por ser divido em duas fases:

- Fase de subida (*Up-Phase*): são computados os valores das médias λde cada agregação partindo do nível mais detalhado (cubóide base), utilizando o método de computação do cubo[1] de Agarwal et.al.(1996). Podemos observar através da Figura 3.4 como esta tarefa é realizada. Note que os valores das células de um cubóide podem ser computados a partir de diferentes outros cubóides. O método de Agarwal et.al., entre outras coisas, indica qual a melhor opção.

[1] Método para computação de agregações em cubos OLAP que tenta minimizar o número de acessos a disco pela sobreposição da computação dos vários cubóides. Utiliza algoritmo para redução do número de passos de ordenação necessários ao cálculo das agregações (Agarwal, 1996).

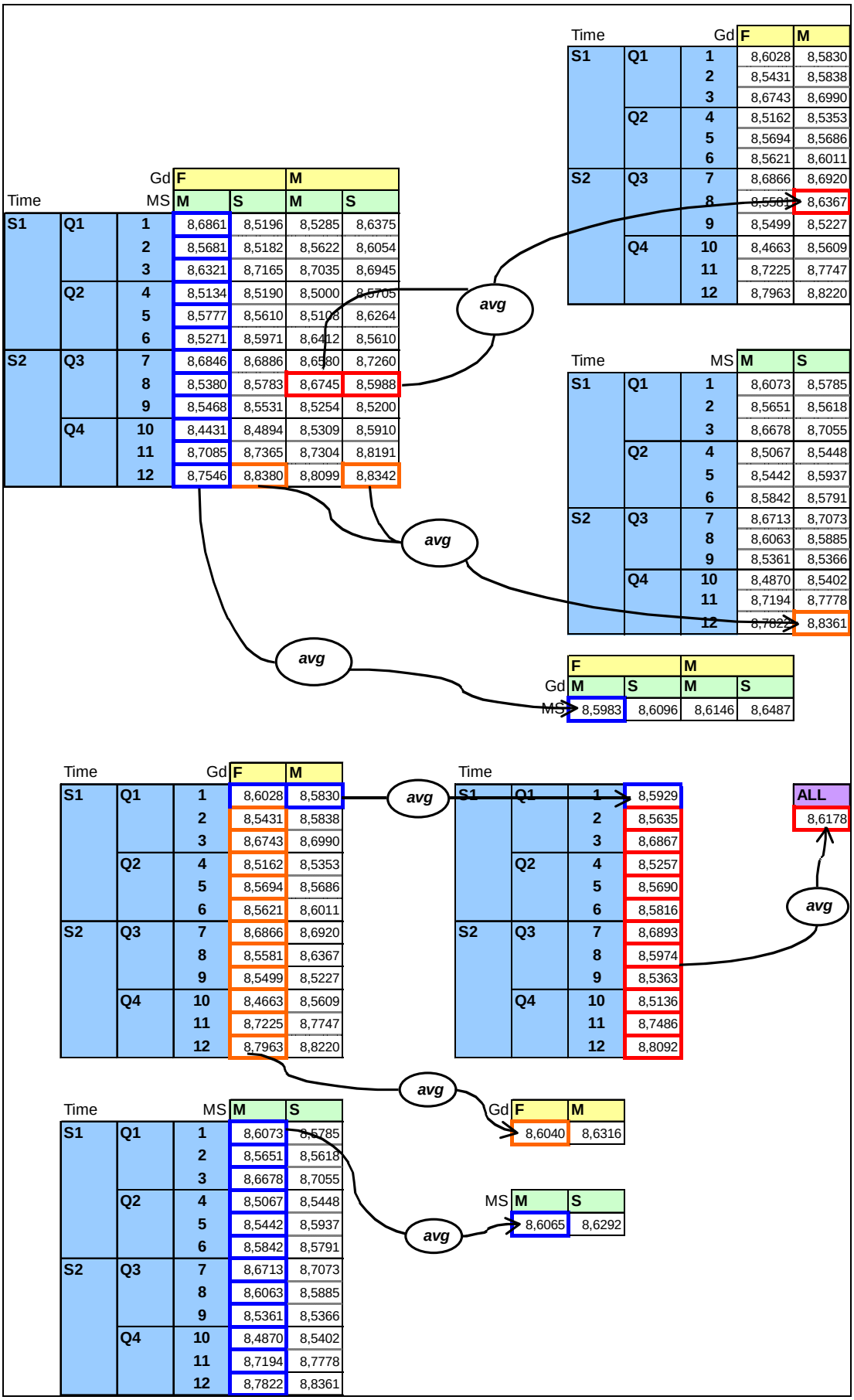


Figura 3.5 – *Up-Phase*

- Fase de descida (*Down-Phase*): que subtrai de cada agregação *G* os coeficientes de todos os seus subconjuntos iniciando a partir do nível menos detalhado (*ALL*), como está demonstrado na Figura 3.6. Por exemplo, o valor do coeficiente da célula *[F].[S]* do cubóide de duas dimensões *Gender X Marital Status* é calculado pela subtração do seu valor médio $\lambda = 8.6096$ dos coeficientes de suas agregações e da média geral γ: $\gamma_{[F][S]}^{[Gd][MS]} = \lambda_{[F][S]} - \gamma_{[F]}^{[Gd]} - \gamma_{[S]}^{[MS]} - \gamma = 8.6096 - (-0.0138) - (0.0113) - (8.6178) = -0.0057$.

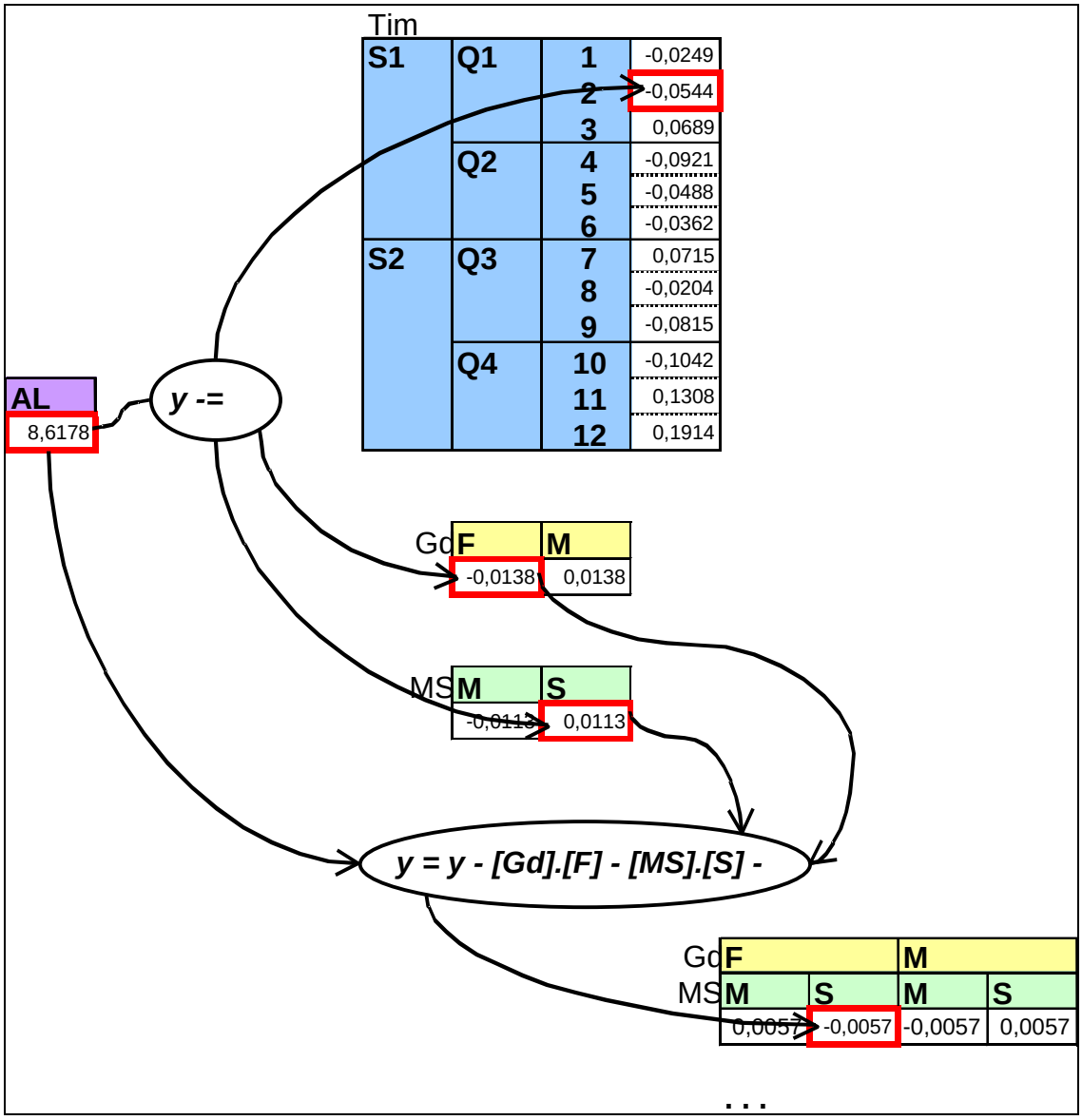


Figura 3.6 – *Down-Phase*

Após o cálculo de todos os coeficientes, os valores estimados para as células do cubo base podem ser calculadas através da fórmula 3.3, como está demonstrado na Figura 3.7.

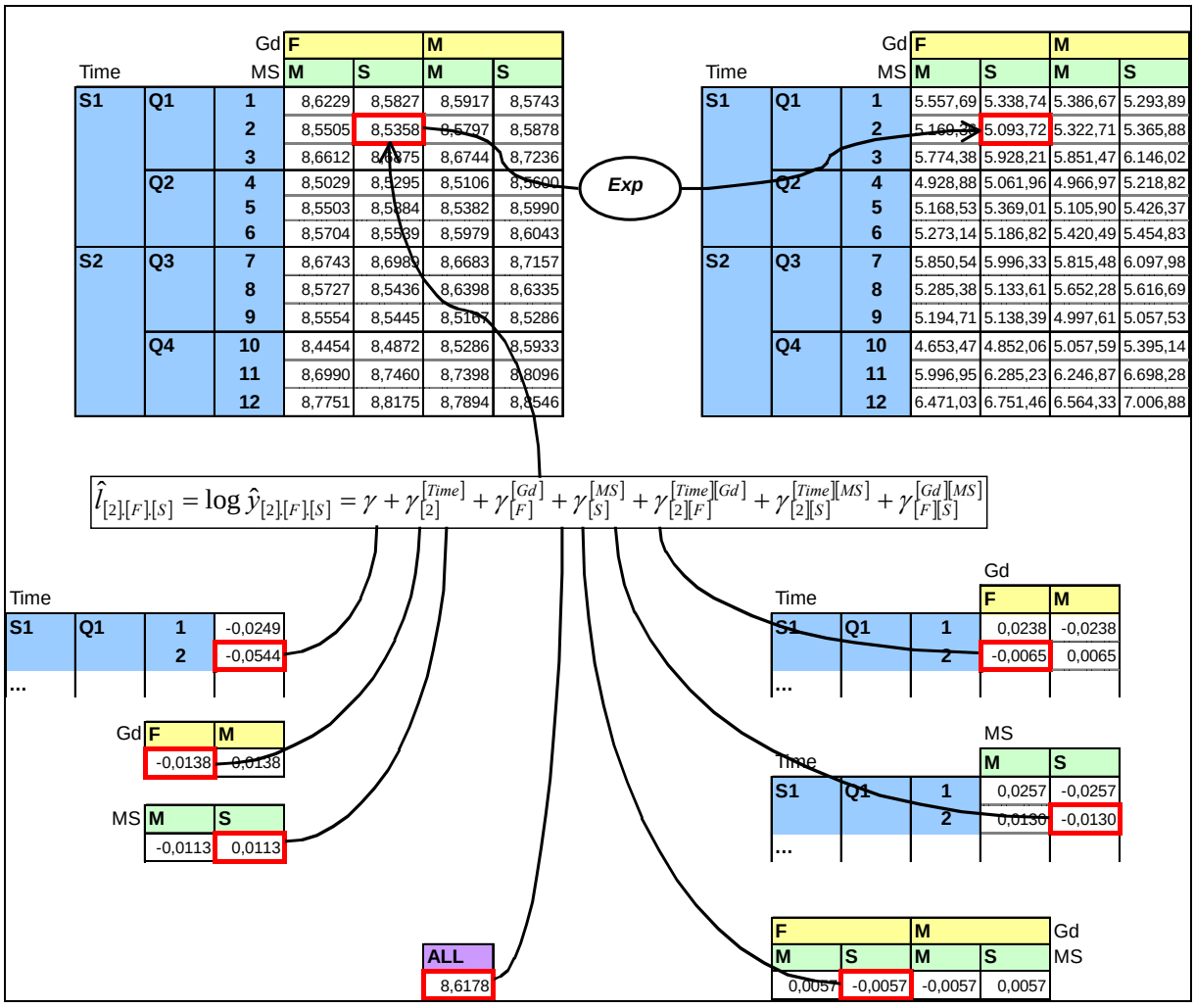


Figura 3.7 – Cálculo dos valores estimados das células do cubo.

Estimativa do Desvio Padrão

Em métodos clássicos de análise de variância, o desvio padrão de todas as células é assumido como idêntico. A variância (quadrado do desvio padrão) é estimada como a soma dos quadrados dos resíduos dividido pelo número de entradas, conforme a fórmula:

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{(n-1)} \quad (3.4)$$

Este método não é adequado quando se tratando de dados OLAP. Em análise de tabelas de contingência[2] (BISHOP et al., 1975), onde entradas das células representam quantidades, a

[2] As tabelas de contingência são utilizadas para estudar a relação entre duas variáveis categóricas descrevendo a freqüência das categorias de uma das variáveis relativamente às categorias de outra.

distribuição de Poisson[3] é presumida. Esta concepção implica que a variância seja equivalente à média. Quando as entradas não são quantidades (ex. valores muito altos em dólar), isto tipicamente leva a uma sub-estimativa da variância.

O método utilizado por Sarawagi et al. para estimar a variância é baseado em uma leve modificação do modelo citado. Em seu modelo, a variância é uma potência do valor da média $\hat{y}_{i1,i2,\ldots,in}$:

$$\sigma^2_{i_1 i_2 \mathrm{K}\, i_n} = \left(\hat{y}_{i_1 i_2 \mathrm{K}\, i_n}\right)^p \quad (3.5)$$

Para calcular *p* utiliza-se o princípio de probabilidade máxima[4] (COOLEY; LOHNES, 1986) em dados que se supõem serem normalmente distribuídos com o valor médio $\hat{y}_{i_1 i_2 \mathrm{K}\, i_n}$. De acordo com o citado, uma possível derivação para o valor estimado de *p* deve satisfazer:

$$\sum \frac{\left(y_{i_1 i_2 \mathrm{K}\, i_n} - \hat{y}_{i_1 i_2 \mathrm{K}\, i_n}\right)^2}{\left(\hat{y}_{i_1 i_2 \mathrm{K}\, i_n}\right)^p} \cdot \log \hat{y}_{i_1 i_2 \mathrm{K}\, i_n} - \sum \log \hat{y}_{i_1 i_2 \mathrm{K}\, i_n} = 0 \quad (3.6)$$

A solução da equação 3.6, que permite encontrar o valor de *p*, deve ser feita de forma iterativa, ou seja, o resultado da equação é avaliado para um conjunto de valores de *p* (geralmente 10, igualmente espaçados entre 0 e 3). O valor final é o ponto intermediário entre os dois pontos em que o sinal da equação muda, ou seja, na iteração *i*, na qual o sinal muda,

$$p = \frac{\left|p_i - p_{i-1}\right|}{2} + p_{i-1}.$$

Voltando ao nosso exemplo, o valor de *p* que melhor satisfaz a equação 3.6 é 1.185 (o valor de *p* que zera a equação é 1.173773), resultando nos valores de σ da tabela na Figura 3.8.

---

[3] Esta distribuição descreve eventos raros, em que se faz um enorme número de tentativas e aplica-se à situação em que o evento (ou entidade) de interesse está homogeneamente distribuído na população. Pelo processo de Poisson, a ocorrência de um evento em um intervalo de espaço ou de tempo não tem qualquer efeito sobre a probabilidade de ocorrência de um segundo evento, ou seja, a ocorrência dos eventos é independente.

[4] Segundo o princípio de probabilidade máxima, a função que é melhor aproximação para um conjunto de medidas é a que torna máxima a probabilidade de ocorrência de tal conjunto de medidas. Um caso particular é o problema de calcular uma média a partir de *n* medidas de uma mesma grandeza *y*, sendo que cada medida y_i está afetada de um erro σ_i. O problema consiste em obter a *melhor aproximação* $\hat{y}$ para a quantidade não conhecida *y*. De acordo com o método de "probabilidade máxima", a melhor aproximação $\hat{y}$ para a quantidade de *y* é tal que torna máxima a probabilidade de se obter o conjunto de *n* medidas.

		Gender	F		M	
Time	Marital Status		M	S	M	S
S1	Q1	1	157,85	154,17	154,98	153,41
		2	151,28	149,98	153,90	154,63
		3	161,44	163,95	162,70	167,46
	Q2	4	147,11	149,43	147,78	152,13
		5	151,27	154,68	150,19	155,65
		6	153,06	151,58	155,55	156,13
S2	Q3	7	162,68	165,05	162,11	166,69
		8	153,27	150,67	159,42	158,83
		9	151,72	150,75	148,31	149,35
	Q4	10	142,23	145,76	149,35	155,13
		11	165,06	169,67	169,07	176,13
		12	172,60	176,95	174,06	180,85

Figura 3.8 – Valores de σ, calculados através da equação 3.6.

Aplicando-se a equação 3.2 para cálculo dos valores residuais do modelo, obtemos os dados da Figura 3.9. Observe que não há nenhuma célula excepcional ($s > \tau$).

		Gender	F		M	
Time	Marital Status		M	S	M	S
S1	Q1	1	2,19	2,02	2,03	2,15
		2	0,58	0,56	0,57	0,59
		3	0,98	1,02	1,01	1,00
	Q2	4	0,34	0,34	0,34	0,35
		5	0,90	0,89	0,88	0,92
		6	1,39	1,44	1,47	1,41
S2	Q3	7	0,35	0,35	0,35	0,36
		8	1,12	1,15	1,19	1,15
		9	0,28	0,28	0,28	0,28
	Q4	10	0,07	0,07	0,07	0,07
		11	0,33	0,33	0,33	0,34
		12	0,72	0,75	0,74	0,75

Figura 3.9 – Valores residuais do modelo.

Em seguida, o processo é repetido para todos os cubóides formados a partir do cubo base.

Hierarquias

A equação do modelo (3.3), utilizada para calcular o valor esperado, pode ser expandida a fim de manusear hierarquias ao longo de uma ou mais dimensões do cubo. A idéia básica é definir o valor antecipado baseado não apenas na posição da célula (linha e coluna) mas também em seus pais ao longo das hierarquias. Por exemplo, considerando os valores λ_{ij} em um cubo consistindo de duas dimensões A e B onde a dimensão A possui dois níveis de hierarquias: $A^1 \rightarrow A^2 \rightarrow ALL$. Para se calcular um valor antecipado na agregação A^1B, em adição ao coeficiente da linha $\gamma_i^{A^1}$ na agregação A^1, o coeficiente da coluna γ_j^B na agregação B e o

coeficiente geral γ na agregação *ALL*, nós temos dois novos termos correspondentes a duas novas agregações A^2 e A^2B ao longo da hierarquia de *A*. A equação modificada é:

$$\hat{\lambda}_{ij} = \gamma + \gamma_i^{A^1} + \gamma_j^{B} + \gamma_{i'}^{A^2} + \gamma_{i'j}^{A^2B}$$

onde i' denota o pai de i no nível de hierarquia A^2, $\gamma_{i'}^{A^2}$ denota a contribuição do i-ésimo valor na agregação A^2 e $\gamma_{i'j}^{A^2B}$ denota a contribuição combinada do $i'j$-ésimo valor na agregação A^2B. Estes coeficientes adicionais têm o efeito de dividir conceitualmente a tabela original de duas direções *A* e *B* em um conjunto de sub-tabelas A^2B correspondentes a diferentes valores de A^1. Isto se justifica uma vez que esperamos que valores pertencentes ao mesmo grupo de hierarquia tenham comportamento similar.

A quantidade de cubos a serem gerados, em um modelo com hierarquias, pode ser calculada através da fórmula $\prod_{r=1}^{d}(m_r+1)-1$. Em nosso exemplo, a dimensão *Time* possui três níveis de hierarquia, e *Gender* e *Marital Status*, um nível, portanto deveriam ser gerados ((3+1)x(1+1)x(1+1)) -1 = 15 cubóides. A Figura 3.10 mostra os demais cubóides gerados.

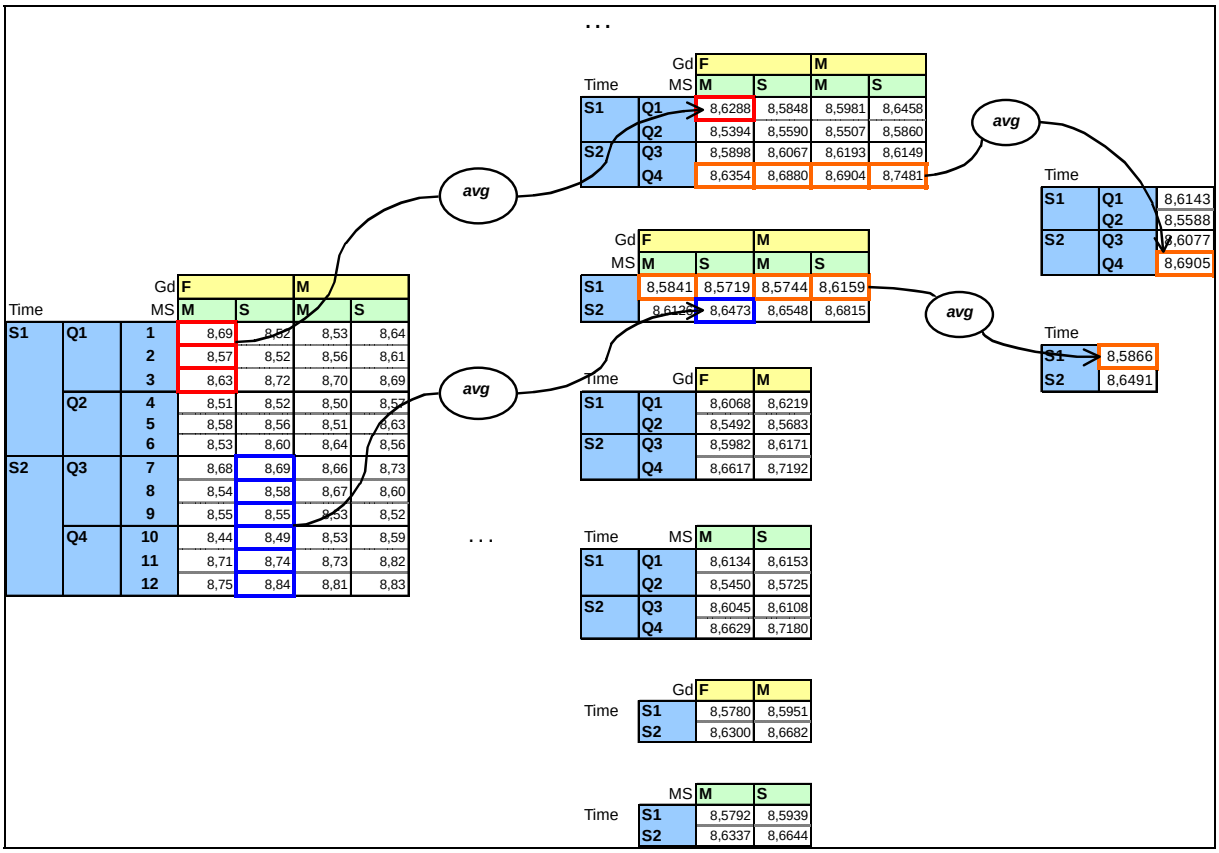


Figura 3.10 – Cubóides gerados com o uso de hierarquias.

Neste caso, a equação dos valores estimados (3.3) toma a forma:

$$\hat{l} = \log \hat{y}_{ijk} = \gamma + \gamma_i^{[Time].[M]} + \gamma_{i'}^{[Time].[Q]} + \gamma_{i''}^{[Time].[S]} + \gamma_j^{[Gd]} + \gamma_k^{MS} +$$
$$+ \gamma_{ij}^{[Time].[M]+[Gd]} + \gamma_{i'j}^{[Time].[Q]+[Gd]} + \gamma_{i''j}^{[Time].[S]+[Gd]} +$$
$$+ \gamma_{ik}^{[Time].[M]+[MS]} + \gamma_{i'k}^{[Time].[Q]+[MS]} + \gamma_{i''k}^{[Time].[S]+[MS]} +$$
$$+ \gamma_{jk}^{[Gd]+[MS]} + \gamma_{i'jk}^{[Time].[Q]+[Gd]+[MS]} + \gamma_{i''jk}^{[Time].[S]+[Gd]+[MS]}$$

Reescrita

Sarawagi et.al. também desenvolveram um método para computação *eficiente* de exceções. Técnicas de reescrita são utilizadas a fim de reduzir o custo computacional da descoberta de exceções. Estas otimizações permitem o cálculo de diferentes equações para cada uma das diferentes agregações do cubo, praticamente ao mesmo tempo em que agregações são computadas. O processo de busca de exceções envolve o mesmo tipo de agregações e operações de busca que agregações pré-computadas normais. A exceção por si mesma pode ser armazenada, indexada e recuperada da mesma forma como agregações pré-computadas.

Em vez dos 2^n-1 termos da equação 3.3, os valores esperados podem ser expressos como uma soma de *n* termos como a seguir:

$$\hat{\lambda}_{i_1 \mathrm{K}\, i_n} = g^1 + \mathrm{K} + g^n \text{, onde } g^r = avg_{i_r}\left(\lambda_{i_1 \mathrm{K}\, i_n} - g^1 - \mathrm{K} - g^{r-1}\right) \quad (3.7)$$

Como um exemplo, considere um cubo com três dimensões *A*, *B*, *C*.

$$\hat{\lambda}_{ijk} = g_{ij}^1 + g_{ik}^2 + g_{jk}^3 \text{, onde}$$
$$g_{ij}^1 = avg_k(\lambda_{ijk})$$
$$g_{ik}^2 = avg_j(\lambda_{ijk} - g_{ij}^1)$$
$$g_{jk}^3 = avg_i(\lambda_{ijk} - g_{ij}^1 - g_{ik}^2)$$

Hierarquias podem ser incorporadas a esta equação facilmente. A equação continua contendo *k* termos para um cubo *k*-dimensional. A única diferença é que em vez de computarmos as médias para todos os valores ao longo de uma dimensão, trabalhamos apenas com os elementos que pertencem ao próximo nível de hierarquia da dimensão.

A utilização deste método traz uma série de benefício, listados a seguir:

- Nós computamos os resíduos de um cubo com *k* dimensões pela junção com pelo menos *k* outras agregações, em vez de 2^k – 1 agregações como no modelo anterior – uma diferença exponencial no número de operações de junção;

- Podemos computar os resíduos ao mesmo tempo em que computamos os valores agregados da fase 1, salvando custos de ordenação e comparação;
- Não há custo adicional de ordenação na computação do cubo, já que, diferentemente do método *UpDown*, a operação de subtração é imediatamente seguida da operação de agregação.

Um exemplo mostrando o método de reescrita pode ser visto na Figura 3.11. Note que apesar das 15 agregações (cubóides) possíveis a partir do cubo base, só foi preciso utilizar 6 agregações para o cálculo dos valores estimados das células.

Medidas

Como visto anteriormente, é necessário sumarizar exceções nos níveis mais baixos do cubo como um único valor no nível mais alto. Para todas as células mostradas, o usuário deve ser hábil a encontrar: (1) quais células são exceções; (2) em quais células é necessário a realização de uma operação de *drill-down* para encontrar exceções nos níveis mais baixos e, para cada célula, qual o melhor caminho para encontrarmos exceções. Estas questão são respondidas por três tipos de medidas: *SelfExp*, *InExp* e *PathExp*. Em termos do modelo, os valores definidos pelas expressões *SelfExp*, *InExp* e *PathExp* tomam a forma descrita abaixo:

- *SelfExp*: denota o valor de exceção da célula. Este valor é definido como uma escala do valor absoluto do resíduo definido na equação 2 com a remoção do limiar τ. Formalmente,

$$SelfExp(y_{i1i2\mathrm{K}\,in}) = \max\left(\frac{|\, y_{i1i2...in} - \hat{y}_{i1i2...in}\,|}{\sigma_{i1i2...in}} - \tau, 0\right)$$

- *InExp*: denota o grau máximo de surpresa sobre todos os elementos cobertos por *drill-downs* a partir da célula.
- *PathExp*: denota o grau de surpresa a ser antecipado se executarmos um *drill-down* através de um caminho particular, para cada caminho *drill-down* possível de uma célula. *PathExp* é definido como o maior *SelfExp* sobre todas as células cobertas por um *drill-down* ao longo de determinado caminho.

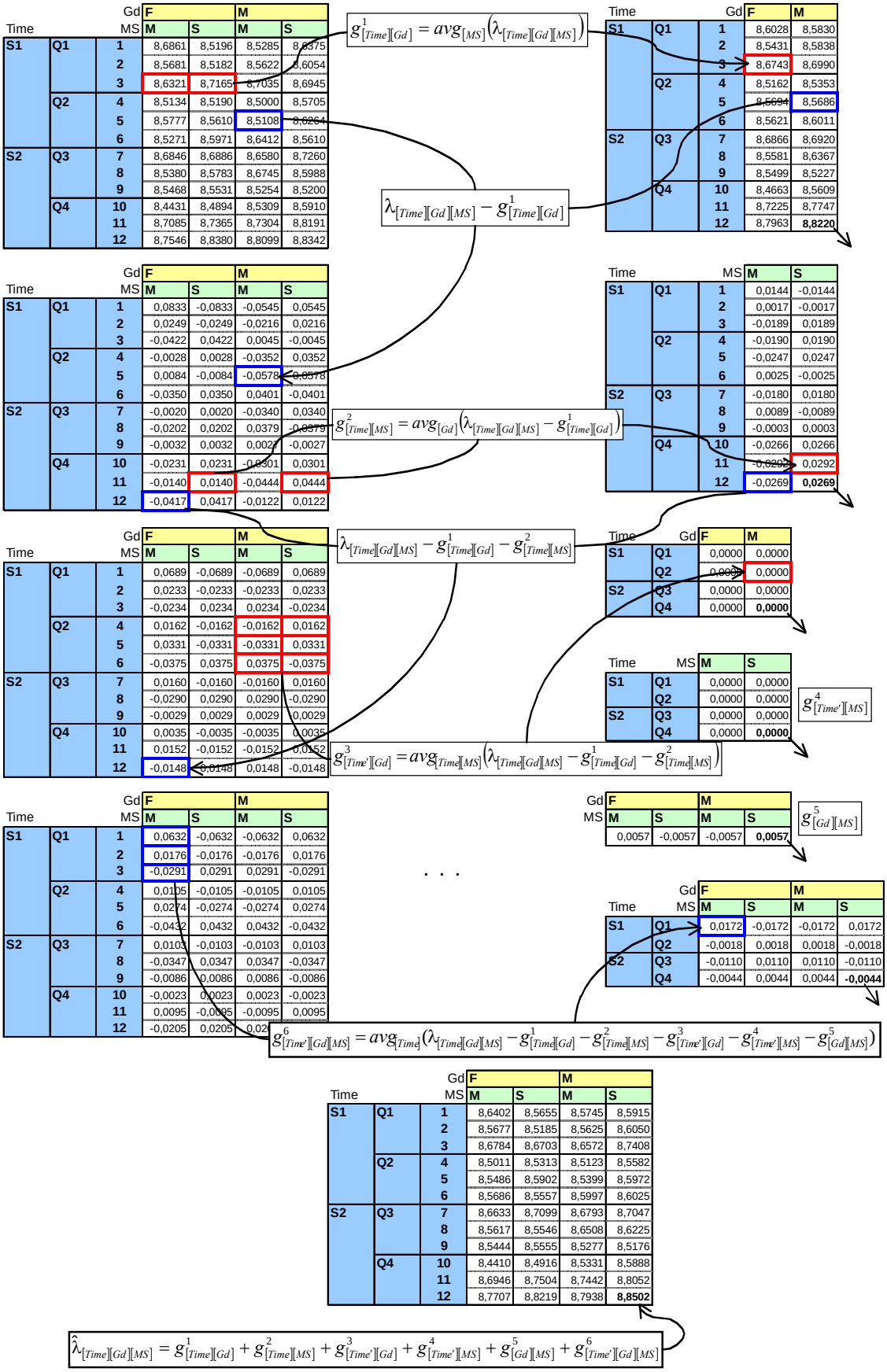


Figura 3.11 – Método de reescrita com hierarquias.

3.1.2 Medidas e Algoritmo de Chen

Da mesma forma que o algoritmo de Sarawagi et al., a idéia básica do algoritmo de Chen (1999) é adicionar indicadores de exceções automaticamente computados para todos os níveis do cubo de dados com a finalidade de ajudar o usuário a eficientemente identificar e explorar células excepcionais.

A forma da função *f* na equação 3.1 pode ser uma das seguintes:

- Aditiva: *f* retorna a soma de seus argumentos. Modelos deste tipo são chamados de lineares.
- Multiplicativa: *f* retorna o produto de seus argumentos. Modelos deste tipo são chamados log-lineares.
- Complexa: *f* é uma mistura das formas aditiva e multiplicativa. São geralmente chamados de modelos mistos.

Segundo Chen, os modelos mistos não são práticos para cubos com mais de 2-3 dimensões, por causa da sobrecarga na computação do cubo. A principal diferença do modelo de Chen em relação ao modelo introduzido por Sarawagi et al. é o fato de utilizar modelos diferentes a depender do tipo de função de agregação:

- Quando a função de agregação (medida) é uma quantidade (*count*) ou soma (*sum*), o modelo log-linear introduzido por Sarawagi et al. e mostrado na subseção anterior, é a melhor escolha;
- Quando a função de agregação é baseada na média (*average*) o modelo linear, introduzido por Chen, traz melhores resultados.

O Modelo Linear

Como a resolução da equação para o modelo log-linear já foi vista anteriormente, trataremos aqui apenas da resolução da equação para o modelo linear a que Chen chama de Modelo Linear com Pesos (*Weighted Linear Model*):

$$\hat{y}(i) = f\left(\lambda^{G}_{i_r | D_r \in G)} \mid G \subset \{D_1, D_2, \mathrm{K}\ , D_n\}\right)$$

O valor antecipado para um célula na posição (*i*, *j*, *k*) das dimensões *A*, *B*, *C*, será

$$\hat{y}(i,j,k) = f\left(\lambda^{\phi}, \lambda^{A}(i), \lambda^{B}(j), \lambda^{C}(k), \lambda^{AB}(i,j), \lambda^{BC}(j,k), \lambda^{AC}(i,k)\right)$$

Os coeficientes da equação são calculados da seguinte maneira:

- $\lambda^{\phi} = y_{+\Lambda+}$, que é a média de todos os valores *y*;
- $\lambda^{D_r}(i_r) = y_{+\Lambda+i_r+\Lambda+} - \lambda^{\phi}$, onde $y_{+\Lambda+i_r+\Lambda+}$ é a média dos valores *y* cuja coordenada na dimensão D_r é i_r;
- $\lambda^{D_r D_s}(i_r, i_s) = y_{+\Lambda+i_r+\Lambda+i_s+\Lambda+} - \lambda^{D_r}(i_r) - \lambda^{D_s}(i_s) - \lambda^{\phi}$, onde $y_{+\Lambda+i_r+\Lambda+i_s+\Lambda+}$ é a média dos valores *y* cujas coordenadas nas dimensões D_r e D_s são i_r e i_s;
- ...

O modelo acima pode ser utilizado se quisermos encontrar exceções nos valores das células sem considerar as quantidades de cada célula. Para considerar as linhas que formam cada célula (transações) devemos utilizar uma nova equação em que as tuplas que formam o cubo são os pesos da equação. Detalhes sobre o modelo podem ser vistos em (CHEN, 1999, p. 50-53).

Estimativa do Desvio Padrão

O desvio padrão para a célula, necessário para resolução da equação 3.2, pode ser estimado como

$$\begin{aligned} \sigma_y(i) &= \sigma_{\bar{x}}(i) \\ &= \frac{s_x(i)}{\sqrt{N(i)}} \end{aligned} \qquad (3.8)$$

A forma como esta equação é resolvida pode ser vista em (CHEN, 1999, p. 57-59), não mostrada aqui por não ser adotada em nosso trabalho, já que OCCOM não tem acesso às transações que formam os valores agregados (pesos no modelo linear com pesos de Chen).

Hierarquias

Hierarquias podem ser introduzidas na equação do modelo linear, de maneira simples, da mesma forma como introduzidas no modelo de Sarawagi et al.

Medidas

Diferentemente de Sarawagi et al., Chen apresenta quatro tipos de medidas de exceção a computar:

- *CellExp*(*c*): medida representando o grau de exceção da célula *c*, onde *c* denota uma célula em algum nível hierárquico *l*. Células com valores significantemente superiores ao esperado, mostrarão valores positivos de *CellExp*(*c*). Células com valores

significantemente abaixo do esperado, mostrarão valores negativos de *CellExp*(*c*). A cor de fundo da célula, bem como a sua intensidade, indicará o nível de exceção encontrado. A equação para cálculo de *CellExp*(*c*) pode ser vista em (3.9).

$$CellExp(i) = \begin{cases} sr(i) - \tau & , \quad sr(i) > \tau \\ sr(i) + \tau & , \quad sr(i) < -\tau \\ 0 & , \quad |sr(i)| \le \tau \end{cases} \quad (3.9)$$

- *UnderExp*(*c*): medida representando o grau de exceção que poderá ser encontrado abaixo da célula *c* se realizarmos uma operação de *drill-down* em alguma dimensão. Esta medida guia usuários a eficientemente encontrarem exceções escondidas nos níveis mais baixos. A cor da borda da célula e sua intensidade indicam o nível de exceção encontrado. *UnderExp*(*c*) é definida como

$$UnderExp(c) = \sum_{c' \in desc(c)} |CellExp(c')| \quad \text{ou}$$

$$UnderExp(c) = CellExp(x), \quad |CellExp(x)| = \max_{c' \in desc(c)} |CellExp(c')|$$

  de acordo com a preferência do usuário. A primeira fórmula toma a soma de todos as exceções descendentes, a segunda toma a exceção descendente mais anômala.
- *DimExp*(*l*, *k*): medida representando o grau de exceção que poderá ser encontrado se realizarmos uma operação *drill-down* ao longo da dimensão *k*, onde *l* representa o nível corrente da hierarquia, e *k* é o número da dimensão. O nome de cada dimensão em um painel é marcado com uma cor de fundo de intensidade variada indicando o grau de exceção encontrado em cada uma das dimensões.
- *DimExpForCell*(*c*, *k*): medida representando o grau de exceção que poderá ser encontrado abaixo da célula *c* quando realizarmos uma operação de *drill-down* ao longo da dimensão *k*. *c* define uma célula e *k* é o número da dimensão. Quando o usuário selecionar uma determinada célula, o nome de cada dimensão em um painel é marcado com uma cor de fundo de intensidade variada que indica o grau de exceção que poderá ser encontrado abaixo da célula *c* se realizarmos uma operação *drill-down* em cada uma das dimensões.

3.1.3 Discussão

Os modelos vistos não são os únicos utilizados para detecção de exceções em células de cubos OLAP. Knorr e Ng (KNORR; NG, 1997,1998) desenvolveram uma abordagem utilizando métodos não estatísticos, onde são apresentados algoritmos para detecção de exceções em grandes volumes de dados utilizando testes de discordância específicos. A sua justificativa é de que a complexidade da aplicação de métodos estatísticos prejudica o desempenho dos algoritmos de mineração.

A nossa escolha pela aplicação dos modelos desenvolvidos por Sarawagi et al. e Chen se deu pelo fato de terem sido desenvolvidos especificamente para detectar exceções em células de cubos OLAP, além de serem computacionalmente viáveis. Acreditamos que a integração dos dois modelos fornece uma base teórica sólida e confiável para o desenvolvimento de nossa ferramenta.

É importante salientar também a não existência de implementação disponível para os modelos. Se existem, estão embutidos em sistemas de KDD proprietários, monolíticos e fechados e que não podem ser expandidos, não podendo, portanto, sua funcionalidade ser utilizada fora destes ambientes.

No Capítulo 5, mostraremos como a nossa ferramenta se enquadra nos modelos vistos, e quais as soluções adotadas para os problemas de implementação encontrados.

3.2 Modelos para Acesso a Dados em Servidores OLAP

Vários foram os esforços para criação de um modelo padrão que forneça métodos de acesso a dados multidimensionais armazenados em um servidor OLAP: o OLAP Council (1997), conselho de produtores de software liderado pela Oracle, propôs um padrão de API chamado de MDAPI; a Microsoft propôs a API *OLE DB for OLAP* (Microsoft, 1998) com o apoio de mais 35 empresas de desenvolvedores de software; e algumas propostas acadêmicas como a MDS (*MultiDimensional System*) (CABIBBO; TORLONE , 1999) e a nD-SQL (GINGRAS; LAKSHMANAN, 1998) foram desenvolvidas com a finalidade de interoperabilidade de sistemas OLAP. Durante a realização deste trabalho, surgiu uma proposta do *Object Management Group* (OMG), chamada *Common Warehouse Metamodel* (OMG, 2001), que conta com o apoio de empresas como IBM, Unisys e Oracle.

Veremos ODCI - *Object Data Cube Interface* (FIDALGO, 2000; LINO, 2000) – um modelo abstrato orientado a objetos para consulta a *data warehouses* dimensionais, que foi desenvolvido baseado em duas APIs existentes para OLAP: *OLE DB for OLAP* e *MDAPI*. O modelo ODCI encapsula as funcionalidades necessárias para que uma aplicação cliente conecte-se a um servidor OLAP, carregue dados e metadados multidimensionais, e realize consultas OLAP. Com sua implementação, sistemas OLAP (não necessariamente conformes a *OLE DB for OLAP*), bem como outras tecnologias que necessitem de processamento analítico, poderão beneficiar-se do poder da linguagem *MDX* de *OLE DB for OLAP*, sem ter que dar suporte à plataforma COM da Microsoft.

Motivado pelo projeto MATRIKS, que visa o desenvolvimento de um ambiente abrangente e aberto de KDD para disponibilização e integração de serviços OLAP, ODCI tem a função de traduzir objetos OLAP proprietários em objetos OLAP padrão, permitindo o seu uso por ferramentas de mineração de dados e a realização de consultas a bases de dados multidimensionais.

ODCI (e sua implementação JDCI) foi desenvolvido no sentido de solucionar o problema de portabilidade do modelo de programação *ActiveX Data Object MultiDimensional* (ADO MD) (MICROSOFT, 1998) com Visual J++[5], já que o mesmo não usufrui da portabilidade e interoperabilidade de Java padrão por utilizar tipos que apenas a máquina virtual Java dentro da plataforma Windows dá suporte.

A fim de compreendermos como a interface ODCI foi desenvolvida, mostraremos duas APIs existentes para OLAP, nas quais a ODCI foi baseada, e que surgiram da tentativa de se tornarem um padrão de interoperabilidade OLAP: *OLE DB for OLAP* desenvolvida pela Microsoft; e a MDAPI, proposta pelo OLAP Council.

Destacamos a importância de ODCI dentro de nosso trabalho já que OCCOM se utiliza de ODCI, através de sua implementação JODI (*Java OLAP Data Interface*), para acessar metadados do servidor OLAP, bem como para realização de consultas MDX.

3.2.1 MDAPI

MDAPI (*Multi-Dimensional Application Program Interface*) (OLAP Council, 1998) é uma proposta coordenada pela organização internacional *OLAP Council*, que é uma associação

[5] Plataforma proprietária da Microsoft para desenvolvimento em Java.

sem fins lucrativos de vendedores de banco de dados multidimensionais, e que foi fundada pelas empresas *Arbor Software* e *Oracle*, com a finalidade de fornecer padrões de interoperabilidade entre os desenvolvedores de ferramentas OLAP.

A MDAPI é uma API orientada a objetos, independente da representação física subjacente de dados, para banco de dados modelados dimensionalmente. Conceitos a nível lógico de OLAP, tais como cubos, dimensões, hierarquias, e consultas, são representados por classes e interfaces de programação orientada a objetos. O modelo da *API* foi projetado utilizando-se UML, padrão para modelagem de objetos, e concretizada na especificação de duas interfaces de linguagem de programação (Java e COM) para auxiliar a tarefa de desenvolvedores de serviços de consultas *OLAP*.

A principal classe dessa *API* é a *Session* (ver Figura 3.12), pois é a partir dela que se pode explorar todas as funcionalidades da MDAPI. É de responsabilidade do *OLAP Council* fornecer a implementação da classe *Session*, enquanto que seus consorciados, devem implementar as outras interfaces da MDAPI, onde cada interface implementa as funcionalidades da MDAPI para um servidor OLAP particular. É importante ressaltar que um dos principais propósitos da MDAPI é prover uma noção uniforme dos metadados (Figura 3.13) de um esquema multidimensional, sendo indiferente ao servidor OLAP que os mantém.

Toda consulta *MDAPI* retorna dados em um objeto da classe cubo, e a aplicação recupera esses dados em instâncias da classe célula a partir deste cubo (ver Apêndice B para o entendimento do modelo global de classes da *MDAPI*). Um cubo pode ser n dimensional (hipercubo). Consultas na *MDAPI* são bem diferentes de consultas *OLE DB for OLAP*, pois aqui consultas e resultados são na forma de objetos. Resultados de uma consulta a um objeto cubo, é um outro objeto cubo que é um subcubo ou supercubo do anterior. Operações *OLAP* são implementadas na forma de métodos de classe.

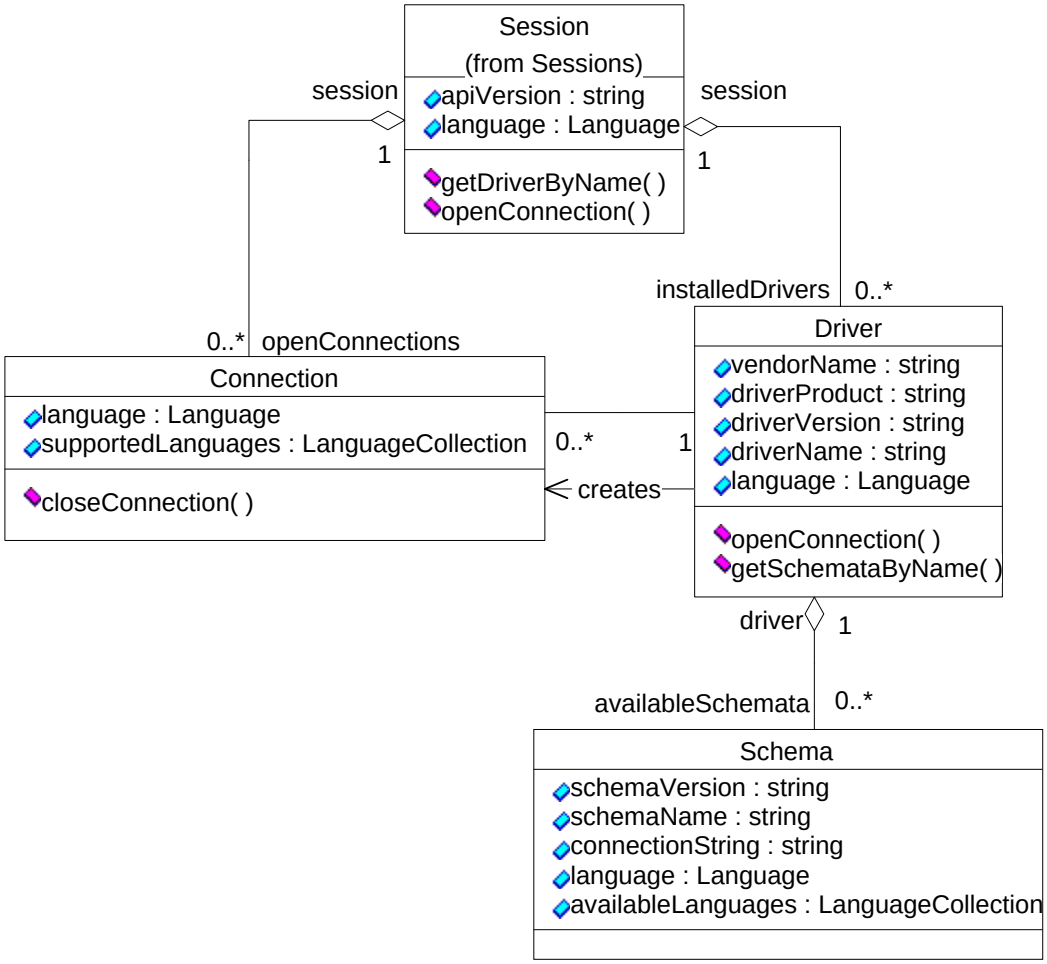


Figura 3.12 – Classes de conexão – MDAPI (OLAP Council 1998).

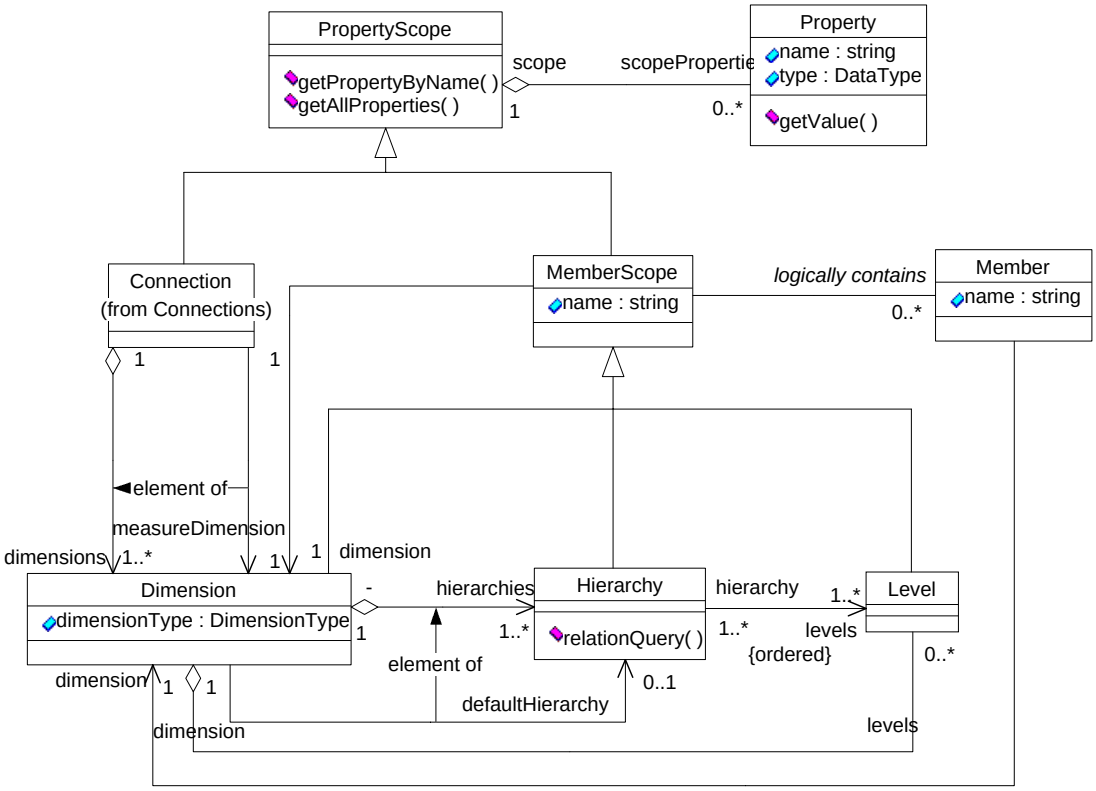


Figura 3.13 – Metadados MDAPI (OLAP Council, 1998).

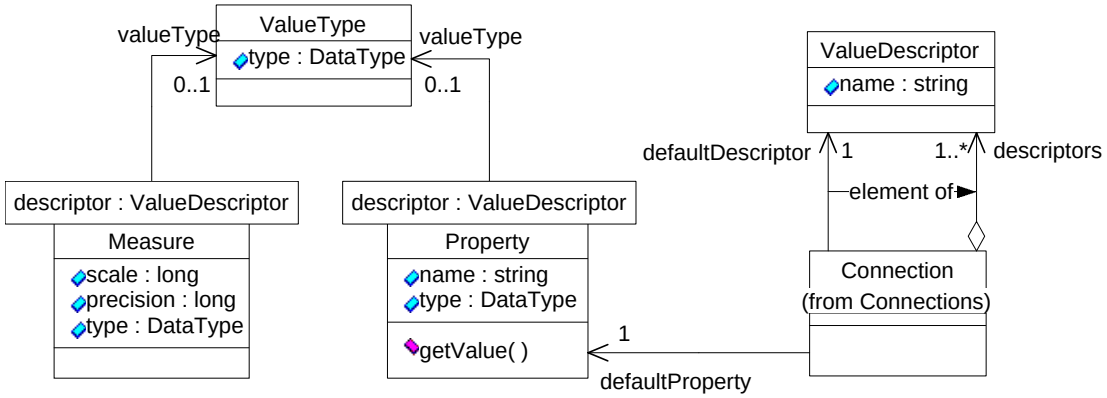


Figura 3.14 – Medidas e Propriedades de MDAPI (OLAP Council, 1998).

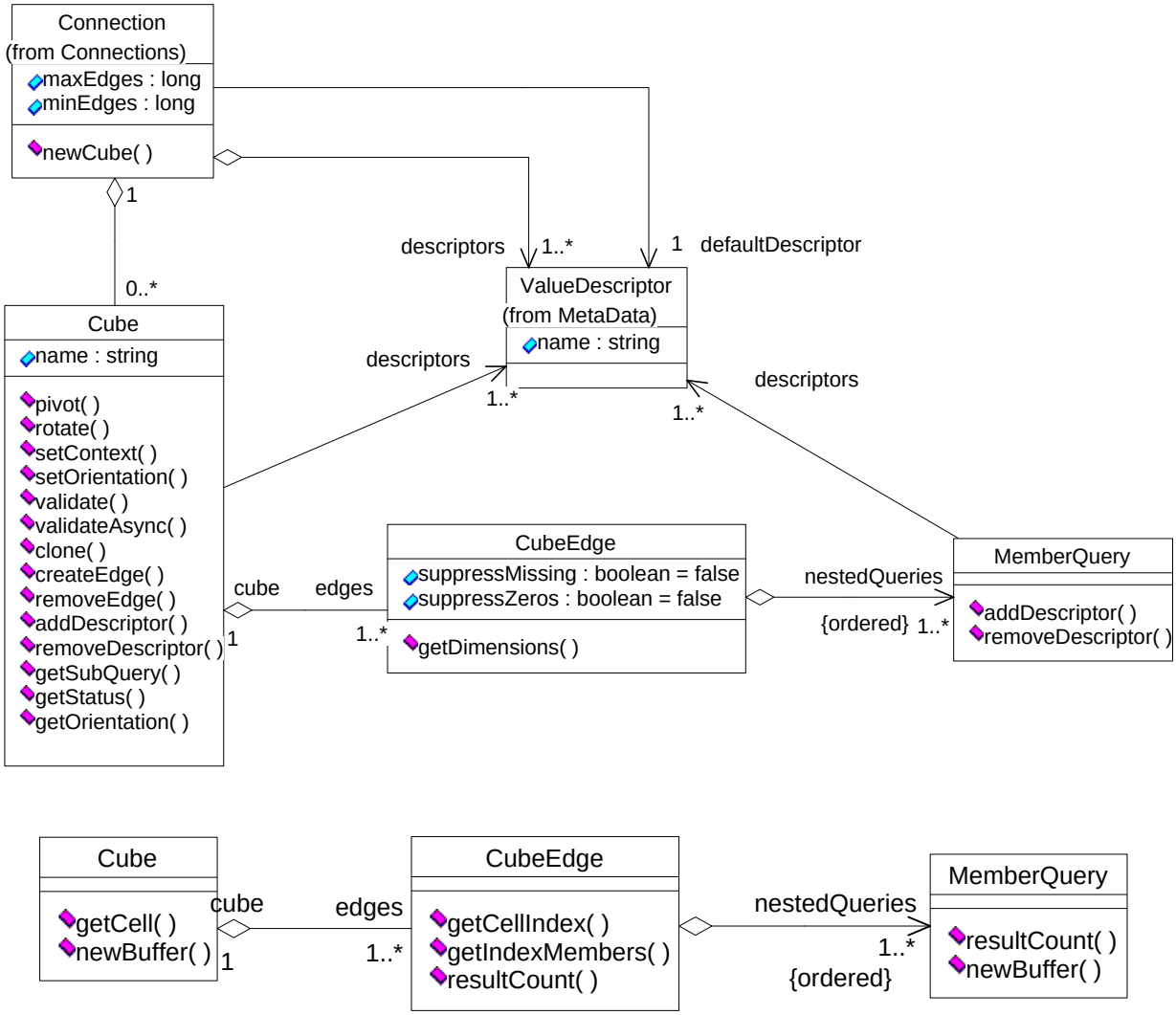


Figura 3.15 – Classes de consulta MDAPI (OLAP Council, 1998).

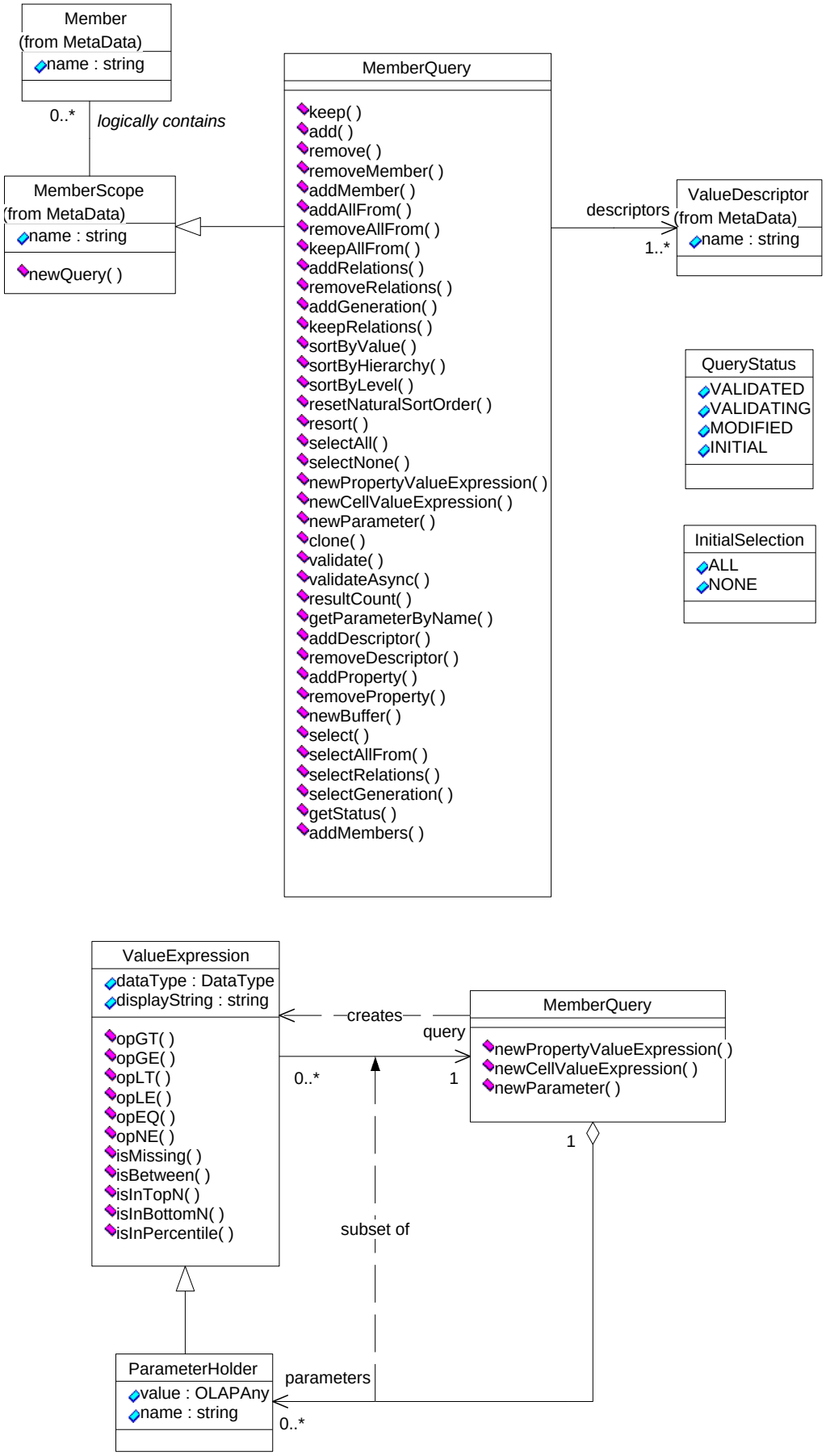


Figura 3.16 – Classes de consulta MDAPI (cont.) (OLAP Council, 1998).

Uma aplicação com a *MDAPI* envolve as seguintes ações:

(1) Obter um objeto da classe *connection* em um esquema multidimensional;

(2) Realizar uma consulta inicial a metadados, isto é, especificar os metadados que se deseja consultar (cubo, dimensões, medidas, hierarquia);

(3) Executar uma consulta OLAP inicial;

(4) Executar novas consultas OLAP a partir da última consulta realizada (Figuras 3.15 e 3.16): de acordo com requisições feitas pelo usuário, é possível navegar pelos dados através de operações para adicionar e remover membros, rotacionar o cubo, realizar sub-consultas, modificar ordenação dos valores, dentre outras.

A seguir analisaremos a *MDAPI* enumerando os aspectos positivos e negativos da mesma.

Pontos Positivos

Podemos enumerar os seguintes aspectos positivos da MDAPI:

- Faz uso de uma linguagem de modelagem de objeto padrão, como UML, que facilita a compreensão e especificação da API;
- Fornece transparência para o usuário quanto à estrutura nativa das diferentes fontes de dados multidimensionais;
- As consultas são representadas como um conjunto de objetos, possuindo todas as vantagens das características da orientação a objetos como modularidade, reusabilidade e extensibilidade;
- É uma especificação pública e multiplataforma para interoperabilidade entre sistemas OLAP;
- Facilita o trabalho do usuário no gerenciamento necessário de metadados para a formulação de consultas OLAP, isto é, o usuário especifica subcubos (*data marts*) em um *data warehouse*, e o servidor de dados é quem tem a responsabilidade de gerenciar os dados e metadados dos subcubos extraídos da base multidimensional.

Pontos Negativos

Como aspectos negativos da MDAPI podemos enumerar:

- A última versão divulgada da MDAPI (Versão 2) consiste em uma API apenas de leitura. Não existe nenhuma maneira de criar ou modificar dados em uma fonte de dados multidimensional, modificar a estrutura de um esquema multidimensional, nem realizar funções administrativas (por exemplo, concessão de autorização para acesso aos dados e metadados);

- É inerentemente baseada em um formalismo orientado a objetos, construído sobre uma base imperativa, o que impede a declaratividade das consultas. Uma sintaxe puramente orientada a objetos é inadequada para a comunidade de Banco de Dados, pois esta já está acostumada com sintaxes inspiradas em SQL, e não seria fácil adotar um padrão de consultas totalmente diferente;
- O aspecto prático de que, apesar da associação *OLAP Council* ser composta por fortes companhias como a Oracle e IBM, a API não foi implementada por nenhuma delas. Durante o demorado processo de publicação da MDAPI, a Microsoft lançou no mercado o servidor *OLAP MS OLAP Server*, usando o padrão alternativo da API *OLE DB for OLAP*, o que levou quase todos os participantes da MDAPI a também dar suporte a esse padrão.

Depois de ter perdido força mercadológica, o OLAP Council renasceu como a organização *ASF* (*Analytical Solutions Fórum* <http://www.tasf.org/>), cujo principal propósito é apenas policiar o mercado de suporte à decisão, e desde então, não se tem mais informações sobre implementações ou novas versões da MDAPI.

3.2.2 OLE DB for OLAP

OLE DB for OLAP (*Object Linking and Embedding DataBase for OLAP*) (Microsoft, 2002) é um dos componentes da arquitetura da *Microsoft* para acesso universal a dados chamada de UDA (*Universal Data Access*) (Microsoft, 2001; BLAKELEY; PIZZO, 1998). Esta arquitetura serve como plataforma para desenvolvimento de aplicações multi-camadas, que necessitam de acesso a fontes de dados heterogêneas, e consiste em uma coleção de componentes de software que interagem com várias fontes de dados através de um conjunto comum de interfaces definidas pela *API OLE DB* (Microsoft, 1998).

A arquitetura *UDA*, que é ilustrada na Figura 3.17, é dividida em três módulos, de acordo com a sua função:

- Fornecedores: armazenam e fornecem dados;
- Serviços: processam e transformam os dados ou fornecem serviços comuns, e;
- Consumidores de dados (*Consumers*): fazem uso dos dados.

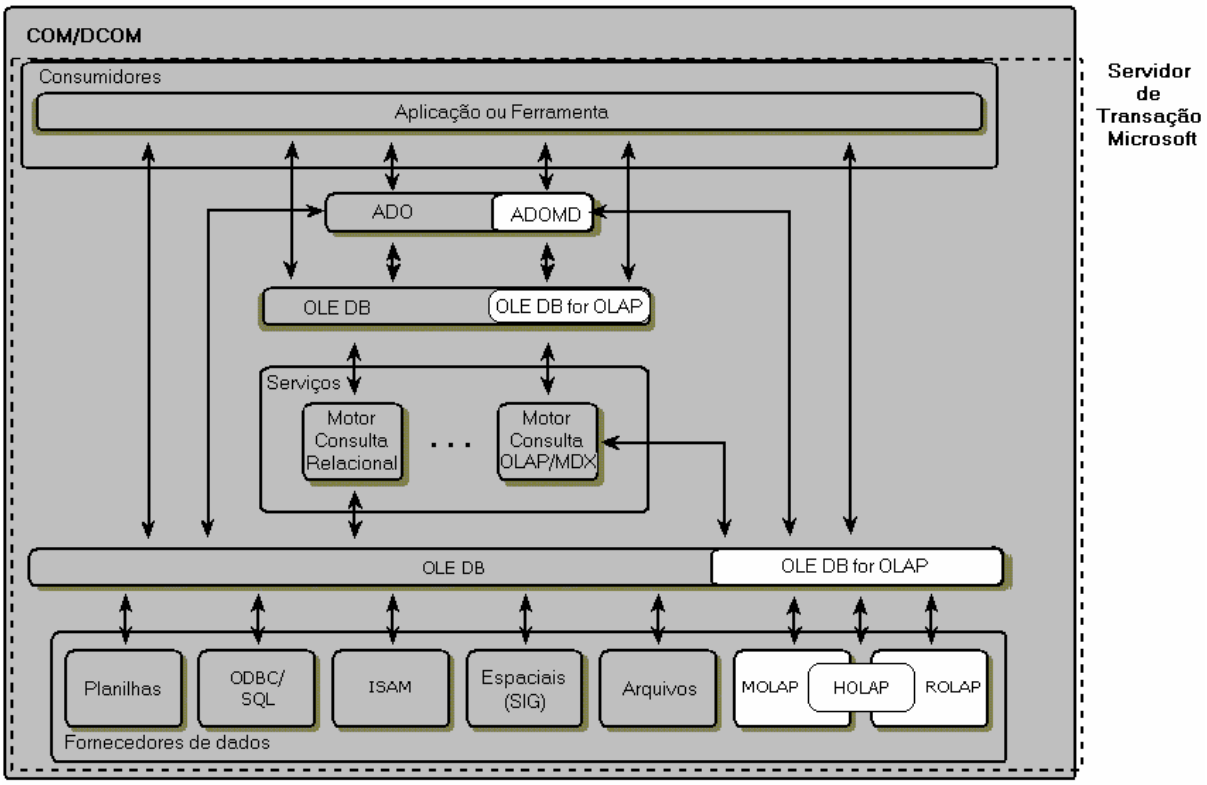


Figura 3.17 – Arquitetura UDA da Microsoft (Microsoft, 2001).

A comunicação entre os componentes da arquitetura UDA (setas bidirecionais), ocorrem através de:

(1) Troca de objetos *COM* (*Component Object Model*) (Microsoft, 1995) – uma arquitetura de software que permite que aplicações sejam construídas a partir de componentes de software binários. *COM* é uma arquitetura subjacente que compõe a base para serviços de software de mais alto nível, como os providos por *OLE*;

(2) Troca de objetos *DCOM* (*Distributed Componente Object Model*) (Microsoft, 2000) –uma extensão de *COM* que dá suporte a objetos distribuídos em uma rede. Trata-se de um protocolo que permite que componentes de software se comuniquem diretamente em uma rede, de maneira confiável, segura e eficiente; ou

(3) Protocolos, como por exemplo, *HTTP* (IETF, 2000).

Além dos protocolos citados acima, empregados na arquitetura UDA, outros protocolos alternativos realizam a tarefa de comunicação entre componentes:

- CORBA (*Common Object Request Broker Architecture*) – arquitetura e infraestrutura independentes de plataforma, utilizada para comunicação entre aplicações através do protocolo padrão IIOP (OMG, 1998). Um programa baseado em CORBA, desenvolvido em praticamente qualquer máquina, sistema operacional, linguagem de programação, e rede, poderá interoperar com outro programa baseado em CORBA, do mesmo ou de outro

desenvolvedor, não importando a máquina, sistema operacional, linguagem de programação ou rede (OMG, 2002).

- SUN Enterprise JavaBeans – arquitetura padrão de componentes para computação distribuída, orientada a objetos e baseada em transações, para aplicações desenvolvidas na linguagem de programação Java. A arquitetura *Enterprise JavaBeans* fornece interoperabilidade entre *enterprise beans* e componentes desenvolvidos na plataforma *Java 2 Enterprise Edition* (J2EE) bem como aplicações em outras linguagens de programação, além de ser compatível com o protocolo CORBA (SUN, 2002).
- Ambiente de desenvolvimento Microsoft .Net – componente Windows que dá suporte a construção e execução de aplicações e serviços XML Web (W3C, 2000). Tem como objetivos fornecer um ambiente de programação orientado a objetos consistente, onde o código do objeto é armazenado e executado localmente, executado localmente mas distribuído na Internet ou executado remotamente; procura minimizar problemas de distribuição e controle de versão, segurança e performance (MICROSOFT, 2001).

Na arquitetura UDA, *OLE DB for OLAP* defini-se como uma extensão de *OLE DB* para acesso a fontes de dados multidimensionais ROLAP, MOLAP ou HOLAP e ambas definem um conjunto de interfaces COM que permitem às aplicações um acesso uniforme aos dados armazenados em diversas fontes, independentemente de localização e de tipo.

A *OLE DB for OLAP*[6] é a extensão de *OLE DB* (também na forma de um conjunto de objetos e interfaces *COM*) usada para o desenvolvimento e acesso de provedores de dados multidimensionais (*Multidimensional Data Providers – MDPs)*.

O acesso direto a *OLE DB* e *OLE DB for OLAP* só pode ser feito através da linguagem de programação Visual C++. Entretanto a Microsoft desenvolveu outras duas APIs: (1) *ADO* (*ActiveX Data Object*) (Microsoft, 1998, 2000) e, (2) *ADO MD* (*ActiveX Data Object MultiDimensional*) (Microsoft, 1998), que permitem manipular, respectivamente, as funcionalidades de *OLE DB* e *OLE DB for OLAP* por linguagens de programação como Visual Basic, Visual J++ e também Visual C++. As APIs ADO e ADO MD representam uma ponte entre uma aplicação desenvolvida em alguma destas linguagens e os dados fornecidos pelas APIs *OLE DB* e *OLE DB for OLAP*.

Conceitualmente, dados multidimensionais são representados como um cubo. O cubo – ou mais apropriadamente, o hipercubo – é o objeto central dos metadados reconhecidos por *OLE DB for OLAP*. Expresso através de um objeto de conjunto de dados (*dataset*), o hipercubo é

[6] *OLE DB for OLAP* faz parte da versão 2.0 da *OLE DB.*

específico de *OLE DB for OLAP*. Consumidores podem disponibilizar os dados como um conjunto de linhas (*rowset*), pela conexão a um provedor de dados tabular (TDP – *Tabular Data Provider*); ou como um conjunto de dados (*dataset*) pela conexão a um provedor de dados multidimensional (MDP – *Multidimensional Data Provider*), como mostrado na Figura 3.18 (Microsoft, 2002).

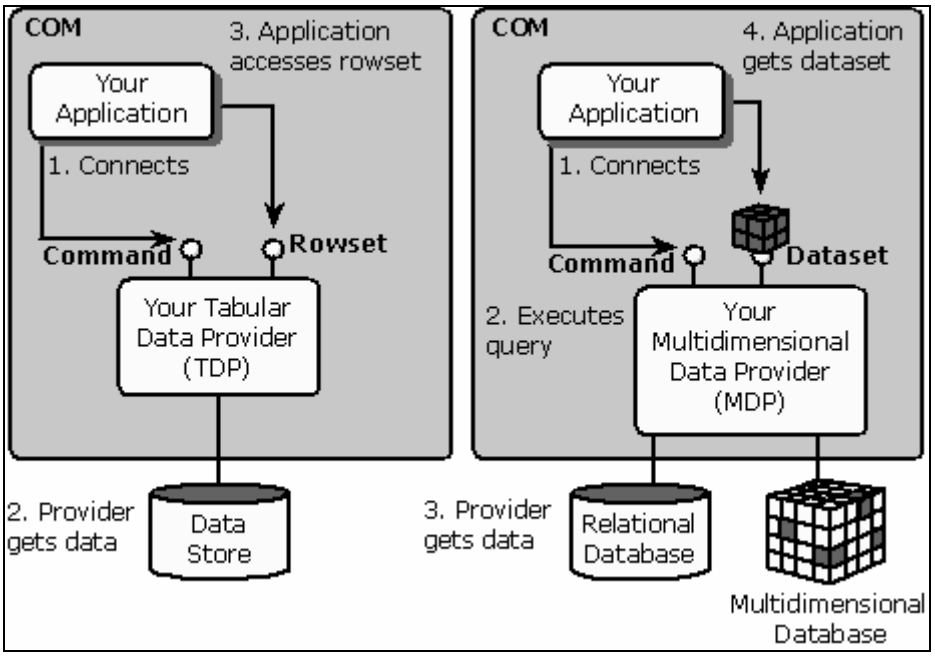


Figura 3.18 – Conexão a provedores *OLE DB* e *OLE DB for OLAP* (Microsoft, 2002).

A API ADO MD, que permite o acesso a dados multidimensionais através das linguagens Visual Basic, Visual C++ e Visual J++, é uma extensão de ADO para incluir objetos específicos de dados multidimensionais como *CubeDef* (cubo de dados) e *Cellset* (conjunto de células). ADO MD permite a navegação através do esquema multidimensional, consultar um cubo e recuperar os resultados de uma consulta MDX. A estrutura de ADO MD é mostrada na Figura 3.19.

Observe que a notação utilizada para representar ADO MD consiste apenas em uma hierarquia de classes, sem definição de atributos e métodos para composição de seu modelo orientado a objetos e é a única representação do modelo de classes dessa API.

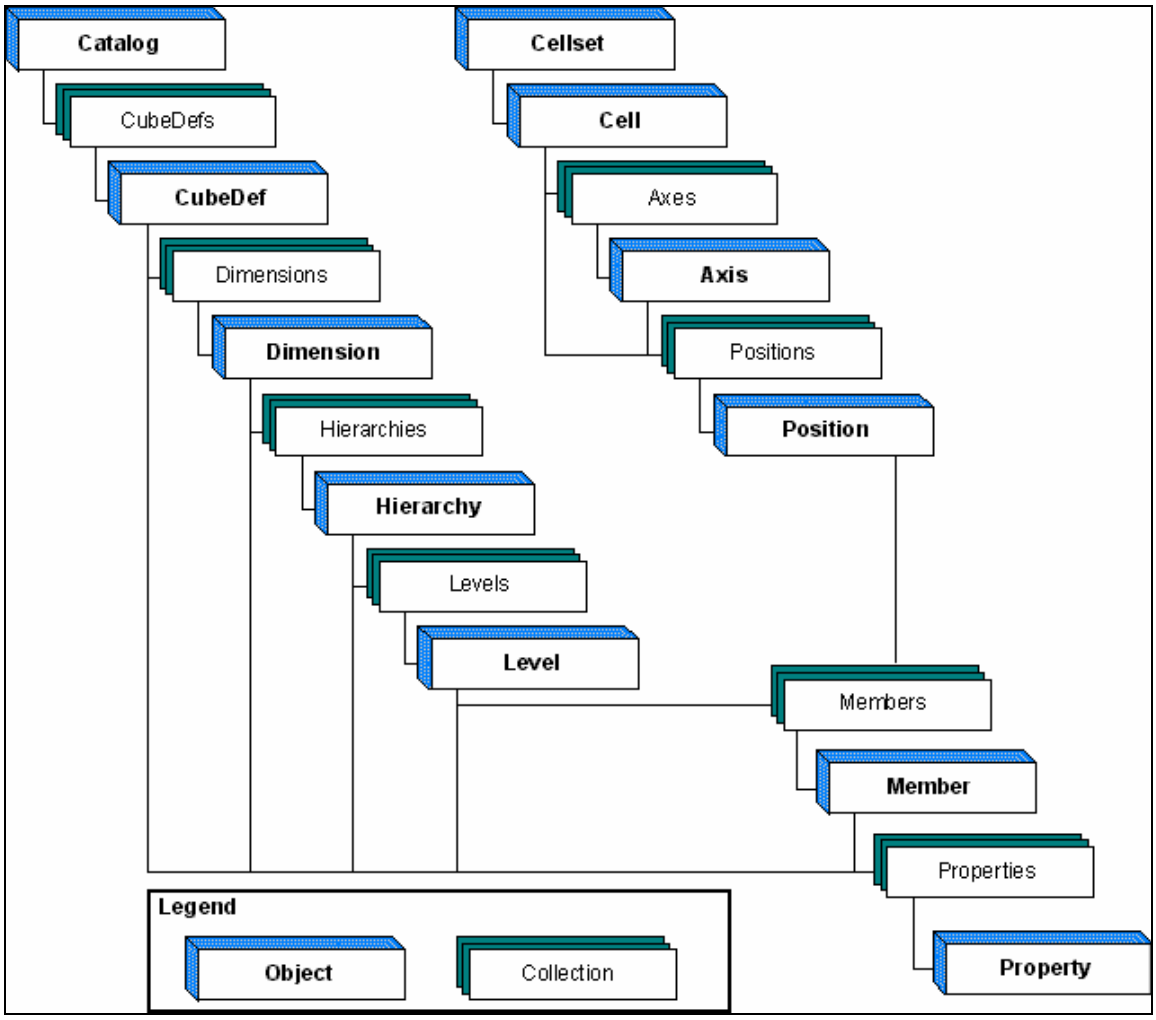


Figura 3.19 – Estrutura de ADO MD (Microsoft, 2002).

A seguir mostraremos uma explicação resumida sobre cada objeto do esquema multidimensional de ADO MD:

- *Catalog Object*: contém informações do esquema multidimensional (cubos e suas dimensões, hierarquias, níveis e membros) específicos de um provedor de dados multidimensionais (MDP). Através de coleções e objetos *Catalog*, podemos abrir um catálogo pela especificação de uma conexão válida; identificar o catálogo através da propriedade nome; ou iteragir através dos cubos disponíveis no catálogo através da coleção de cubos (*CubeDefs collection*).
- *CubeDefs Collection*: coleção de objetos do tipo cubo (*CubeDef*).
- *CubeDef Object*: objeto central dos metadados ADO MD, o cubo consiste de uma estrutura que é formada por um conjunto de dimensões, hierarquias, níveis e membros.
- *Dimensions Collection*: coleção de objetos do tipo dimensão (*Dimension*).
- *Dimension Object*: categoria independente de dados de um banco de dados multidimensional, derivado de entidades do negócio. Uma dimensão tipicamente contém

itens utilizados como critério de consulta para as medidas do banco de dados. Ex.: Tempo, Localização, Produto.

- *Hierarchies Collection*: coleção de objetos do tipo hierarquia (*Hierarchy*).
- *Hierarchy Object*: uma hierarquia é um caminho de agregação em uma dimensão. Uma dimensão pode conter múltiplos níveis de granularidade que possuem relacionamento tipo pai-filho. Uma hierarquia define como estes níveis estão relacionados. Ex.: uma dimensão Tempo pode possuir as hierarquias Dia→Mês→Trimestre→Ano e Dia→Dia da Semana→Ano.
- *Levels Collection*: coleção de objetos do tipo nível (*Level*).
- *Level Object*: um nível é um passo de agregação em uma hierarquia. Para dimensões com múltiplas camadas de informação, cada camada é um nível. Ex.: Dia, Mês, Trimestre e Ano em uma dimensão Tempo.
- *Members Collection*: coleção de objetos do tipo membro (*Member*).
- *Member Object*: um membro é um item de dado em uma dimensão. Ex.: para o nível Mês de uma dimensão Tempo, teríamos os membros Jan, Fev, Mar, ..., Dez.
- *Properties Collection*: coleção de propriedades (*Property*) de um objeto.
- *Property Object*: uma propriedade é uma determinada informação sobre um objeto. Exemplos de propriedades mais comuns são nome, nome único (identificador do objeto), descrição, rótulo e tipo.
- *Cellset Collection*: representa o resultado de uma consulta multidimensional. É um conjunto de células selecionadas de cubos ou de outros conjuntos de células.
- *Cell Object*: representa o dado da interseção das coordenadas dos eixos, contido em um conjunto de células (C*ellset*). Possui informações como valor (formato numérico – 10000,00) e valor formatado (formato textual – R$ 10.000,00).
- *Axes Collection*: representa o conjunto de eixos de uma consulta multidimensional (coluna e linha). Toda consulta multidimensional deve possuir pelo menos um eixo (coluna).
- *Axis Object*: representa um eixo de posição ou de filtro de um conjunto de células, contém membros selecionados de uma ou mais dimensões.
- *Positions Collection*: conjunto de posições em que um eixo ou célula pode estar inserido.
- *Position Object*: representa um conjunto de um ou mais membros de diferentes dimensões que definem um ponto ao longo de um eixo. A posição é responsável pela ligação entre a consulta multidimensional e os metadados do cubo de dados.

Um dos principais problemas apresentados por ADO MD é o fato de que os seus atributos não são manipulados diretamente e sim através das interfaces *Properties* e *Property*. Por exemplo,

para acessar os atributos *tipo, data de criação* e *data de atualização* da interface *CubeDef*, deve-se chamar o método *getProperties*(), para retornar a coleção de atributos do objeto; chamar o método *getItem*(*arg0: Variant*), para retornar um atributo, dado o ordinal que o identifica nessa coleção (objeto do tipo *Property*); e chamar o método *getValue*(), que retorna o valor do atributo; ou seja, a instrução a seguir deveria ser executada: *objetoADOMD.getProperties().getItem(ordinal).getValue()*. Além disso, o usuário deve conhecer de antemão o ordinal que identifica a propriedade que deseja acessar.

Pontos Positivos

Os principais aspectos positivos da *API OLE DB for OLAP* e do *wrapper ADO MD* são:

- Expressam suas consultas multidimensionais, assim como a definição de metadados, por meio de uma linguagem (*MDX*) simples e com sintaxe inspirada na linguagem de consultas SQL, o que a torna de fácil aceitação e padronização pelos desenvolvedores de aplicações de Banco de Dados;
- Permite acesso uniforme e transparente a dados de servidores do tipo ROLAP, MOLAP ou HOLAP;
- Possui suporte da maioria dos desenvolvedores OLAP. Aproximadamente 44 companhias[7] se comprometeram a disponibilizar uma API para *OLE DB for OLAP*;
- A sua implementação como primeiro servidor OLAP para mercado de massa: o *MS OLAP Server,* que está incluso a partir da Versão 7.0 do *SQL Server* da Microsoft.

Pontos Negativos

As principais limitações da API *OLE DB for OLAP* e ADO MD, solucionadas através da API abstrata *ODCI* (FIDALGO, 2000; LINO, 2000), podem ser classificadas em três aspectos:

- Aspectos referentes aos princípios da Engenharia de Software de Orientação a Objetos;
- Aspectos intrínsecos ao seu modelo de objetos;
- Aspectos referentes à portabilidade da *API*.

A seguir são detalhados os pontos negativos de cada um desses itens separadamente.

[7] Entre elas: *Brio Technology, Business Objects, Cognos, Hyperion Software, Microstrategy Inc, Pilot Software, Sagent Technology, Inc., SAS Institute e Seagate Software.*

Limitações quanto aos Princípios da Engenharia de Software Orientada a Objetos

- Objetos de *OLE DB for OLAP* e *ADO MD* não obedecem ao princípio de encapsulamento da orientação a objetos, pois guardam e manipulam propriedades que são inerentes a outros objetos. Por exemplo, na Figura 3.20, os métodos *getLevelName()* e *getLevelDepth()* da interface *Member* deveriam estar apenas na interface *Level* já que manipulam atributos de *Level*. Em muitos casos inclusive, objetos das interfaces *OLE DB for OLAP* e ADO MD possuem atributos que não lhe dizem respeito, mas sim a um outro objeto/dado OLAP ao qual está associado. No exemplo da Figura 3.21 podemos notar que os atributos *CatalogName e CubeName* dizem respeito ao objeto Cubo e não à Dimensão;

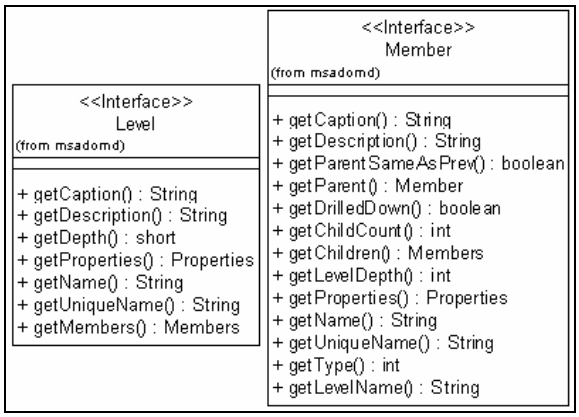


Figura 3.20 – Interfaces *Level* e *Member* de ADO MD (FIDALGO, 2000).

- *OLE DB for OLAP* e ADO MD não obedecem o princípio de facilidade de uso de qualidade de software, pois só permitem manipular seus objetos OLAP e alguns de seus atributos (propriedades) exclusivamente através de um ordinal. Por exemplo, para se ter acesso a uma dimensão de um cubo, deve-se chamar o método de cubo *cubo.obterDimensao(int numDimensao)*, passando-se como parâmetro o inteiro que representa a dimensão que se quer consultar;

Atributos	Descrições
CatalogName	Nome do esquema multidimensional que o cubo pertence
CubeName	Nome do cubo que a dimensão pertence
DimensionName	Nome da dimensão
DimensionUniqueName	Nome único da dimensão
DimensionCaption	Rótulo da dimensão
DimensionOrdinal	Ordinal que distingue a dimensão das demais existentes no cubo
DimensionType	Tipo da dimensão
DimensionCardinality	Quantidade de membros da dimensão
DefaultHierarchy	Hierarquia padrão da dimensão
Description	Descrição da dimensão

Figura 3.21 – Atributos de um objeto *Dimension* de *OLE DB for OLAP* e ADO MD (FIDALGO, 2000).

Limitações Intrínsecas ao Modelo de Objetos

- Ausência de um modelo abstrato de objetos. A Figura 3.19 extraída do manual de *ADO MD* é a única representação da hierarquia de classes destas APIs, e que não traz muita informação;
- Em *OLE DB for OLAP* e ADO MD não está definida nenhuma forma para atualizações de cubos do esquema multidimensional. São APIs apenas para consultas OLAP, não definindo nenhuma interface, classe ou cláusula MDX para atualizar ou criar, no servidor OLAP, metadados de um esquema multidimensional, ou seja, só é permitido ao usuário realizar consultas em cubos previamente criados pelo servidor *OLE DB for OLAP*;
- ADO MD tem a ausência de uma classe de conexão bem definida; a classe *Catalog* de ADO MD além do gerenciamento do esquema multidimensional, também é responsável por abrir uma conexão;
- *OLE DB for OLAP* e ADO MD não têm uma classe para gerenciar informações do servidor OLAP conectado. Esta informação só fica subtendida quando o servidor OLAP é especificado para abrir uma conexão;
- *OLE DB for OLAP* e ADO MD não possuem uma classe bem definida para epecificação de uma consulta MDX e outra para o resultado da mesma. A interface *IMDDataset* na API *OLE DB for OLAP* e a classe *Cellset* no ADO MD realizam ao mesmo tempo estas duas tarefas ortogonais;
- ADO MD cita explicitamente na sua documentação a diferença entre membros de um cubo e membros de um resultado MDX[8], mas não implementa estes objetos em classes ou interfaces distintas, ficando-se sujeito a erros caso sejam manipulados indevidamente, pois alguns atributos só são pertinentes aos membros de um cubo, enquanto que outros só são pertinentes a membros de um resultado MDX.

Limitações Referentes à Portabilidade da API

- *OLE DB for OLAP* e ADO MD são baseadas no padrão de interoperabilidade orientado a objetos COM, e por isso, só são executadas em plataforma Windows;
- A implementação com objetos COM não é realmente orientada a objetos, o que faz perder as potencialidades desse paradigma (LEWANDOWSKI, 1998);

[8] Semanticamente membros de um cubo e membros de um resultado *MDX* significam a mesma coisa. Entretanto a *OLE DB for OLAP* faz distinção entre um dado armazenado e um dado apresentado.

- Apesar de ADO MD possuir uma *API* para *Visual J++* (a versão proprietária da Microsoft para Java), a mesma não é Java padrão. A API Java de ADO MD possui código COM embutido no código Java, o que implica na não portabilidade da API. Na Figura 3.22 é apresentado um trecho de código da interface *Cellset*, interface para manipular consultas MDX da API *Visual J++* para ADO MD, que ilustra as diretivas COM embutidas no código em forma de comentário. Estas diretivas só são compreendidas pela máquina virtual Java da Microsoft. Também são apresentados os tipos COM proprietários da Microsoft (*Variant* e *SafeArray*) aos quais Java padrão não dá suporte.

Em resumo, *OLE DB for OLAP* é uma API orientada a objetos, fundamentada na arquitetura UDA da Microsoft, que permite acesso a consultas MDX a bases relacionais (ROLAP), multidimensionais (MOLAP) e híbridas (HOLAP). Esta, apesar de possuir limitações quanto a princípios de orientação a objetos, qualidade de software, sua modelagem e portabilidade, tem sido bem aceita pelos desenvolvedores de sistemas OLAP e encaminha-se para ser um padrão de fato.

```
// WARNING: Do not remove the comments that include "@com" directives.
// This source file must be compiled by a @com-aware compiler.
// If you are using the Microsoft Visual J++ compiler, you must use
// version 1.02.3920 or later. Previous versions will not issue an error
// but will not generate COM-enabled class files.

package msadomd;

import com.ms.com.*;
import com.ms.com.IUnknown;
import com.ms.com.Variant;
import msado15.*;

/** @com.class(classid=228136B8-8BD3-11D0-B4EF-00A0C9138CA4,DynamicCasts)
    @com.interface(iid=2281372A-8BD3-11D0-B4EF-00A0C9138CA4, thread=AUTO, type=DUAL) */
public class Cellset implements IUnknown,com.ms.com.NoAutoScripting,msadomd.ICellset
{
  /** @com.method(vtoffset=4, dispid=0, type=PROPGET, name="Item", name2="getItem", addFlagsVtable=4)
      @com.parameters([in,vt=16396,type=SAFEARRAY] idx, [iid=2281372E-8BD3-11D0-B4EF-
00A0C9138CA4,thread=AUTO,
         type=DISPATCH] return) */
  public native msadomd.Cell getItem(com.ms.com.SafeArray idx);

  /** @com.method(vtoffset=5, dispid=1610743809, type=METHOD, name="Open", addFlagsVtable=4)
      @com.parameters([in,type=VARIANT] DataSource, [in,type=VARIANT] ActiveConnection) */
  public native void Open(Variant DataSource, Variant ActiveConnection);
  …
}
```

Figura 3.22 – Exemplo de código Java/COM da interface *Cellset* (FIDALGO, 2000).

3.2.3 CWM

CWM (*Common Warehouse Metamodel*) (OMG, 2001) é uma especificação que descreve a troca de informações entre ferramentas, plataformas e repositórios de metadados de *data warehouses*, em ambientes abertos e heterogêneos. CWM é baseado em três padrões da indústria de software:

- UML (*Unified Modeling Language*) – padrão OMG para modelagem de aplicações (OMG, 2002);
- MOF (*Meta Object Facility*) – tecnologia OMG para definição e representação de metadados como objetos CORBA. Dá suporte a qualquer tipo de metadados que pode ser descrito utilizando técnicas de modelagem de objetos. Podem descrever qualquer aspecto de um sistema e a informação que ele contém, em qualquer nível de detalhe (CWM, 2001);
- XMI (*XML Metadata Interchange*) – padrão OMG para troca de metadados, permite que um modelo seja transmitido de uma forma serializada. O seu foco é a troca de metadados MOF. Pode ser vista como um formato para troca de metadados independente da tecnologia de *middleware* (CWM, 2001).

O padrão UML define uma rica linguagem para modelagem orientada a objetos, que é utilizada por várias ferramentas gráficas de projeto. O padrão MOF define um extenso ambiente para construção de modelos para metadados, e fornece ferramentas com interfaces de programação para armazenar e acessar metadados em um repositório. E o padrão XMI permite que metadados sejam trocados como arquivos serializados em um formato padrão baseado em XML. A arquitetura completa oferece uma larga faixa para escolhas de implementação para desenvolvedores de ferramentas, repositórios e objetos. XMI em particular, diminui a barreira para aceitação do uso do padrão de metadados OMG (OMG, 2001).

Dentre as especificações de CWM, destacam-se as seguintes aplicações:

- Construção de *data warehouses*, incluindo passos de integração, seleção e transformação de dados;
- Dados multidimensionais e OLAP;
- Mineração de dados.

CWM tem como objetivo solucionar um dos problemas mais críticos na construção de *data warehouses* e inteligência de negócios atualmente – a troca e o gerenciamento de metadados.

Até o término deste trabalho não tínhamos conhecimento de nenhuma aplicação implementada utilizando o padrão CWM.

3.2.4 ODCI

ODCI (*Object Data Cube Interface*) é um modelo abstrato e orientado a objetos para disponibilização e integração de serviços OLAP. Por estar acima da arquitetura UDA, ODCI usufrui da abstração fornecida por esta arquitetura quanto ao conhecimento das estruturas multidimensionais nativas dos servidores OLAP, e mapea a abstração destas estruturas nativas – estruturas OLAP – no seu modelo de objetos (FIDALGO, 2000).

O gerenciamento dessas estruturas multidimensionais, bem como validação e execução das consultas MDX, é de responsabilidade única do servidor *OLE DB for OLAP*, sendo a implementação de ODCI apenas uma ponte para consultá-las.

Com sua implementação, sistemas OLAP (não necessariamente conformes a *OLE DB for OLAP*), bem como outras tecnologias que necessitem de processos analíticos, poderão beneficiar-se do poder da linguagem MDX de *OLE DB for OLAP*, sem ter que dar suporte a plataforma COM da Microsoft.

A implementação de ODCI deve manter a compatibilidade com objetos Java padrão a fim de manter sua portabilidade, não devendo exportar objetos proprietários da plataforma COM. ODCI baseou-se em conceitos padrões de OLAP, mantendo-se a compatibilidade com a proposta de *OLE DB for OLAP* e superando as suas limitações. No entanto, a limitação de não permitir a criação ou atualização de metadados de um esquema multidimensional ainda persiste.

Na Figura 3.23 podermos ver o Modelo de Dados de ODCI.

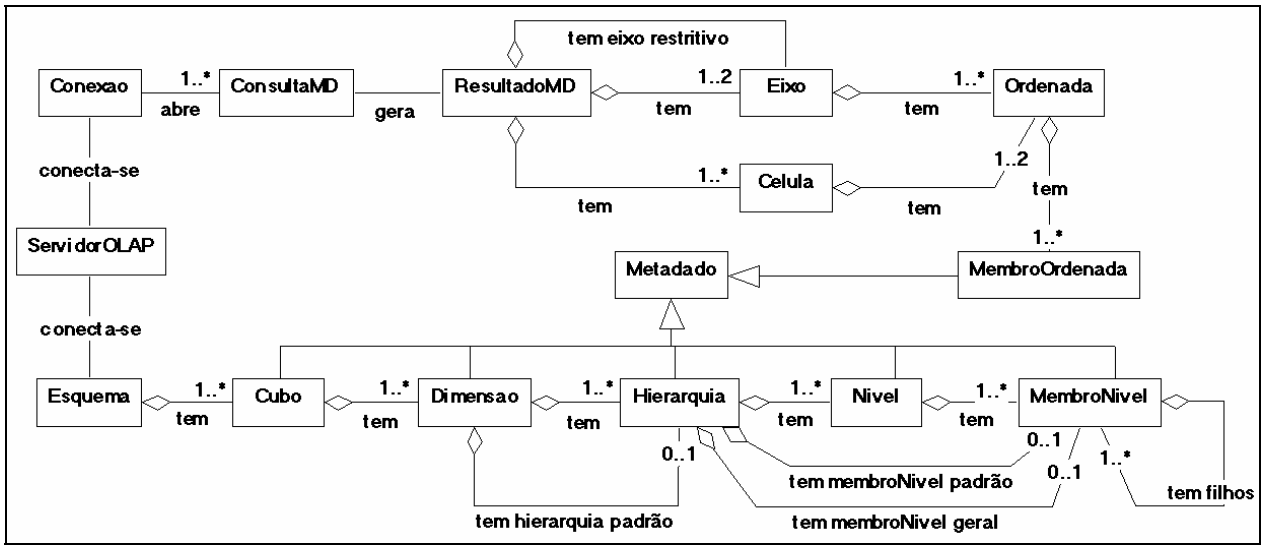


Figura 3.23 – Modelo de Dados de ODCI (LINO, 2000)

Uma aplicação com a *ODCI* envolve as seguintes ações (LINO, 2000):

(1) Conectar-se a um esquema multidimensional de um servidor OLAP – geralmente, um esquema multidimensional representa um *data warehouse*, e os cubos desse esquema os *data mart*s desse *data warehouse*;

(2) Consultar um esquema multidimensional, que é composto pelos seguintes objetos: cubo, dimensão, hierarquia, nível e membro;

(3) Definir consultas multidimensionais baseadas na linguagem MDX do padrão *OLE DB for OLAP*, que recuperam subcubos de dados; e

(4) Consultar a estrutura dos objetos que formam o resultado de uma consulta multidimensional (MDX), onde os objetos que compõem um resultado multidimensional são: eixo, célula, ordenada e membro.

De acordo com essas ações podemos dividir o modelo de dados da ODCI em quatro partes distintas: (1) módulo de conexão, onde pode ser realizada a ação 1, (2) módulo de metadados, onde são representados os metadados e pode-se realizar a ação 2, (3) módulo de consulta, onde são realizadas as consultas multidimensionais, que envolve as ações 3 e (4) módulo de resultados multidimensionais, que dá suporte a ação 4.

Para permitir que as ações que envolvem a utilização da ODCI sejam transparentemente realizadas por qualquer sistema OLAP, esta API por meio da implementação Java JDCI (FIDALGO, 2000) de suas classes *Conexao* e *ConsultaMD*, se encarregará de realizar automaticamente toda a operação de reescrita dos objetos COM de ADO MD, no momento em que uma conexão for aberta ou uma consulta MDX for executada. Cabe a estas classes instanciarem os restantes dos objetos de ODCI, que serão reescritos e manipulados apenas

pelas suas respectivas interfaces. Na Figura 3.24 podemos visualizar o modelo de classes de ODCI com as suas respectivas interfaces.

As limitações de ADO MD/*OLE DB for OLAP* que a ODCI corrigiu são: (1) ausência de encapsulamento das propriedades dos objetos; (2) quebra do princípio de facilidade de uso de qualidade de software; (3) modelo confuso e com pouca semântica; (4) ausência de uma classe conexão bem definida; (5) ausência de uma classe para gerenciar informações do servidor OLAP conectado; (6) ausência de uma classe bem definida para a consulta e outra para o resultado; (7) não distinção no modelo entre membros de um cubo e membros de um resultado multidimensional; e (8) modelo que não é totalmente orientado a objetos.

3.2.5 JDCI

JDCI (*Java Data Cube Interface*) (FIDALGO, 2000) é uma interface de programação (API) que implementa o modelo ODCI visto na seção anterior. Esta surgiu com a necessidade de disponibilizar e integrar serviços OLAP providos pelos servidores *OLE DB for OLAP*, com outras tecnologias de suporte à decisão e KDD, por exemplo, sistemas especialistas e bancos de dados dedutivos. Foi baseado em conceitos padrões de OLAP e inspirado nos pontos positivos do modelo de programação *Visual J++* com ADO MD, de forma que JDCI, ao implementar o modelo ODCI, reescreve objetos Java/COM em objetos 100% Java padrão, permitindo a portabilidade da linguagem Java; e mantém a compatibilidade com a proposta de *OLE DB for OLAP*, corrigindo grande parte de suas limitações (FIDALGO, 2000).

JDCI é implementado em uma arquitetura cliente/servidor, de maneira a manter o cliente independente de plataforma, já que o servidor, desenvolvido na linguagem *Visual J++*, deve dar suporte a reescrita dos objetos Java/COM em objetos Java padrão, rodando em equipamentos com o sistema operacional Windows.

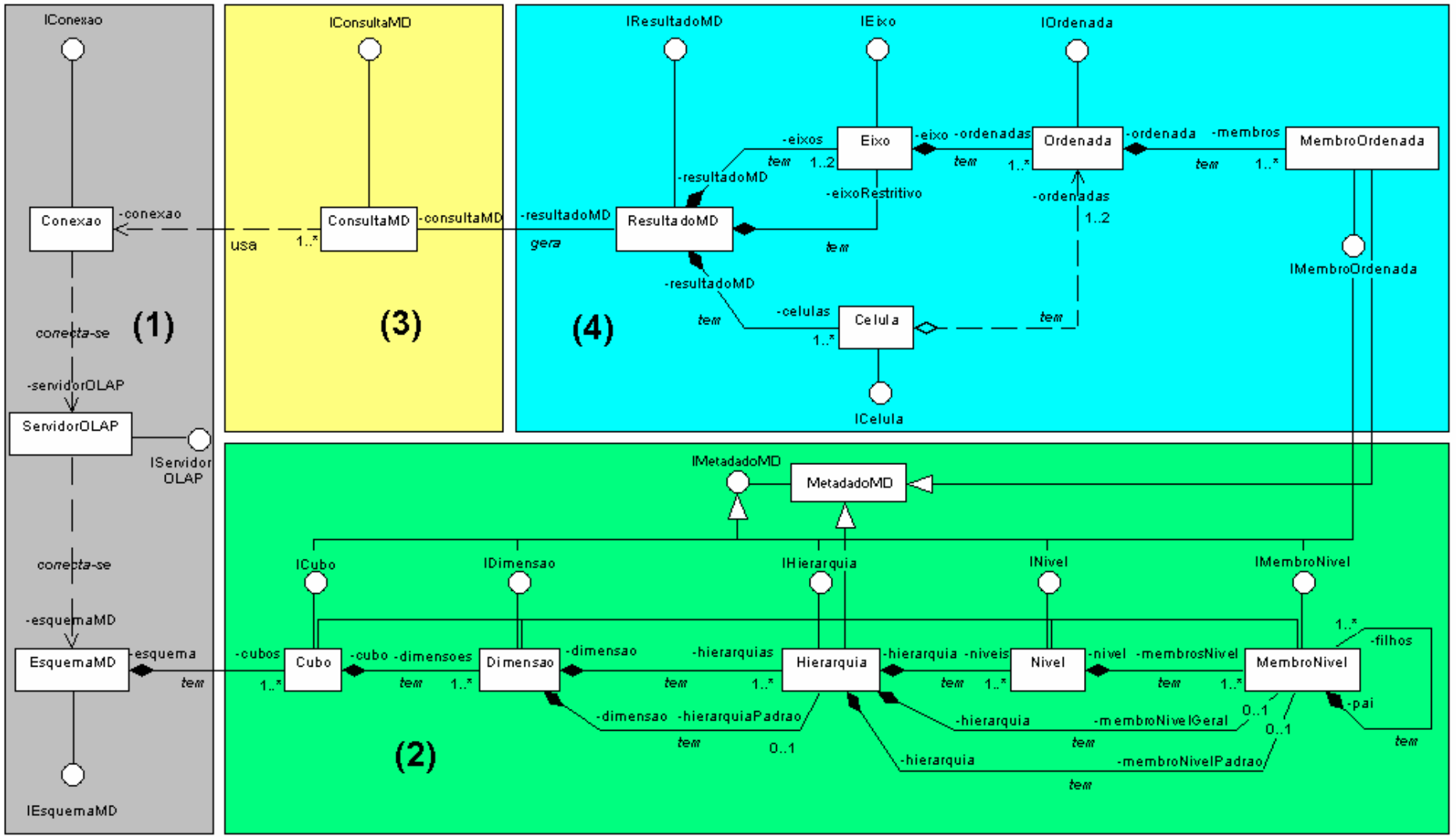


Figura 3.24 – Modelo de classes e interfaces de ODCI (FIDALGO, 2000)

A comunicação entre o cliente e o servidor JDCI é feita através da API RMI (*Remote Method Invocation*) (SUN, 2002). RMI é o pacote Java que permite o desenvolvimento de aplicações distribuídas sem preocupar-se com os problemas e detalhes de baixo nível, como por exemplo: conversões de protocolos, portas de comunicação e sincronização de processos. Além disso, RMI é capaz de fornecer uma camada de abstração na qual seus objetos remotos podem ser instanciados e acessados como se fossem locais.

JDCI, após aberta a conexão por parte do cliente, automaticamente reescreve os objetos Java/COM de *Visual J++*/ADO MD que representam objetos multidimensionais em objetos 100% Java padrão: cubos, dimensões, hierarquias, níveis e membros, além dos objetos que representam o servidor OLAP e o esquema multidimensional, são automaticamente instanciados e podem ser acessados através de suas interfaces remotas: IServidorOLAP, IEsquemaMD, ICubo, IDimensao, IHierarquia, INivel e IMembroNivel.

Após aberta a conexão com o servidor *OLE DB for OLAP,* outra operação que também pode ser realizada é a manipulação de consultas MDX. Estas através do método *definirConsulta*( ) da classe *Conexao*, definirá a consulta MDX que após ser executada (método *executar*( ) da classe *ConsultaMD*) automaticamente iniciará o processo de reescrita dos objetos Java/COM em objetos JDCI 100% conformes com Java padrão. Os objetos que compõem o resultado da consulta deverão ser manipulados pelas suas interfaces remotas: IConexao, IConsultaMD, IResultadoMD, IEixo, ICelula, IOrdenada e IMembroOrdenada.

A reescrita dos objetos Java/COM é realizada completamente pelos métodos construtores e, de forma simples, pode-se caracterizá-la como o processo de busca e reescrita das propriedades dos objetos de ADO MD e instanciação automática das suas classes subjacentes. Detalhes sobre a implementação e utilização de JDCI podem ser vistos em FIDALGO, 2000.

A arquitetura ODCI/JDCI pode ser vista na Figura 3.25.

3.2.6 Discussão

Como vimos, *OLE DB for OLAP* vem se tornando um padrão de API para disponibilização e integração de serviços OLAP, estando, contudo, limitada à plataforma Windows por manipular tipos proprietários COM.

O modelo de classes ODCI para acesso a dados em servidores *OLE DB for OLAP* e sua implementação Java JDCI, trazem como principal contribuição o fato de permitir o acesso a servidores MS OLAP a partir de qualquer plataforma, e, por utilizar uma linguagem de

programação largamente difundida com modelo de dados orientado a objetos, simplificar o processo de desenvolvimento de aplicações neste ambiente.

No entanto, JDCI, em sua versão inicial, não se preocupa com fatores como concorrência, distribuição e desempenho, que deveriam ser considerados no desenvolvimento de qualquer software voltado para uma aplicação prática. Como exemplo poderíamos citar o fato de JDCI instanciar todos os objetos de metadados em memória, mesmo antes de sua utilização por parte do cliente, o que, além de sobrecarregar o uso da memória (os objetos não são persistentes), torna o processo de conexão lento (o usuário deve aguardar o processo de reescrita de todos os objetos).

A nossa implementação de ODCI, JODI (*Java OLAP Data Interface*) tenta solucionar os problemas apontados por JDCI, melhorando o seu desempenho e servindo como camada de acesso aos dados a OCCOM, um aplicativo para mineração de exceções em cubos OLAP, como veremos nos capítulos seguintes.

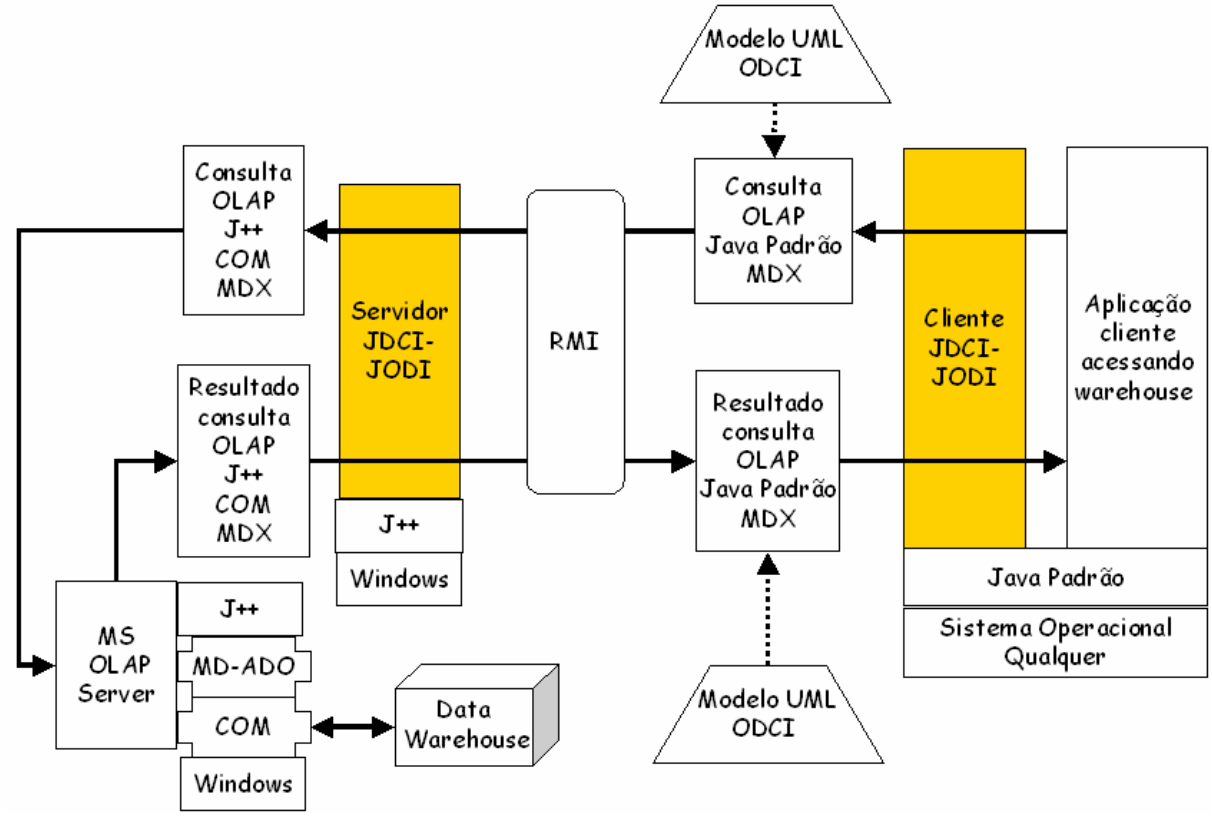


Figura 3.25 – Arquitetura ODCI/JDCI-JODI.

4 JODI: Java OLAP Data Interface

JODI (*Java OLAP Data Interface*) é uma interface de programação (API) que implementa o modelo ODCI, mostrado no capítulo anterior, conforme arquitetura apresentada na Figura 3.25, com algumas modificações. A finalidade de JODI é disponibilizar e integrar serviços OLAP, providos pelo servidor *OLE DB for OLAP*, com outras tecnologias de suporte à decisão e KDD, no nosso caso, com o minerador de exceções em células de cubos OLAP – OCCOM.

Assim como JDCI, JODI é implementado em uma arquitetura cliente-servidor, como pode ser visto na Figura 4.1. A comunicação entre o servidor e o cliente JODI é feita através da API RMI (*Remote Method Interface*) (SUN, 2002), que permite o acesso remoto aos objetos re-escritos e exportados pelas interfaces de JODI. RMI permite o desenvolvimento de aplicações distribuídas sem preocupação com os problemas e detalhes de baixo nível, como por exemplo, a conversão de protocolos, portas de comunicação e a sincronização de processos. Os objetos remotos RMI podem ser instanciados e acessados como se fossem locais, de forma transparente para o usuário.

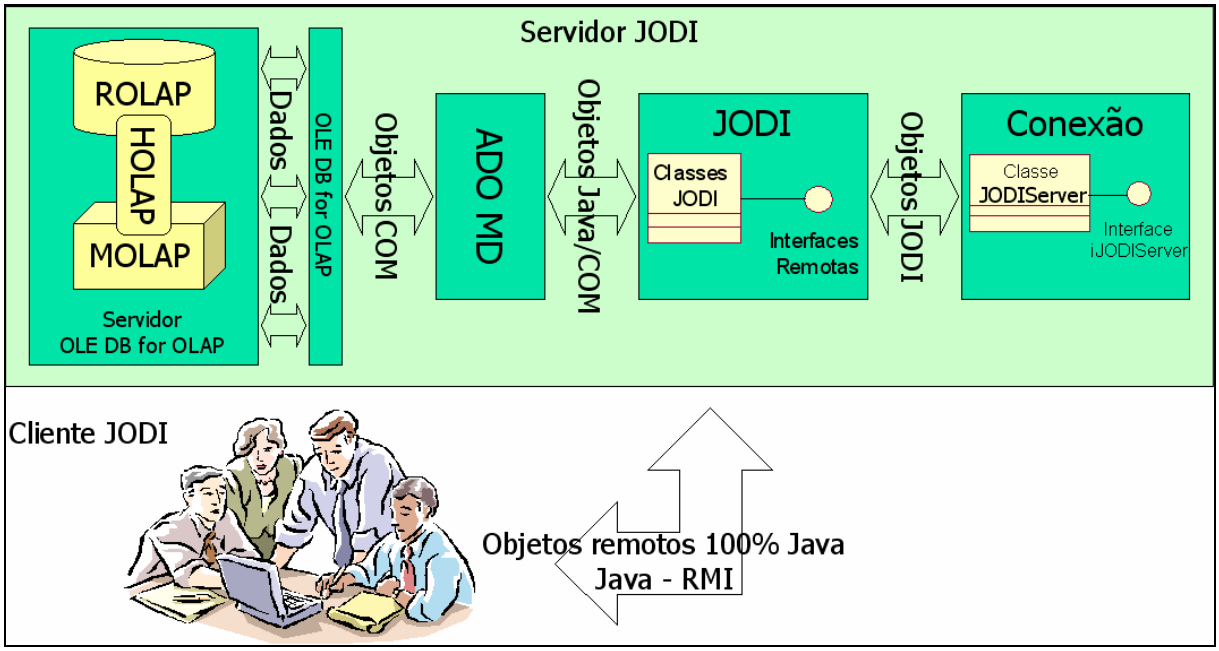


Figura 4.1 – Arquitetura JODI.

Para o desenvolvimento do software foi adotada uma abordagem seqüencial, que inicia no nível do sistema e avança através da análise, projeto, codificação, teste e suporte, como mostra a Figura 4.2.

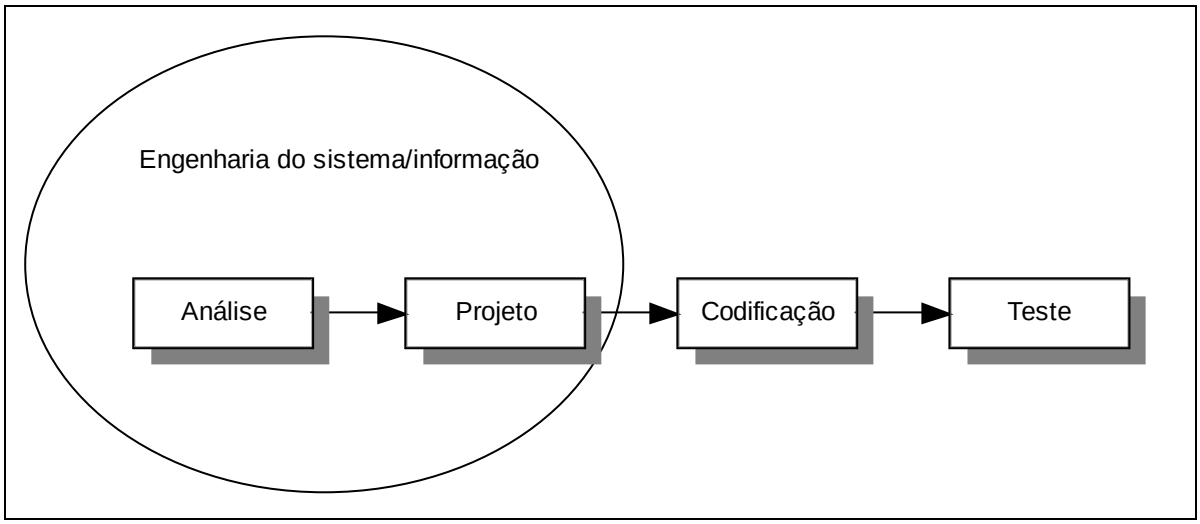


Figura 4.2 – O modelo seqüencial linear (PRESSMAN, 2001).

O modelo de desenvolvimento seqüencial linear engloba as seguintes atividades (PRESSMAN, 2001):

- Engenharia do sistema e modelagem: como o software é sempre parte de um sistema maior (ou negócio), o trabalho de desenvolvimento se iniciar pelo estabelecimento de requerimentos para todos os elementos do sistema e então alocação de alguns subconjuntos destes requerimentos para o software. Esta visão de sistema é essencial quando o software deve interagir com outros elementos como hardware, pessoas e bancos de dados.
- Análise de requisitos do software – O processo de coleta de requisitos é intensificado e com foco especificamente no software. Para entender a natureza do programa a ser construído, o engenheiro de software (“analista”) deve conhecer o domínio da informação para o software, bem como funções requeridas, comportamento, performance e interface.
- Projeto – A elaboração do projeto do software é, atualmente, um processo de vários passos, que tem como foco quatro atributos distintos de um programa: estrutura de dados, arquitetura do software, representações de interface e detalhe do procedimento (algoritmo). O processo de modelagem traduz requisitos em uma representação do software que possa garantir qualidade antes que a codificação inicie.
- Geração do código – O projeto deve ser traduzido de forma que a máquina reconheça. O processo de geração do código realiza esta tarefa.
- Teste – Uma vez que o código foi gerado, os testes no programa iniciam. O processo de teste é voltado para a lógica interna do software, de forma a garantir que todas as funções tenham sido testadas, e para a funcionalidade externa. Os testes devem ser conduzidos

para descobrir erros e garantir que uma entrada previamente estabelecida irá produzir resultados equivalentes aos esperados.

Apesar do modelo seqüencial linear ser o mais antigo e mais largamente utilizado paradigma para engenharia de software, críticas têm sido feitas quanto a sua eficácia. Porém, as críticas envolvem principalmente a dificuldade de interação com clientes e a aplicação desse paradigma a times de desenvolvimento. Como o nosso trabalho não envolve time de desenvolvimento, nem tão pouco relação com clientes, acreditamos que este modelo é adequado ao desenvolvimento de nossa aplicação.

A seguir veremos cada uma das etapas do processo de desenvolvimento de JODI.

4.1 Análise de Requisitos

O principal objetivo de JODI é disponibilizar e integrar serviços OLAP, providos pelo servidor *OLE DB for OLAP*. Para realizar esta tarefa, JODI deve possuir os seguintes requisitos:

- Manter a proposta de desenvolvimento em três camadas, apresentada por Fidalgo (2000);
- Como o MSOLAP está limitado à plataforma Windows por manipular tipos proprietários COM, o servidor JODI deve necessariamente ser desenvolvido para esta plataforma;
- O servidor JODI não deve estar instalado necessariamente na mesma máquina do servidor *OLE DB for OLAP*;
- O cliente JODI deve ser independente de plataforma, tornando-o, portanto, um sistema portável;
- Deve manter a compatibilidade com a proposta de *OLE DB for OLAP*, corrigindo as suas limitações;
- Permitir conexão ao servidor *OLE DB for OLAP* – Para realizar esta tarefa, JODI deve receber como parâmetros o nome da máquina onde está instalado o servidor, o nome do esquema multidimensional a conectar e o nome do servidor *OLE DB for OLAP*;
- Permitir acesso (consulta) aos metadados do esquema multidimensional – Após a conexão ao servidor *OLE DB for OLAP*, o cliente poderá acessar as seguintes informações manipuladas por ADO MD: cubos, dimensões, hierarquias, níveis e membros;

- Permitir que os clientes realizem consultas multidimensionais (MDX) – O cliente deve fornecer o texto de consulta como parâmetro e, caso esta seja bem sucedida, deverá poder acessar os objetos manipulados por ADO MD: células, eixos, posições e membros;
- Por servir como camada intermediária entre OCCOM (*OLAP Cuboid Cell Outlier Miner*) e *OLE DB for OLAP*, como pode ser visto na arquitetura do projeto MATRIKS apresentada na Figura 2.6, JODI deve se preocupar com questões como concorrência (acesso de usuários simultâneos), distribuição (deve ser o mais simples possível, simplificando a sua instalação) e desempenho (como JODI é uma camada intermediária um mal desempenho de JODI pode comprometer todo o projeto).

Mostramos acima, tanto requisitos funcionais (que descrevem os serviços providos) como requisitos não funcionais (que definem limitações no sistema e no processo de desenvolvimento).

Como vimos, o único ator[1] do sistema é o cliente, que pode realizar as atividades de conectar ao servidor JODI, conectar ao servidor OLAP, acessar metadados do esquema multidimensional e realizar consultas multidimensionais, como pode ser visto no diagrama de caso de uso[2] da Figura 4.3.

Note que para que o cliente possa conectar ao servidor OLAP, ele deve realizar a atividade de conectar-se ao servidor JODI anteriormente. De forma similar, para que possa consultar metadados multidimensionais ou realizar uma consulta multidimensional (MDX), o cliente deverá estar conectado ao servidor OLAP (*OLE DB for OLAP*).

O diagrama de seqüência[3] da Figura 4.4 mostra em detalhes a seqüência de operações que deverão ser realizadas por JODI para cada uma das atividades do diagrama de caso de uso.

O passo seguinte no processo de desenvolvimento do software é a elaboração do projeto, de acordo com os requisitos especificados.

---

[1] Um ator representa um papel que um ser humano, um dispositivo de hardware ou até outro sistema desempenha com o sistema (BOOCH; RUMBAUGH; JACOBSON, 2000).

[2] É um diagrama UML aplicado para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementado (BOOCH; RUMBAUGH; JACOBSON, 2000).

[3] Um diagrama de seqüência é um diagrama UML que mostra uma interação, formada por um conjunto de objetos e seus relacionamentos, incluindo as mensagens que poderão ser enviadas entre eles, e que dá ênfase à ordenação temporal das mensagens (BOOCH; RUMBAUGH; JACOBSON, 2000).

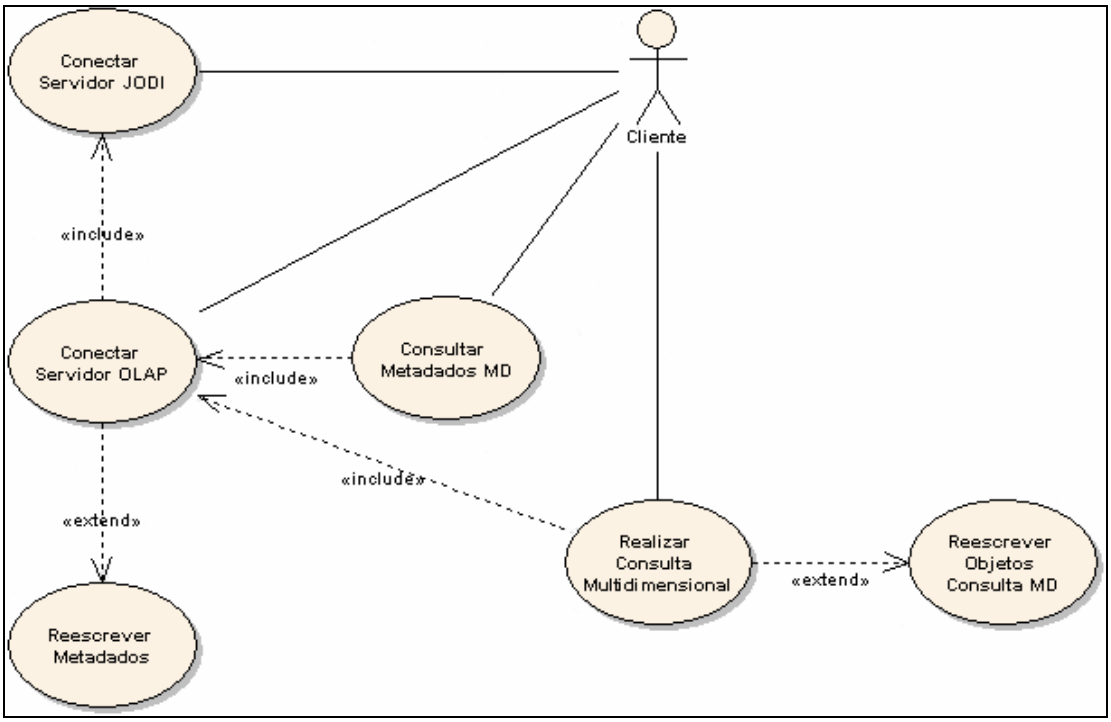


Figura 4.3 – Diagrama de caso de uso – JODI.

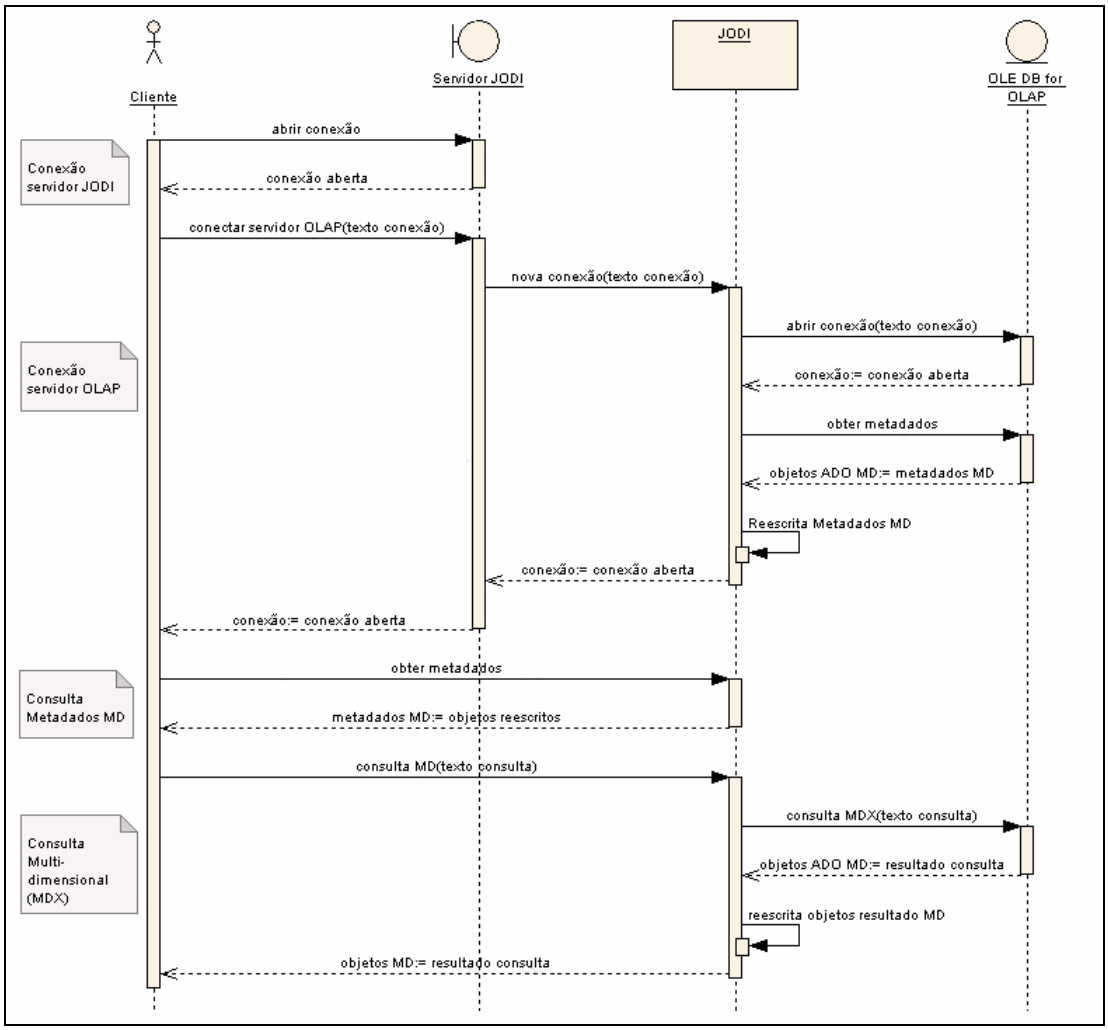


Figura 4.4 – Diagrama de Seqüência – JODI.

4.2 Modelo

Como vimos no capítulo 3, ODCI foi baseado em conceitos padrão OLAP e inspirado nos pontos positivos do modelo de programação *Visual J++* com ADO MD, de forma que sua implementação deve: (1) re-escrever os objetos Java/COM em objetos 100% Java padrão, que poderão usufruir da portabilidade da linguagem Java; e (2) manter a compatibilidade com a proposta de *OLE DB for OLAP*, corrigindo as suas limitações.

Algumas críticas podem ser feitas ao modelo ODCI apresentado por Fidalgo e Lino (2000):

- Como visto anteriormente, a proposta ODCI é de implementação de um modelo em três camadas: (1) a camada do servidor OLAP, no nosso caso *OLE DB for OLAP*; (2) a camada do servidor JODI, que reescreve os objetos de *OLE DB for OLAP* no formato de objetos Java padrão e que é escrita em *Visual J++*; e (3) a camada do cliente JODI que pode estar em qualquer ambiente que utilize Java padrão. No entanto, não aparece no modelo de ODCI o servidor RMI para acesso remoto ao qual o cliente deve necessariamente se conectar para acessar os objetos ODCI.
- Não vimos a necessidade de inclusão das classes *ServidorOLAP* e *EsquemaMD*, presentes no modelo ODCI, já que, para se conectar ao servidor OLAP, o usuário é obrigado a informar o nome do servidor e o nome do esquema multidimensional que deseja acessar no momento da conexão. Na nossa proposta tais informações se encontram embutidas na classe conexão (*cConnection*).
- Os métodos que retornam o tipo da classe no formato de um número inteiro não dão sentido à informação retornada. Para solucionar este problema foram inseridas constantes nas interfaces que indicam cada tipo permitido.
- O modelo é dividido em 4 conjuntos de classes, conforme visto na Figura 3.21: (1) conexão, (2) metadados, (3) consulta multidimensional, e (4) resultado de consulta multidimensional. Propomos que o modelo passe a ser composto apenas de 3 grupos: (1) conexão, (2) metadados e (3) consulta multidimensional, tornando a sua representação mais simples e concisa.
- Para melhor visualização do modelo, optamos por remover a super-classe *MetadadoMD*, já que a sua contribuição é muito pequena (atributos e métodos de identificação das classes – nome, nome único, rótulo e descrição).

Na Figura 4.5 podemos visualizar o modelo ODCI, já com as alterações propostas e de maneira atender aos requisitos da seção anterior.

O modelo ODCI proposto é dividido em 3 grupos de classes, que permitem as seguintes ações:

(1) Conectar-se ao servidor JODI e a um servidor *OLE DB for OLAP*. Composto pelas classes *JODIServer*, *cConnection* e suas respectivas interfaces.

(2) Consultar todos os objetos de um esquema multidimensional, após a conexão ao servidor JODI e a um servidor *OLE DB for OLAP*. Composto pelas classes *cCube*, *cDimension*, *cHierarchy*, *cLevel*, *cLevelMember* e suas interfaces.

(3) Realizar consultas multidimensionais baseada na linguagem MDX e acessar os objetos de seus resultados. Composto pelas classes *cMDQuery*, *cMDResult*, *cCell*, *cAxis*, *cPosition*, *cPositionMember* e suas respectivas interfaces.

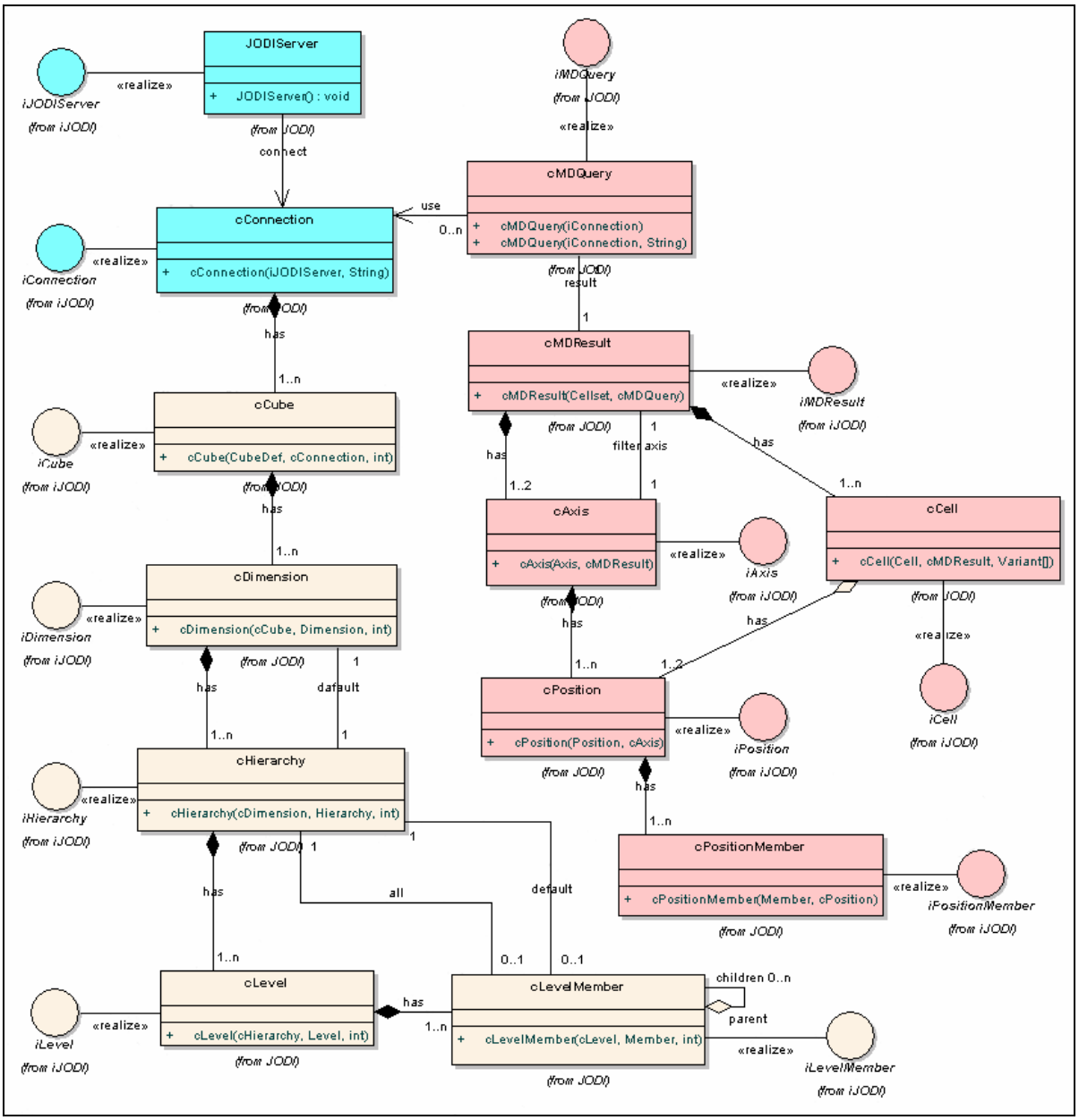


Figura 4.5 – Modelo ODCI Proposto

A seguir apresentaremos a descrição das classes e interfaces do modelo JODI proposto. A descrição detalhada de todos os métodos das classes de JODI pode ser encontrada no Apêndice A.

Na Figura 4.6 podemos ver a classe *JODIServer*, que implementa a interface *iJODIServer*, responsáveis pela conexão ao servidor JODI. O método construtor da classe automaticamente instancia o servidor e aguarda por conexões.

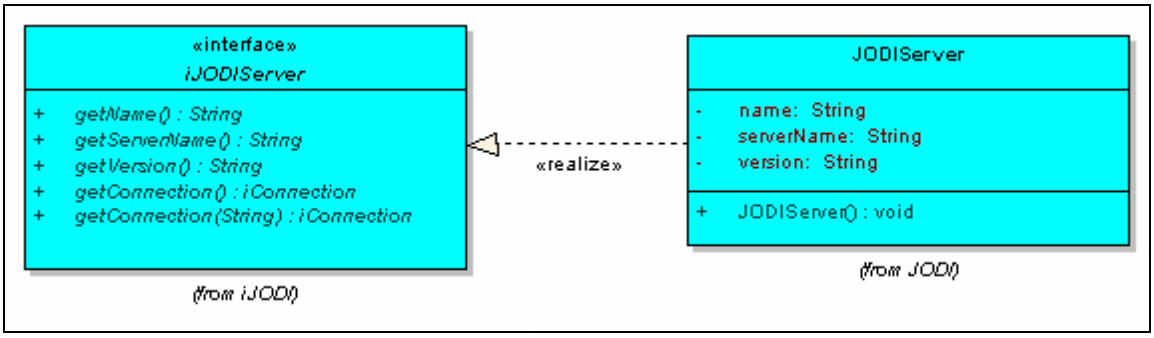


Figura 4.6 – Classe *JODIServer* e sua interface *iJODIServer*

A classe responsável pela conexão ao servidor OLAP *cConnection* e a sua interface *iConnection* podem ser vistas na Figura 4.7.

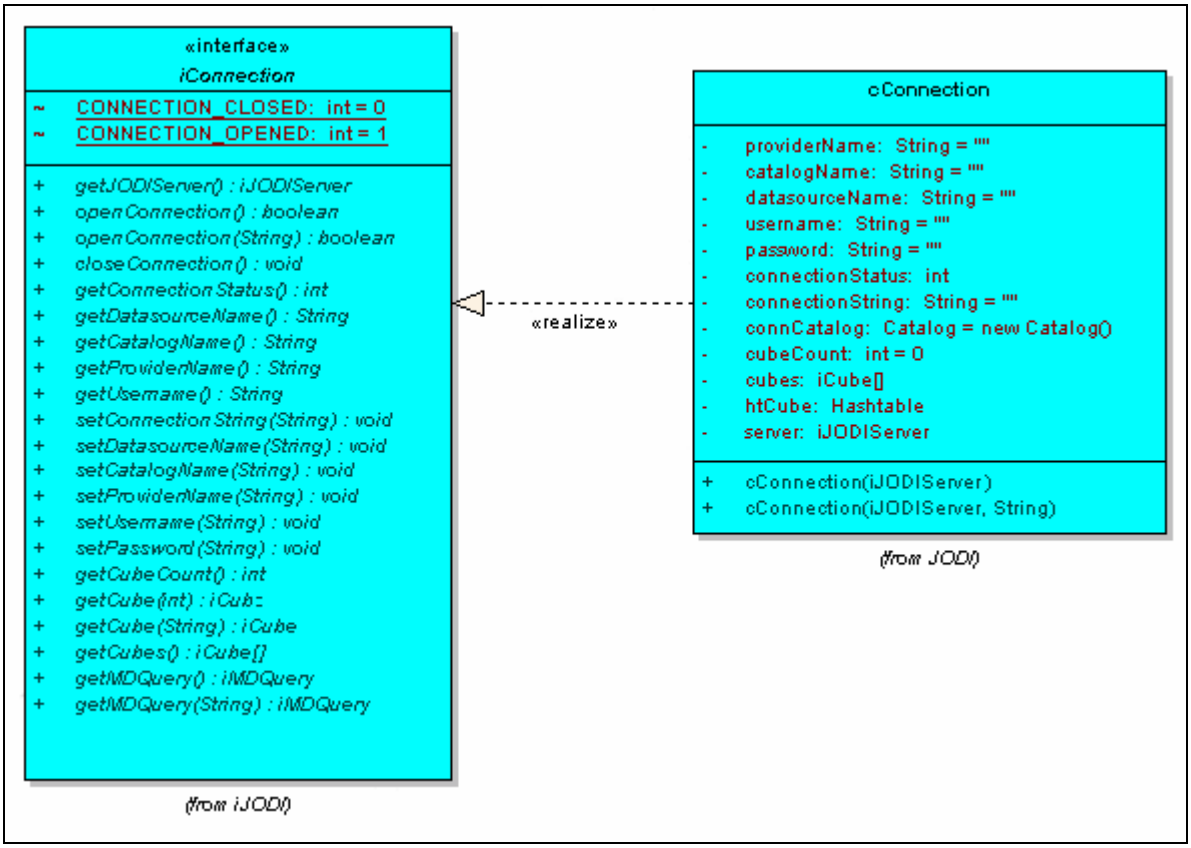


Figura 4.7 – Classe *cConnection* e a sua interface *iConnection*.

ADO MD não possui uma classe de conexão bem definida. Para resolver esta limitação, a classe *cConnection* de ODCI encapsula os métodos *setConnection*() e *Open*(),

respectivamente de ADO MD e ADO, para abrir a conexão com o servidor *OLE DB for OLAP*. E os métodos *close*() e *getState*() de ADO para respectivamente fechar e retornar o estado da conexão. A Tabela A.2, presente no Apêndice A, possui uma descrição dos métodos implementados por *cConnection*.

A seguir apresentaremos as classes de metadados do esquema multidimensional: *cCube*, *cDimension*, *cHierarchy*, *cLevel* e *cLevelMember*.

A classe *cCube* (Figura 4.8) implementa a interface *iCube* e re-escreve os objetos COM da interface *CubeDef* de ADO MD (Figura 3.11), além de registrar a conexão a que o cubo pertence e instanciar, automaticamente, todos os objetos correspondentes às suas dimensões.

A classe *cCube* e todas as demais classes de ODCI, diferentemente da interface *CubeDef* e o restantes das interfaces de ADO MD, manipulam explicitamente os seus atributos, pois em ODCI, os atributos de um objeto, são manipulados exclusivamente pelas suas classes/interfaces e não por uma outra, como faz ADO MD com as suas interfaces *Properties* e *Property* (Figura 3.11). Por exemplo, enquanto a classe *cCube* possui métodos para explicitamente manipular seus atributos tipo (*type*), data de criação (*creationDate*) e data de atualização (*updateDate*), a interface *CubeDef* só poderá realizar a mesma operação após especificar os métodos: (1) *getProperties*() – para retornar a coleção de atributos do objeto da interface *CubeDef*; (2) *getItem*(*arg0: Variant*) – para retornar um atributo, dado o ordinal que o identifica nessa coleção; e (3) *getValue*() – para retornar o valor desse atributo. Ou seja, enquanto um objeto ODCI pode retornar qualquer um de seus atributos, com a simples declaração *objetoODCI.métodoParaObterAtributo*(), um objeto de ADO MD só poderá retornar o mesmo atributo se fizer a seguinte declaração *objetoADOMD.getProperties*(). *getProperty*(*ordinal*).*getValue*(). Por isso, como citado na seção 3.2.1, algumas propriedades dos objetos de ADO MD (exceto nome e descrição), só são acessadas mediante o ordinal que a representa, o que dificulta o uso deste modelo de programação.

Os métodos implementados por *cCube* podem ser vistos na Tabela A.3, do Apêndice A.

A seguir veremos a classe *cDimension* (Figura 4.9) que implementa a interface *iDimension* e é responsável por reescrever os objetos COM da interface *Dimension* de ADO MD. *cDimension* implementa os métodos listados na Tabela A.4, presente no Apêndice A. Através desta classe é possível acessar informações acerca das dimensões do cubo, tais como tipo, cardinalidade e hierarquias.

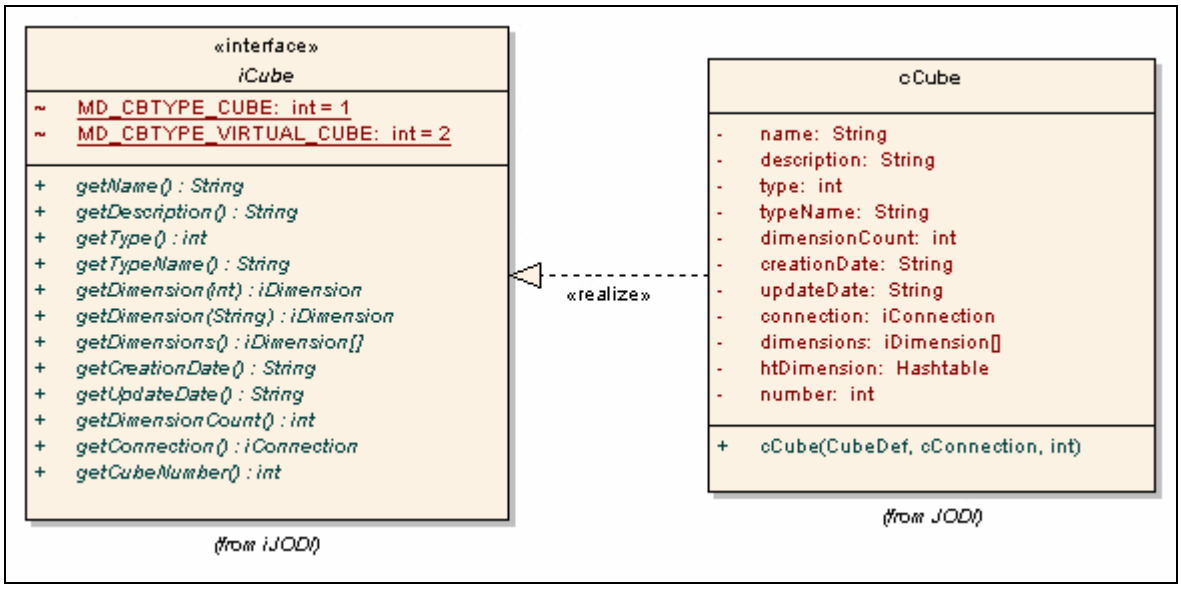


Figura 4.8 – Classe *cCube* e a sua interface *iCube*.

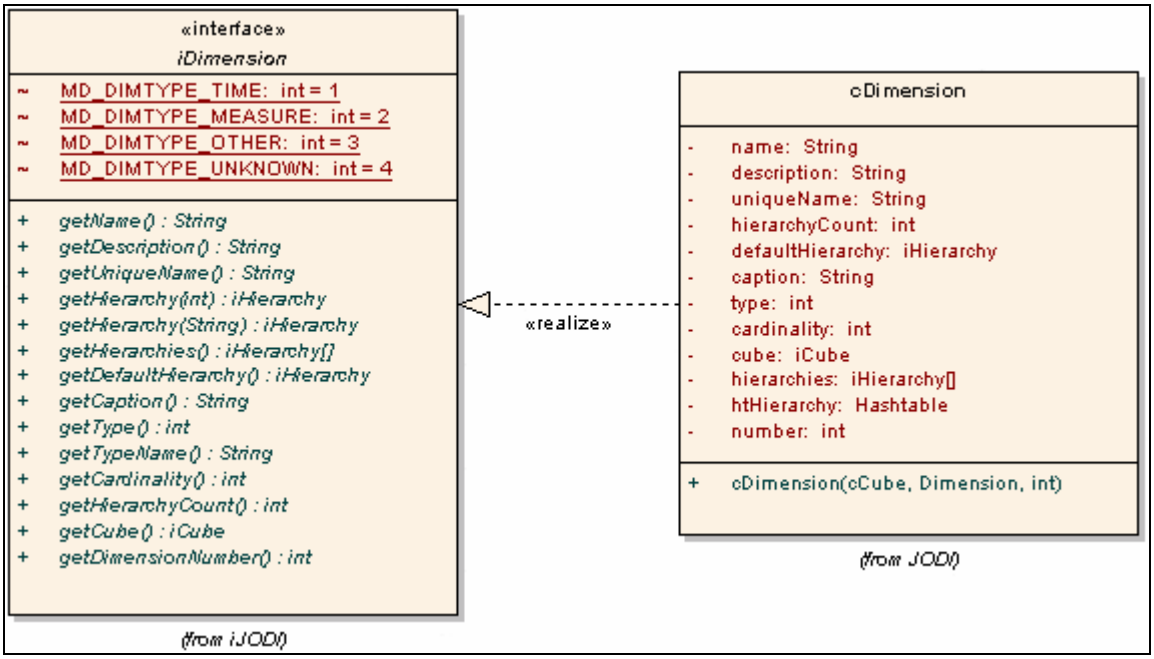


Figura 4.9 – Classe *cDimension* e a sua interface *iDimension*.

Como definido no modelo ODCI, uma dimensão pode ter mais de uma hierarquia. Quando isto ocorrer, pode-se definir apenas através da interface gráfica do servidor *OLE DB for OLAP*, qual delas é a hierarquia padrão que será retornada pelo método *getDefaultHierarchy*().

A classe *cHierarchy*, apresentada na Figura 4.10, além de implementar a interface *iHierarchy*, deve ser capaz de reconstruir os objetos COM da interface *Hierarchy* de ADO MD, bem como instanciar, automaticamente, todos os objetos da classe *cLevel* existentes em um objeto de hierarquia.

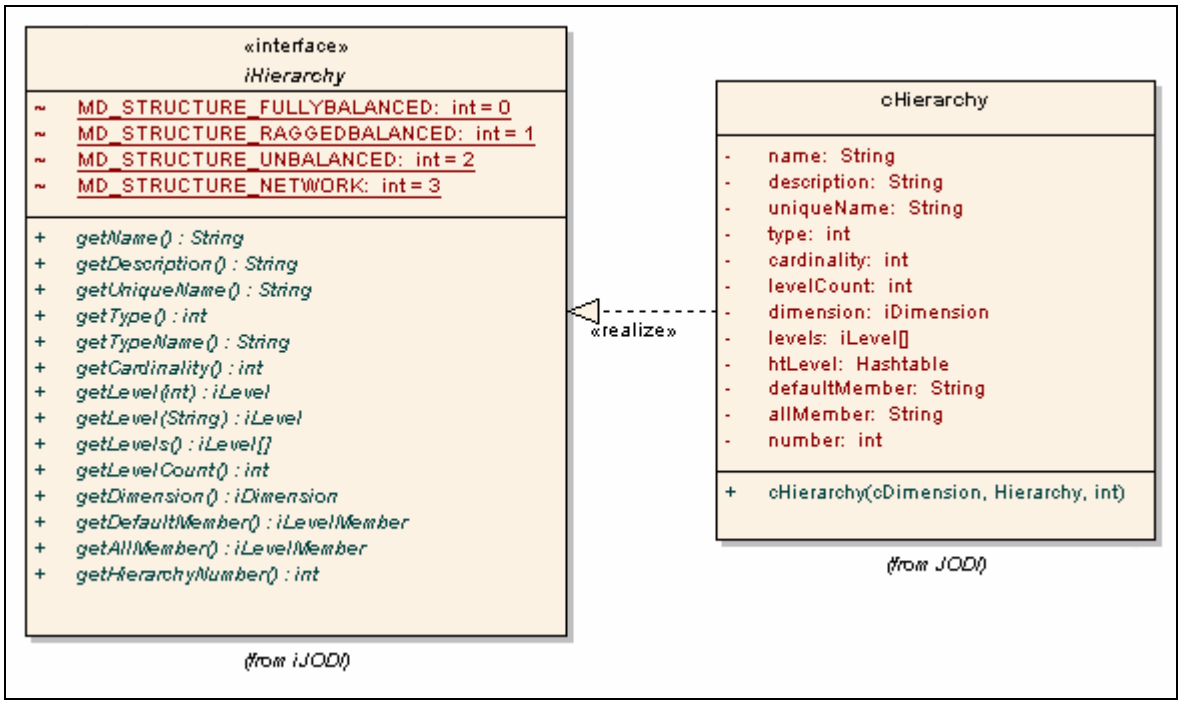


Figura 4.10 – Classe *cHierarchy* e a sua interface *iHierarchy*.

Os métodos implementados pela classe *cHierarchy* podem ser vistos na Tabela A.5 do Apêndice A. Através desta classe é possível acessar propriedades das hierarquias, como tipo, cardinalidade, os seus membros constituintes, o membro padrão e o membro geral.

A Figura 4.11 ilustra a classe *cLevel*, que deve ser capaz de re-escrever os objetos COM da interface *Level* de ADO MD. c*Level* implementa os métodos listados na Tabela A.6 e permite acesso às propriedades do nível, tais como tipo, nome, descrição, profundidade, cardinalidade e membros do nível.

A Figura 4.12 ilustra a classe *cLevelMember* que implementa a interface *iLevelMember* e, analogamente a todas as outras classes de metadados do esquema multidimensional, re-escreve no seu método construtor os objetos COM de ADO MD (objeto da interface *Member*). *cLevelMember* implementa os métodos da Tabela A.7, permitindo acesso às seguintes propriedades do membro do nível: nome, descrição, nome único, apelido e número, além de seu pai e filhos na hierarquia do nível.

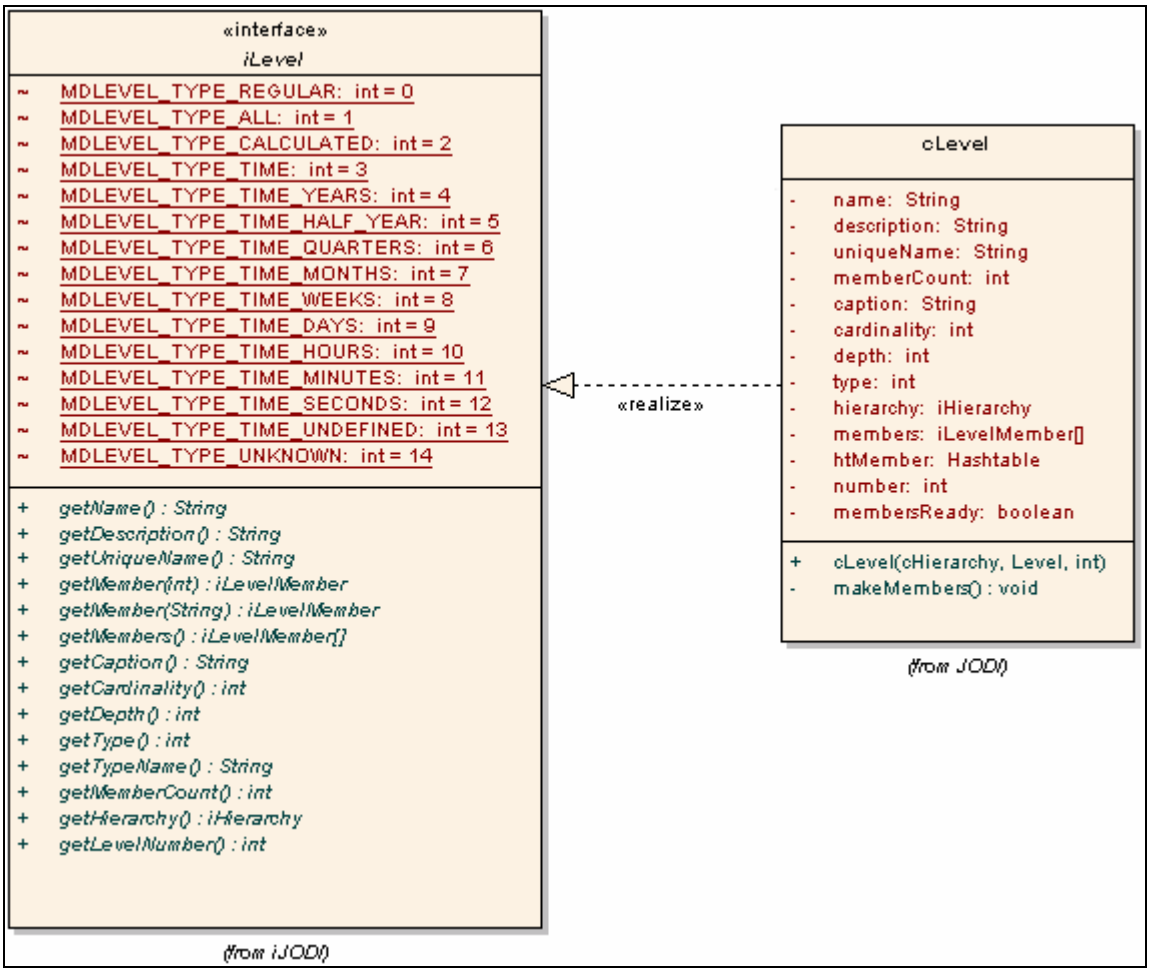


Figura 4.11 – Classe *cLevel* e a sua interface *iLevel*.

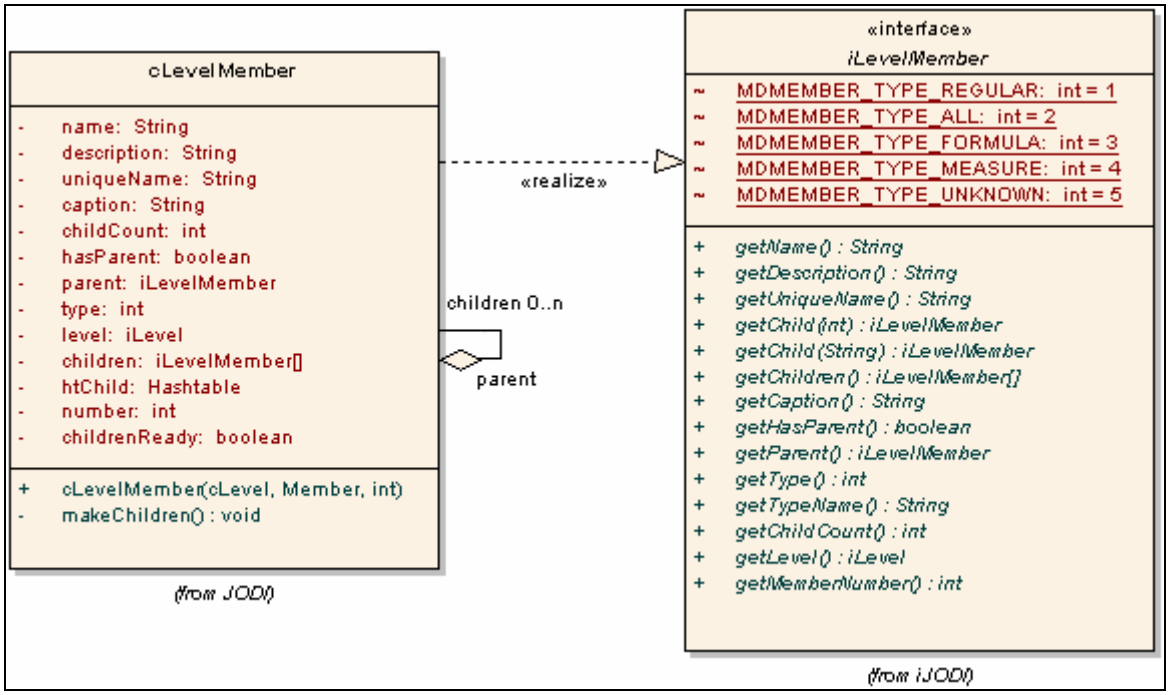


Figura 4.12 – Classe *cLevelMember* e a sua interface *iLevelMember*.

Veremos a seguir as classes responsáveis pela realização de consultas multidimensionais: *cMDQuery, cMDResult, cCell, cAxis, cPosition* e *cPositionMember*.

A classe *cMDQuery* (Figura 4.13) implementa a interface *iMDQuery* que, após aberta a conexão ao esquema multidimensional do servidor *OLE DB for OLAP*, se destinará à manipulação de uma consulta MDX. Através do seu método *open*(), *cMDQuery* inicia o processo de re-escrita dos objetos das interfaces *Axis, Cell, Position e Member* de ADO MD nos respectivos objetos das classes *cAxis, cCell, cPosition* e *cPositionMember* de ODCI, que irão compor a classe *cResultMD*, mostrada na Figura 4.15.

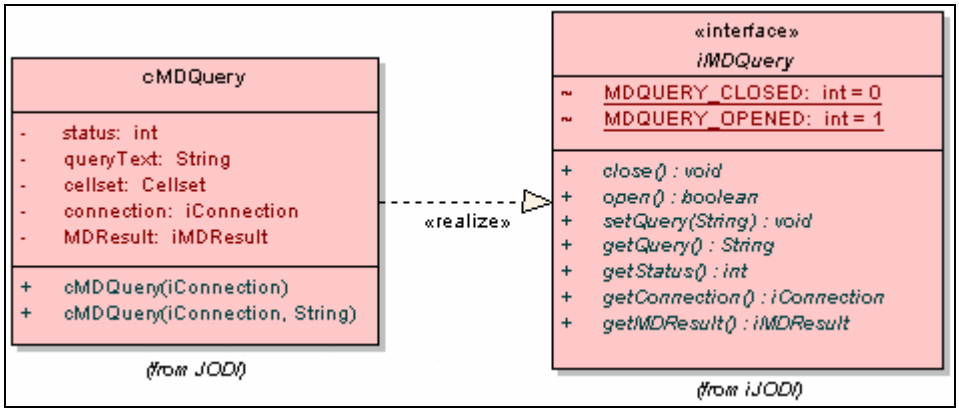


Figura 4.13 – Classe *cMDQuery* e a sua interface *iMDQuery*.

A classe *cMDQuery* implementa os métodos da Tabela A.8. Através dela podemos descobrir o texto da consulta e o status da conexão, além de acessar o resultado da consulta multidimensional.

Após uma consulta MDX ser aberta, um objeto da classe *cMDResult* (Figura 4.15) é instanciado. Este, para re-escrever os objetos COM da classe *Cellset* e das interfaces *Axis, Cell* e *Position* de ADO MD, deve no método construtor de sua classe, se encarregar de automaticamente registrar o objeto da classe *cMDQuery* que o chamou e, posteriormente, instanciar objetos da classe *cAxis*, que, por sua vez, fará o mesmo para os objetos das classes *cPosition* e *cPositionMember*. Note que, por questões de desempenho, não instanciamos automaticamente objetos da classe *cCell*, diferentemente de JDCI, implementação anterior de ODCI (FIDALGO, 2000), como detalharemos mais tarde.

Para ilustrar o uso de consultas multidimensionais, utilizaremos o cubo de dados da Figura 4.14, que pode ser gerado através dos métodos *myQuery.setQuery*(*"SELECT {[Product].[Product Family].Members} ON COLUMNS, CROSSJOIN({[Gender].[Gender]. Members}, {[Marital Status]. [Marital Status].Members}) ON ROWS FROM [Sales] WHERE [Measures].[Unit Sales]"*) e *myQuery.open*().

		Products		
Gender	Marital Stat.	**Drink**	**Food**	**Non-Cons.**
F	**M**	6.207,00	47.187,00	11.942,00
F	**S**	5.995,00	47.627,00	12.600,00
M	**M**	5.969,00	47.742,00	12.749,00
M	**S**	6.426,00	49.384,00	12.945,00

Figura 4.14 – Cubo de exemplo para consultas MDX.

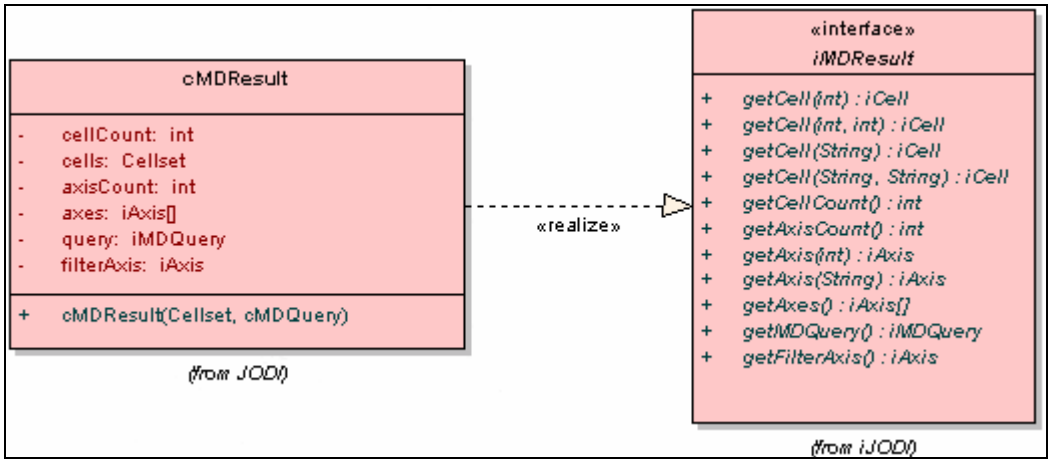


Figura 4.15– Classe *cMDResult* e sua interface *iMDResult*.

Os principais métodos implementados pela classe *cMDResult* podem ser vistos na Tabela A.9. Através desta classe podemos acessar objetos do resultado de uma consulta multidimensional (eixos e células). Por exemplo, a célula *Food+M+M* (valor de 47.742,00) da Figura 4.14 pode ser acessada através do comando *myMDResult.getCell*(1,2) ou *myMDResult.getCell*(7), ou ainda *myMDResult.getCell*("[Food]+[M].[M]").

A Figura 4.16 ilustra a interface *iAxis* e a classe que a implementa *cAxis* (eixo), que é responsável por re-escrever os objetos COM da interface *Axis*. Um eixo é composto de posições. Por exemplo, o eixo 1 (linha) da Figura 4.14 é composto por 4 posições: *[F].[M]*, *[F].[S]*, *[M].[M]* e *[M].[S]*.

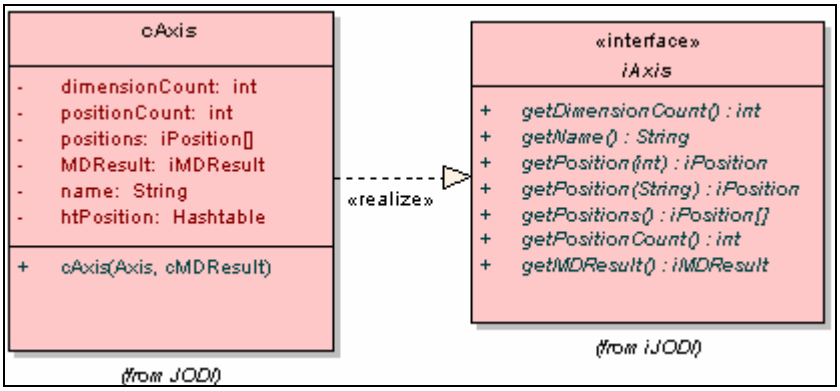


Figura 4.16 – Classe *cAxis* e a sua interface *iAxis*.

Os principais métodos implementados por *cAxis* são mostrados na Tabela A.10.

A Figura 4.17 ilustra a interface *iCell* e a classe que a implementa *cCell* (célula), que é responsável por re-escrever os objetos COM da interface *Cell*.

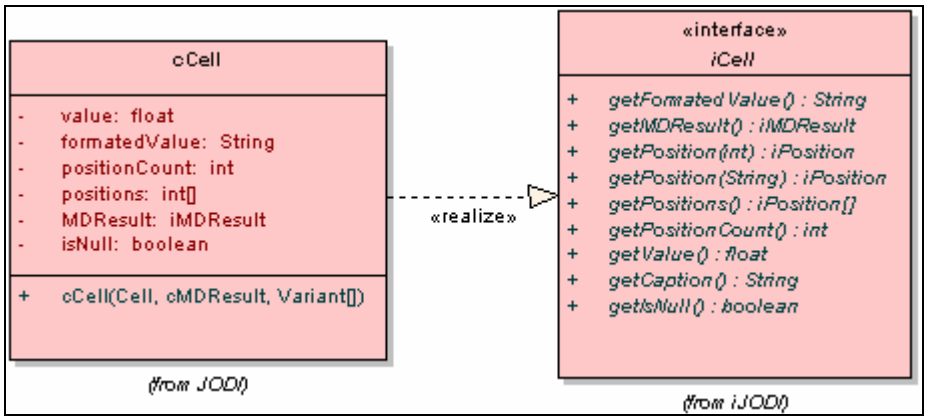


Figura 4.17 – Classe *cCell* e a sua interface *iCell*.

cCell implementa os seguintes métodos listados na Tabela A.11. Através de *cCell* podemos acessar as propriedades das células resultantes de uma consulta MDX, tais como valor, valor formatado, rótulo, posições, e se a célula possui ou não um valor nulo.

A Figura 4.18 ilustra a interface *iPosition* e a classe que a implementa *cPosition* (posição), que é responsável por re-escrever os objetos COM da interface *Position*.Uma posição de uma eixo é composta por membros correspondentes aos membros de um nível em uma dimensão (*cLevelMember*). Por exemplo, a posição 1 (*[F].[S]*) do eixo 1 (linha) do cubo da Figura 4.13, é composta pelos membros *[F]* e *[S]*, que correspondem aos membros *[Gender].[All Gender].[F]* da dimensão *[Gender]*, e *[Marital Status].[All Marital Status].[S]* da dimensão *[Marital Status]*, do esquema multidimensional.

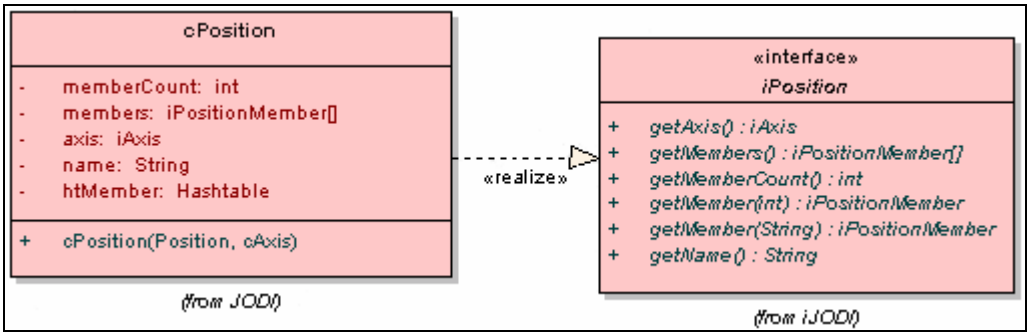


Figura 4.18 – Classe *cPosition* e a sua interface *iPosition*.

cPosition implementa os métodos listados na Tabela A.12.

Finalizando a descrição do modelo ODCI proposto, apresentamos na Figura 4.19 a interface *iPositionMember* e a classe responsável por sua implementação *cPositionMember*, que re-escreve os objetos COM da interface *Member* de ADO MD, correspondentes apenas ao sub-cubo MDX.

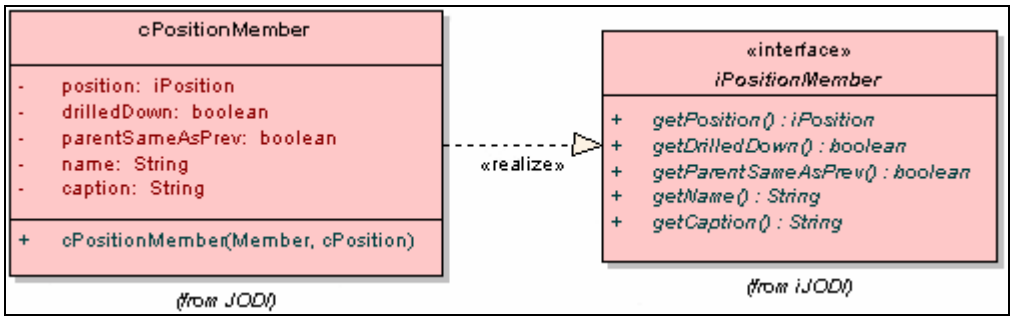


Figura 4.19 – Classe *cPositionMember* e a sua interface *iPositionMember*.

cPositionMember implementa os métodos da Tabela A.13.

Ao contrário dos membro de um cubo (classe *cLevelMember* – Figura 4.12), os membros do resultado de uma consulta MDX (classe *cPositionMember*) são apenas uma visão estática de uma parte do cubo e não da totalidade do mesmo. Ou seja, os membros do resultado de uma consulta MDX perdem os seus vínculos com a estrutura do cubo e, por isso, não possuem a principal característica dos membros de um cubo, que é o fato de possuir filhos. Desta forma, diferentemente de ADO MD, que implementa esses membros unicamente na interface *Member* e apenas informa que o uso indevido deles, gerando um erro, ODCI propõe a definição das classes *cLevelMember* e *cPositionMember*, para que esses objetos sejam manipulados distintamente de forma a impedir a ocorrência do erro citado. Por exemplo, na execução de uma instrução do tipo *myADOPosition.getMembers*().*getItem*(*0*).*getChildren*(), um erro seria retornado.

O fato de um membro do resultado de uma consulta MDX perder o vínculo com o cubo implica na necessidade de redefinição e execução de uma nova consulta MDX para realização de operações como *drill down* ou *roll up*, pois só assim, após a redefinição e execução da consulta MDX, é que se pode obter os dados que irão gerar o novo sub-cubo MDX.

4.3 Implementação

Ao contrário da primeira implementação de ODCI – JDCI (*Java Data Cube Interface*) – que não se preocupava com questões como desempenho, concorrência e distribuição (FIDALGO, 2000), JODI (*Java Olap Data Interface*), por servir como camada intermediária entre OCCOM (*OLAP Cuboid Cell Outlier Miner*) e *OLE DB for OLAP*, como pode ser visto na arquitetura do projeto MATRIKS apresentada na Figura 2.6, teve que sofrer algumas modificações a fim de melhorar o desempenho apresentado por JDCI, simplificar a sua distribuição e permitir conexões concorrentes.

Implementado em uma arquitetura cliente-servidor, como pode ser visto na Figura 4.1, o lado servidor de JODI, obrigatoriamente escrito em *Visual J++*[4], deve dar suporte a re-escrita dos objetos Java/COM em objetos 100% Java padrão, o que não pode ser feito em outra plataforma Java.

Nesta arquitetura, quando o servidor JODI é inicializado, o mesmo é responsável apenas por instanciar um objeto da classe *JODIServer*. Após este objeto estar iniciado, o cliente o aciona remotamente de forma a obter um objeto do tipo *cConnection*, responsável pela conexão ao servidor *OLE DB for OLAP*. Ao realizar a conexão, todos os objetos Java/COM são re-escritos em objeto Java padrão e o cliente, a partir da troca de objetos remotos oferecidos pelas interfaces de JODI, está apto a executar sua aplicação OLAP em uma máquina remota.

O servidor JDCI após instanciar a classe *Conexão*, esta se encarrega de automaticamente instanciar os objetos *ServidorOLAP, EsquemaMD, Cubo, Dimensão, Hierarquia, Nível e MembroNivel*, o que muitas vezes é uma operação demorada. Para solucionar este problema e considerando que o usuário raramente navega por todos os membros de todos os níveis dos metadados de um esquema multidimensional, JODI não instancia os objetos de *cLevelMember* no momento da conexão. Apenas quando o usuário solicita um membro através da chamada dos métodos *getMember* e *getMembers* de *cLevel*, os membros, caso ainda não tenham sido instanciados, são gerados através da chamado ao método privado *makeMembers*. De forma similar, a relação pai-filho (*parent-child*) de *cLevelMember*, também uma operação demorada, só é gerada no momento em que algum dos métodos *getChild* ou *getChildren* é chamado, através da chamada ou método privado *makeChildren*. Com estas medidas, o tempo necessário para conexão ao servidor OLAP e a quantidade de memória utilizada foram reduzidos sensivelmente.

Outra medida adotada para melhorar o desempenho e reduzir o gasto de memória é o fato de JODI não manter os objetos de células do resultado de uma consulta MDX instanciados, como ocorre em JDCI. No momento em que células são solicitadas através de chamadas ao método *getCell* da classe *cMDResult*, a célula é instanciada e passada como resposta ao método. Como as células não permanecem instanciadas, não existe, portanto, em JODI, o método *getCells*, que retorna a coleção de células do resultado de uma consulta MDX, fazendo com que o usuário deva manipular as células uma a uma. Todas estas medidas para melhoria de desempenho foram adotas após a realização dos testes descritos na próxima seção.

[4] Plataforma proprietária da Microsoft para desenvolvimento em Java.

Todas as classes de JODI estendem a classe de RMI *UnicastRemoteObject*, cuja função é exportar o objeto remoto instanciado, e implementam sua respectiva interface remota JODI. Como pode ocorrer uma falha de comunicação no momento de instanciar um objeto remoto, todos os métodos públicos de JODI obrigatoriamente devem especificar a exceção *java.rmi.RemoteException* na cláusula *throws* para informar a ocorrência de uma exceção deste tipo. Além disso, todas as classes JODI devem estender a classe *UnicastRemoteObject*.

Enquanto em JDCI todos os objetos, em seu método construtor, devem instanciar um objeto da super-classe *UnicastRemoteObject* (método *super*()) e registrar o objeto instanciado no servidor de nomes gerenciado por RMI (método *rebind*() da classe *Naming*), em JODI isso só ocorre com o servidor *JODIServer*. Considerando que os demais objetos não são acessados diretamente através do servidor de nomes e sim a partir da chamada de métodos[5], não vimos a necessidade de registrar todos os objetos remotos, o que melhora o desempenho e diminui o uso de memória por parte do servidor JODI.

Quanto à re-escrita dos objetos Java/COM, a mesma é realizada completamente pelos métodos construtores (com exceção de *cLevelMember* e *cCell*) e, de forma simples, pode-se caracterizá-la como o processo de busca e re-escrita das propriedades dos objetos de ADO MD e instanciação automática das suas classes subjacentes. Ressalta-se que na assinatura dos métodos construtores dos objetos de JODI aparece como parâmetro o objeto de ADO MD a ser re-escrito e o objeto JODI que o instanciou (no apêndice A são apresentados os códigos de todas as classes e interfaces de JODI).

Com a finalidade de simplificar a tarefa de distribuição e utilização de JODI, ao contrário de JDCI onde, para utilizar o seu servidor, o usuário deve inicializar o servidor de nomes gerenciado por RMI manualmente (comando *rmiregistry*), ao se executar JODI, o servidor RMI é executado automaticamente e o servidor JODI instanciado com o nome *//nome_host/JODIServer*.

O servidor JODI é formado pelos pacotes *MATRIKS.JODI* (conjunto de classes), *MATRIKS.iJODI* (conjunto de interfaces), ADO e ADO MD (pacotes Java/COM). As interfaces e classes JODI devem ser compiladas através do compilador Java da Microsoft[6]

---

[5] Por exemplo, para acessar um objeto de conexão devemos chamar o método *getConnection*() de *JODISever*, para acessar um objeto do tipo *cCube*, devemos chamar o método *getCube*() de *cConnection*, e assim por diante.

[6] Distribuído com o ambiente de desenvolvimento *Visual J++* ou com o kit de desenvolvimento Microsoft SDK for Java, o compilador Java da Microsoft estende as funcionalidades do compilador *javac* da Sun para suportar os pacotes Java/COM (por exemplo, *msadomd*, *msado15*e *COM*).

(jvc.exe). Deve-se observar que o pacote RMI (*rmi*.zip) não é distribuído com o *Visual J++* e deve ser adquirido no site da Microsoft através do endereço <ftp://ftp.microsoft.com/developr/MSDN/UnSup_ed/>. Após compiladas, as classes JODI devem ser re-compiladas através do compilador RMI da Sun (*rmic*), gerando os arquivos *nome_classe_stub* (responsáveis pela comunicação no lado do cliente RMI) e *nome_classe_skeleton* (responsáveis pela comunicação no lado do servidor).

Para permitir que o cliente se conecte com o registro RMI e com o objeto servidor, é necessário fornecer um arquivo de política (*jodi.policy*), que pode ser modificado a fim de permitir apenas conexões de usuários autorizados (HORSTMANN; CORNELL, 2001).

O cliente JODI deve criar e instanciar um servidor de segurança e buscar, no registro de nomes de RMI, o objeto remoto da interface *iJODIServer* – JODIServer.

A seguir veremos qual metodologia foi adotada para realização de testes em JODI.

4.4 Testes

Com o objetivo de detecção e correção de erros nas aplicações JODI e OCCOM, e assegurar que o software esteja de acordo com os requisitos especificados, foram realizadas as seguintes categorias de testes:

- Testes Funcionais (*black-box*)
- Testes Estruturais (*white-box*, *glass-box*)
- Testes de Desempenho (*performance*)

A base de dados utilizada para testes foi a base de dados de exemplo da Microsoft, FoodMart 2000, fornecida com a instalação do Analysis Services.

Para a realização dos testes foi utilizado um microcomputador com a seguinte configuração: processador Athlon XP 2.0+, 256 Mb memória DDR 333Mhz, HD IDE 60Gb 7.200 rpm. Os seguintes programas foram utilizados: sistema operacional Microsoft Windows XP, Microsoft Visual J++ 6.0, Microsoft SQL Server 2000, *OLE DB for OLAP* v. 2.0, ADO 1.5 e Sun Java 2 Standard Edition v. 1.4.0.

Para se ter uma idéia do volume do trabalho realizado, JODI possui aproximadamente 2.500 linhas de código, destas, a implementação das classes *cConnection* (≅ 350 linhas) e *cLevel* (≅ 250 linhas) são as com maior número de linhas e, conforme (PUC/RS, 2003) , é praticamente impossível testar completamente um sistema, garantindo que ele esteja 100% livre de erros.

A descrição detalhada sobre cada um dos testes realizados é o que veremos a seguir.

4.4.1 Testes Funcionais

Baseado na especificação, testes funcionais ou de caixa preta (*black-box*), buscam encontrar falhas, inconsistências ou omissões no projeto, sem analisar a estrutura interna do módulo a ser testado. Considera tanto a especificação como a implementação do projeto. (LISKOV; GUTTAG, 2001).

Inicialmente buscou-se validar as agregações existentes no modelo, realizando uma varredura nas classes nos seguintes sentidos (conforme Figura 4.20):

- iConnection → iCube → iDimension → iHierarchy → iLevel → iLevelMember (child)
- iLevelMember (parent) → iLevel → iHierarchy → iDimension → iCube → iConnection

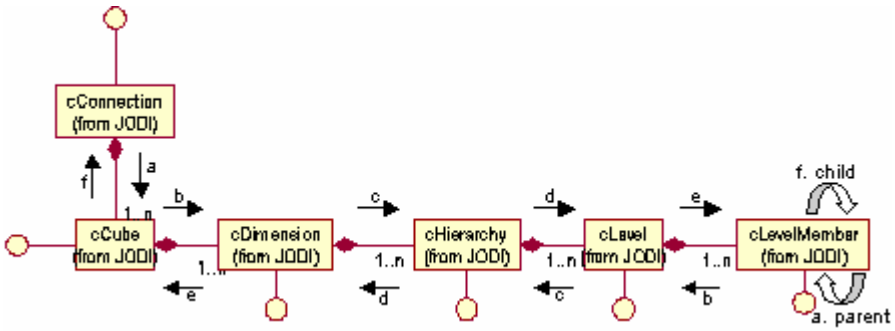


Figura 4.20 – Teste das agregações JODI.

Para validação dos métodos das classes, os resultados dos testes foram comparados aos da ferramentas da Microsoft para gerenciamento do servidor OLAP (*Analysis Manager*) e para realização de consultas MDX (*MDX Sample Application*).

Foram gerados uma série de casos de teste visando explorar caminhos alternativos através da especificação, utilizando a técnica de análise do valor limite, ou seja, usando valores de entrada no seu máximo, logo abaixo do máximo, um valor nominal, logo acima do mínimo e o valor mínimo (MIN, MIN+, NOM, MAX-, MAX).

Como os valores máximos de tamanho de um cubo OLAP não são pré-determinados, estipulamos os valores para realização dos testes considerando que um cubo OLAP possui em média de 4 a 8 dimensões, cada uma com uma hierarquia de 2 a seis níveis.

Verificamos os metadados para um cubo virtual gerado a partir do cubo *Sales* com a dimensão '[Product].[Product Department]', que possui 2 níveis de hierarquia, e para os cubos

Budget, que possui 4 dimensões e média de 3 níveis de hierarquia; e *Sales*, com 12 dimensões e média de 2 níveis de hierarquia.

Para os testes de consultas MDX, foram gerados os seguintes casos:

- Recuperar cubo com 1 dimensão e 1 nível de hierarquia: sub-cubo gerado a partir do cubo *Sales* com a dimensão '[Product].[Product Family]' com 3 células;
- Recuperar cubo com 1 dimensão e 2 níveis de hierarquia: sub-cubo gerado a partir do cubo *Sales* com a dimensão '[Product].[Product Department]' com 23 células;
- Recuperar cubo com 4 dimensões e média de 3 níveis de hierarquia: cubo *Budget* com um total de 14.400 células;
- Cubo com 3 dimensões e média de 4 níveis de hierarquia: sub-cubo gerado a partir do cubo 'Sales' com as dimensões '[Product].[Product Name] x [Time].[1997] x [Store].[Store Name]' com 156.000 células;
- Recuperar cubo com 2 dimensões e média de 3 níveis de hierarquia: sub-cubo gerado a partir do cubo *Sales* com as dimensões '[Product].[Product Department] x [Customers].[Name]' com 236.463 células;
- Cubo com 3 dimensões e média de 4 níveis de hierarquia: sub-cubo gerado a partir do cubo Sales com as dimensões '[Product].[Product Name] x [Time].[Month] x [Store].[Store Name]' com 468.000 células. Vale salientar que a ferramenta da Microsoft para realização de consulta OLAP não foi capaz de gerar este cubo.

Os testes acima foram realizados duas vezes consecutivas, sem abandonar a aplicação. O tempo médio para execução das operações serviram como medida de desempenho. Considerações sobre o desempenho de JODI na realização destes testes serão vistas mais adiante.

4.4.2 Testes Estruturais

Baseados no código, testes estruturais, de caixa branca (*white-box*) ou de caixa de vidro (*glass-box*), examinam a estrutura interna do programa. Para a realização dos testes, foram examinados os códigos fonte das rotinas críticas do sistema. Inserimos instruções no código para podermos acompanhar o resultado das operações realizada passo a passo. Consideramos rotinas críticas:

- Inicialização do servidor JODI: responsável pela comunicação com o cliente JODI e instanciação da classe *cConnection*, o código fonte da classe *JODIServer* for analisada diversas vezes de maneira a fornecer um melhor desempenho com um menor custo;
- Rotinas de criação dos objetos JODI: todos os métodos de criação dos objetos JODI foram testados cuidadosamente a fim de não conter erros, já que são os métodos essenciais de todo os sistema;
- Rotina *makeMembers*() em *cLevel*: rotina responsável pela criação dos objetos *cLevelMembers* e que é disparada quando o usuário chama algum dos métodos *getMember*(*parâmetro*) ou *getMembers*(). Uma vez criados, os objetos permanecem instanciados, ou seja, *makeMembers*() só é executada uma vez para cada nível;
- Rotina *makeChildren*() em *cLevelMember*: responsável pela criação do relacionamento pai-filho (*parent-child*) dos objetos de membros do nível, é disparada sempre que o usuário chama o método *getChild*(*parâmetro*) ou *getChildren*().

4.4.3 Testes de Desempenho

Os testes de desempenho (performance) têm como enfoque a medição de parâmetros de eficiência. O seu foco é medir o comportamento do software como função da carga e dos recursos. Em nosso caso, observamos o tempo de resposta para realização das rotinas com processamento mais robusto:

- Conexão ao servidor JODI;
- Conexão a um servidor OLAP;
- Navegação pelos objetos de metadados do cubo;
- Realização de consultas MDX.

Foi durante a realização dos testes de desempenho que se notou a necessidade de não criar os objetos *cLevelMember* e seus relacionamentos to tipo pai-filho no momento da conexão com o servidor OLAP, como acontecia com JODI em suas primeiras versões. Por ser um processo lento e, considerando que o usuários que se conecta ao servidor nem sempre irá percorrer todos os metadados do cubo[7], não há necessidade dele aguardar a criação de objetos que não irá utilizar.

[7] Um usuário pode, por exemplo, se conectar a um servidor OLAP apenas para realizar uma consulta MDX.

Outra modificação importante, realizada após os testes de desempenho, é o fato de JODI não manter os objetos de células do cubo instanciados. Por exemplo, ao se realizar a quarta consulta dos casos de testes descritos nos testes funcionais que, no caso das primeiras versões, instanciava 236.463 células, o sistema levava horas realizando a tarefa até que a memória se esgotasse. Além disso, se notou que o software da Microsoft para realização de consultas MDX[8] realizava a mesma tarefa em segundos. Após as medidas adotadas, JODI passou a apresentar desempenho similar, como demonstra os resultados apresentados a seguir:

- Conexão ao servidor JODI: apesar de depender do desempenho e tráfego da rede, a conexão ao servidor JODI é realizada quase que instantaneamente. Nos testes realizados, o tempo médio de conexão foi inferior a meio segundo;
- Conexão a um servidor OLAP: apesar de ter que instanciar os objetos *cCube*, *cDimension*, *cHierarchy* e *cLevel*, o tempo médio necessário para conexão ao esquema multidimensional *FoodMart 2000*[9] foi também inferior a meio segundo;
- Navegação pelos objetos de metadados do cubo: para este teste foi utilizado o cubo de dados *Sales* do esquema *FoodMart 2000* que contém 12 dimensões com uma média de 2 níveis por hierarquia, com uma quantidade total de membros em torno de 2.000. JODI levou em média 47 segundos para percorrer todos os membros (passo 1 da Figura 4.26). Notou-se que a primeira vez em que a tarefa é realizada, em que JODI deve instanciar os membros, o tempo necessário para percorrer os membros (média de ≅ 59 segundos) é 70% maior que o tempo necessário quando os membros já estão instanciados (média de ≅ 35 segundos);
- Realização de consultas MDX: utilizou-se para este teste os casos de testes descritos na seção de testes funcionais. Os resultados obtidos estão descritos na Figura 4.21. Como mostra os resultados, podemos notar que não há relação entre o número de células e o tempo necessário para realização da consulta MDX, acreditamos estar o mesmo relacionado a fatores como complexidade da consulta e tipo de armazenamento dos dados (ROLAP, MOLAP ou HOLAP – ver seção 2.2).

[8] O *MDX Sample Application* acompanha o *MS Analysis Services* (Servidor OLAP Microsoft distribuído junto com o *MS SQL Server 2000*).

[9] Base de dados para testes fornecida com o *MS Analysis Services*.

Descrição	Tempo (s)
a. Consulta a um cubo MDX com 3 células	0,039
b. Consulta a um cubo MDX com 23 células	0,039
c. Consulta a um cubo MDX com 14.400 células	0,149
d. Consulta a um cubo MDX com 156.000 células	0,602
e. Consulta a um cubo MDX com 236.463 células	13,274
f. Consulta a um cubo MDX com 936.000 células	1,016

Figura 4.21 – Testes de Desempenho – Consultas MDX

Conseguimos, após a realização dos testes de desempenho em JODI, identificar os pontos problemáticos na sua implementação. Após algumas modificações em sua estrutura, os problemas de desempenho apresentados nas tarefas de conexão ao servidor OLAP e processamento de consultas MDX conseguiram ser sanados.

4.5 Conclusão

Como principal fator motivador para o desenvolvimento de JODI e não utilização de JDCI como camada intermediária entre OCCOM (*OLAP Cuboid Cell Outlier Miner*) e *OLE DB for OLAP*, destacamos o fato de JDCI não se preocupar com fatores como desempenho, concorrência e distribuição (FIDALGO, 2000). O tempo de resposta e o uso eficiente da memória são fatores que tiveram que ser observados para que JODI pudesse se constituir em uma base confiável para o desenvolvimento de OCCOM. Conseqüentemente, mudanças tiveram que ser feitas no modelo ODCI e em sua implementação (JODI).

Com os testes realizados, podemos garantir que JODI atende aos requisitos especificados e se constitui em uma plataforma para desenvolvimento OLAP segura, de bom desempenho, e que utiliza recursos computacionais de forma moderada, garantindo a OCCOM e a outras aplicações OLAP uma camada confiável, portável e de código aberto, que disponibiliza e integra serviços OLAP providos pelos servidores *OLE DB for OLAP*.

5 OCCOM

OCCOM (*OLAP Cuboid Cell Outlier Miner*) é um componente de software portável, de código aberto e baseado em tecnologia de baixo custo, que tem como função minerar dados atípicos em cubóides OLAP por meio de análise estatística multidimensional e multi-granular. OCCOM re-aproveita os algoritmos propostos em pesquisas apresentadas na seção 3.1, que servem como uma base teórica sólida para o seu desenvolvimento.

Um dos componentes de software do projeto MATRIKS (Seção 2.5), OCCOM constitui-se na principal lacuna em sua arquitetura (Figura 2.6), e tem como principal papel localizar anomalias em todos os níveis de agregação em cubos de dados OLAP, fornecendo objetos enriquecidos por mineração que auxiliarão usuários a explorarem cubos OLAP de forma eficiente, além de gerar entradas para HYSSOP (gerador de resumos decisórios em linguagem natural – Seção 2.6), ligando-o com *OLE DB for OLAP*.

Para o desenvolvimento de OCCOM, foi adotada a mesma metodologia utilizada em JODI (Capítulo 4). A seguir apresentaremos cada um dos quatro passos do desenvolvimento de OCCOM.

5.1 Análise de Requisitos

A seguir apresentamos uma série de requisitos que OCCOM deve atender a fim de realizar a tarefa proposta:

- OCCOM deve ser desenvolvido em uma arquitetura de software aberta, portável (independente de plataforma) e de baixo custo.
- OCCOM deve implementar os algoritmos de mineração de exceções em cubos OLAP propostos por Sarawagi, Agrawal e Megiddo (1998; Seção 3.1.1) e por Chen (1999; Seção 3.1.2).
- O cliente OCCOM deve ser capaz de escolher qual algoritmo de mineração deseja aplicar: o algoritmo log-linear (mais apropriado quando a medida tem como função de agregação soma ou quantidade) ou linear (mais apropriado quando a medida tem como função de agregação a média dos elementos).

- O cliente OCCOM deve indicar a qual servidor OLAP, esquema multidimensional e cubo de dados será aplicado o algoritmo de mineração. Como faz parte do projeto MATRIKS, OCCOM deverá necessariamente utilizar o servidor JODI para realizar a tarefa de conexão ao servidor OLAP (*OLE DB for OLAP*), recebendo como parâmetro o objeto de cubo de dados (*iCube* – Figura 4.7), que fornecerá informações de medida e dimensões para o cálculo de exceções.
- Uma vez escolhido o cubo de dados, o cliente OCCOM deverá informar qual medida será utilizada e em quais dimensões (e até que nível em cada dimensão) deseja procurar por exceções.
- O grau de sensibilidade de OCCOM a valores extremos (*outliers*) deverá ser indicado pelo cliente, com os valores mínimo de 0% e máximo de 30%.
- Após o processamento, OCCOM deverá retornar o grau de exceção encontrado em cada uma das células do cubo no seguinte formato: 0 – sem exceção; 1 – baixo grau de exceção; 2 – médio grau de exceção; 3 – alto grau de exceção. A seguir veremos como esses valores devem ser definidos.

  Diferentemente dos algoritmos de mineração vistos, onde é estabelecido um limiar que indica se uma célula é ou não exceção[1], OCCOM deverá retornar o grau de exceção com base nos seguintes valores:

  - se o valor de exceção encontrado for menor ou igual a 1.64 (limiar correspondente a 90% em uma distribuição normal), a célula não possui exceção (grau 0);
  - se o valor encontrado for maior que 1.64 e menor ou igual a 1.96 (correspondente a 95% em uma distribuição normal), a célula possui um grau baixo de exceção (grau 1);
  - se o valor for maior que 1.96 e menor ou igual a 2.58 (correspondente a 99% em uma distribuição normal), a célula possui um grau de exceção médio (grau 2); e
  - se o valor encontrado for superior a 2.58, a célula possui um alto grau de exceção (grau 3).

  Estes valores foram encontrados com base na tabela Z de distribuição normal (SOARES; FARIAS; CESAR, 1991).
- Para assegurar um melhor desempenho no cálculo de exceções, OCCOM deve implementar o método de re-escrita introduzido por Sarawagi, Agrawal e Megiddo (1998).

[1] Sarawagi, Agrawal e Megiddo definem um limiar de 2.5, que corresponde a uma probabilidade de 99% em uma distribuição normal (na verdade, este valor corresponde a uma probabilidade de 98.758%). Chen trabalha com um limiar de 2.57, uma probabilidade de 99% que uma célula seja uma exceção.

- Além das quatro medidas de exceção sugeridas por Chen (1999), OCCOM deve implementar mais duas novas medidas. As seis medidas estão divididas em duas categorias: medidas de exceção das células dos cubos (indicam o grau de exceção da célula e que pode ser encontrado abaixo dela); e medidas de exceção do cubo (indicam o grau de exceção encontrado em alguma célula do cubo ou abaixo dele). A seguir apresentamos a definição de cada uma dessas medidas.

  Medidas de Exceção da Célula

  - *SelfExp*: indica o grau de exceção da célula do cubo. É calculado através da fórmula

$$SelfExp(y_c) = \max\left(\frac{|y_c - \hat{y}_c|}{\sigma_c} - \tau, 0\right) \text{ onde}$$

$$s_c = \frac{|y_c - \hat{y}_c|}{\sigma_c} \text{ é o resíduo padronizado e } \tau \text{ é o limiar.}$$

    Note que este valor deve ser traduzido em números inteiros entre 0 (não é exceção) e 3 (alto grau de exceção), conforme visto anteriormente. Corresponde às medidas *SelfExp* de Sarawagi et al. e *CellExp* de Chen.

  - *UnderExp*: indica o grau de exceção que pode ser encontrado abaixo de cada célula caso realizemos uma operação de *drill-down* (Seção 2.2).

$$UnderExp(y_c) = SelfExp(y_x), \text{ onde } SelfExp(y_x) = \max_{c' \in desc(c)} SelfExp(y_{c'}).$$

    Corresponde às medidas *InExp* de Sarawagi et al. e *UnderExp* de Chen, mas com a seguinte modificação: já que *PathExp* já traz o grau de exceção encontrado abaixo da célula ao longo das hierarquias, achamos mais interessante que *UnderExp* retornasse o grau de exceção encontrado apenas um nível abaixo da célula.

  - *PathExp*: indica o grau de exceção que pode ser encontrado abaixo de cada célula caso realizemos uma operação de *drill-down* ao longo de determinada dimensão.

$$PathExp(y_c, d) = SelfExp(y_x), \text{ onde } SelfExp(y_x) = \max_{c' \in desc(c)} SelfExp(y_{c'})$$

    e $c' \in$ cubo resultante de *drill-down* ao longo da dimensão *d*.

    Corresponde às medidas *PathExp* de Sarawagi et al. e *DimExpForCell* de Chen.

    Note que para o cubóide base, os valores de *UnderExp* e *PathExp* são iguais a 0 (zero), já que não possui descendentes.

Medidas de Exceção do Cubo

- *MaxSelfExp*: indica o maior grau de exceção encontrado em alguma célula do cubo. É uma nova medida, não possuindo correspondentes nos modelos apresentados na Seção 3.1.
- *UnderExp*: indica o grau de exceção encontrado em alguma célula de algum cubo um nível abaixo, resultante de uma operação de *drill-down*. Também é uma nova medida.
- *PathExp*: indica o grau de exceção encontrado em alguma célula de algum cubo resultante de uma operação de *drill-down* ao longo de determinada dimensão. Corresponde à medida *DimExp* de Chen.

Da mesma forma que na célula, para o cubóide base, os valores de *UnderExp* e *PathExp* são sempre iguais a zero.

O diagrama de seqüência mostrando os passos necessários para a utilização de OCCOM pode ser visto na Figura 5.1.

Inicialmente o cliente deve abrir uma conexão com o servidor JODI e utilizá-lo para conectar-se ao servidor OLAP desejado. O cliente deve então escolher e buscar um objeto de cubo de dados (*iCube*), sobre o qual irá aplicar o algoritmo de mineração. Esse objeto deve então ser repassado a OCCOM.

Os objetos de metadados multidimensionais de JODI (*iDimension, iHierarchy, iLevel* e *iLevelMember*) deverão ser consultados para que o cliente possa informar a OCCOM qual a medida utilizada para o cálculo de exceções, e quais dimensões e em até que nível formarão o cubóide base. Por exemplo, o cubo de dados *Sales* da base de dados para testes da Microsoft *FoodMart 2000* possui 12 dimensões com uma quantidade muito grande de membros, impossível de serem manipulados por OCCOM ou JODI; o usuário deve então escolher com quais dimensões deseja trabalhar (Produtos, Tempo, Estado Civil, Sexo, ...) e em até que nível (a dimensão Produtos possui os seguintes níveis de hierarquia: Família de Produtos → Departamento → Categoria → Sub-categoria → Tipo → Nome do Produto; o usuário pode escolher trabalhar até o nível Categoria). Este processo é feito adicionando-se dimensões para cálculo em OCCOM.

Opcionalmente o cliente poderá indicar qual o algoritmo utilizado para cálculo de exceções (modelo linear ou log-linear) e qual o nível de sensibilidade a desvios (*outliers*) de OCCOM (mínimo de 0% e máximo de 30%).

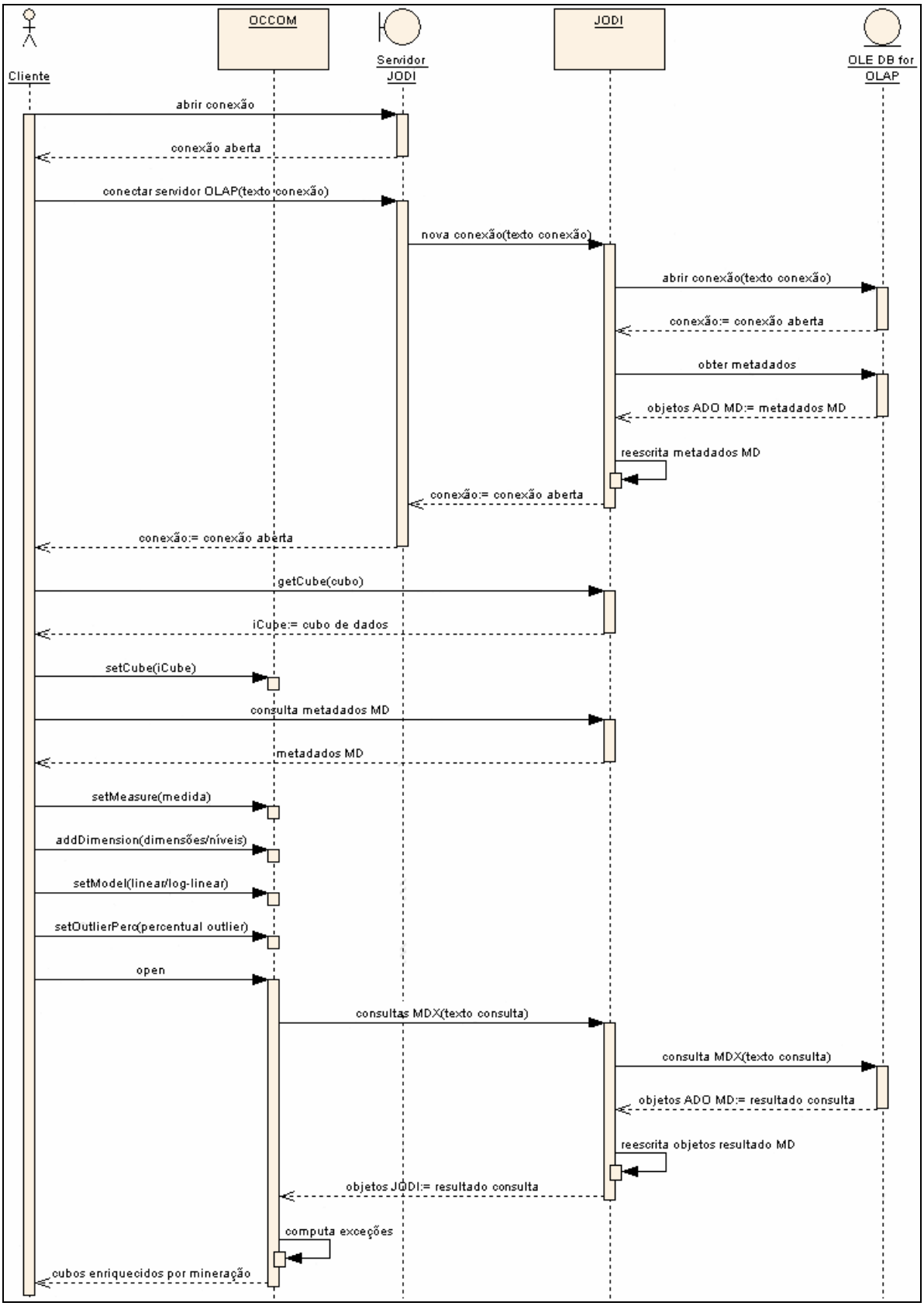


Figura 5.1 – Diagrama de Seqüência – OCCOM.

Em seguida o minerador de exceções deverá ser aberto (*open*), os cubóides de dados serão gerados para o cálculo de exceções através da realização de consultas MDX a JODI, e as

exceções serão computadas. A quantidade de cubos gerados pode ser calculada através da fórmula $\prod_{r=1}^{d}(m_r + 1)$, onde m_r é a quantidade de níveis em cada uma das dimensões. Por exemplo, para um cubo formado com as dimensões Produto (até o nível Categoria: Família → Departamento → Categoria), Tempo (até o nível Mês: Ano→ Trimestre → Mês), Estado Civil (1 nível) e Sexo (1 nível), seriam gerados $\prod_{r=1}^{4}(m_r + 1)$ membros, ou seja (3+1)x(3+1)x(1+1)x(1+1) = 64 cubóides.

OCCOM deverá contar ainda com operações como *drill-downi* e *roll-up* para que o cliente possa navegar entre cada um dos cubos gerados.

5.2 Modelo

> *Modelar software orientado a objetos é uma tarefa difícil, e modelar software orientada a objetos re-usável é ainda mais difícil. Você deve encontrar muitos objetos pertinentes, colocá-los em classes com a granularidade correta, definir interfaces para as classes e hierarquias, e estabelecer relacionamentos chave entre elas. O seu projeto deve ser específico para o problema em questão, mas geral o suficiente para prever problemas e requisitos futuros. Você também quer evitar a remodelagem, ou pelo menos minimizá-la. Projetistas experientes em orientação a objetos irão dizer a você que um modelo re-usável e flexível é difícil, se não impossível de estar "correto" na primeira tentativa. Antes que um modelo esteja acabado, eles usualmente tentarão re-usá-lo várias vezes, modificando-o cada vez.* (GAMMA, 1995)

OCCOM é um software orientado a objetos, que tem por objetivo identificar células excepcionais em cubos OLAP. Uma célula excepcional é aquela que possuir um valor estimado significantemente diferente do seu valor real, calculado com base em modelos estatísticos previamente testados, e que levam em consideração o contexto em que a célula está inserida (dimensões e níveis de granularidade).

Do modelo proposto para OCCOM, apresentado na Figura 5.2, destacamos as seguintes características principais:

- Todos os cubóides que podem ser gerados a partir do cubo base estão representados em OCCOM através da classe *cMinerCube*, que é composto por células (*cMinerCell*);
- O minerador de exceções (*cMiner*) é responsável por conter todos os parâmetros necessários para o cálculo das exceções (modelo, nível de sensibilidade a desvios, cubo, medida, dimensões e níveis);
- Alguns objetos de JODI são reescritos em OCCOM: medida (que em JODI é representada como um objeto do tipo dimensão), dimensão, níveis das dimensões e seus membros.

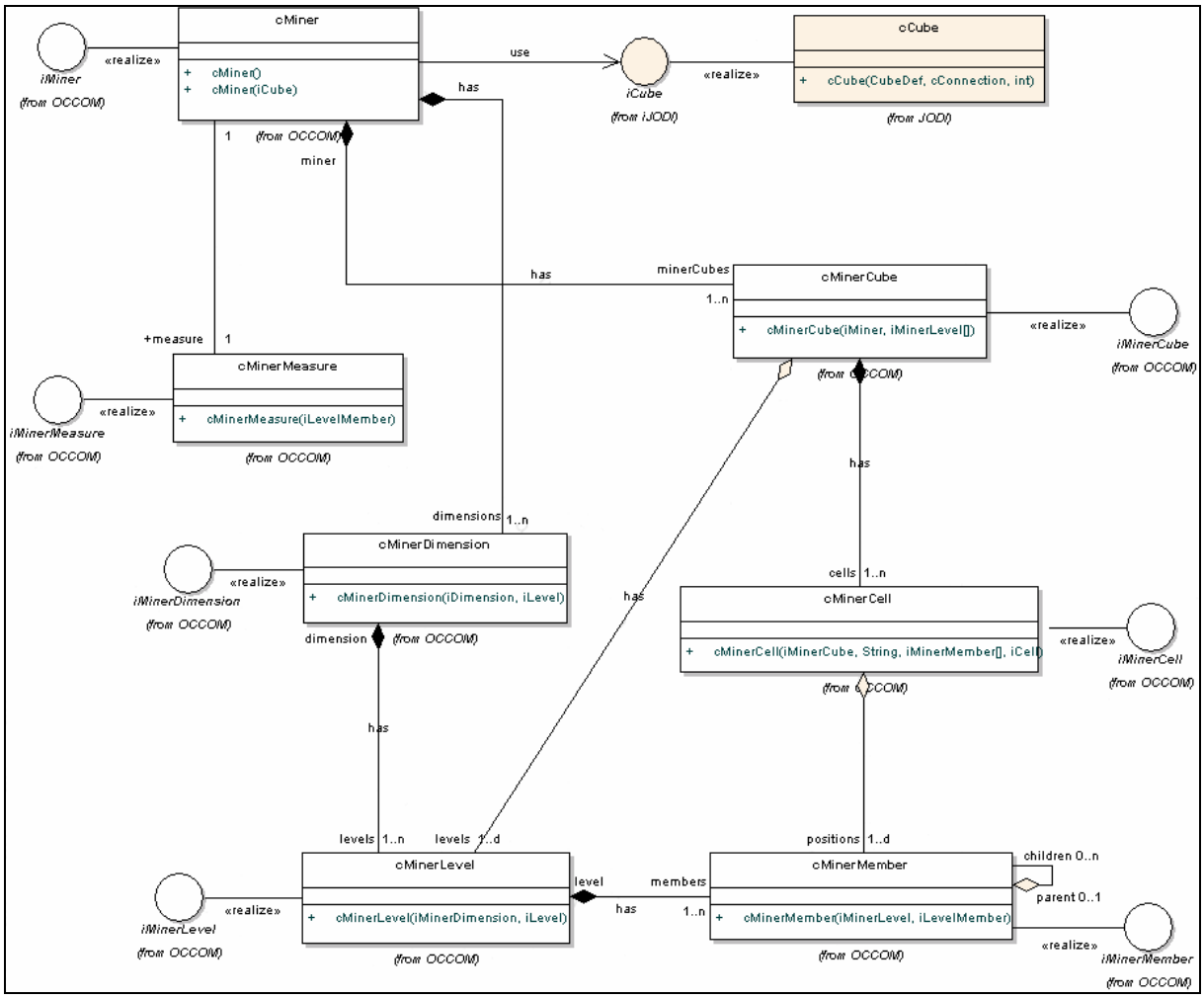


Figura 5.2 – O Modelo OCCOM.

Procuramos tornar o modelo de OCCOM o mais simples possível, já que um modelo mais complexo poderia prejudicar o seu desempenho.

Destacamos que o fato do mesmo reescrever em forma de cache algumas classes de JODI (*cMinerDimension*, *cMinerLevel*, *cMinerMember*), em vez de manter relacionamentos, se deve ao fato de:

- Dimensões em JODI podem possuir diferentes organizações de hierarquias, cada uma com seus níveis – em OCCOM o cliente pode trabalhar apenas com uma hierarquia para cada dimensão e deve selecionar a profundidade máxima que irá buscar por exceções: caso computássemos exceções para todos os níveis da hierarquia, o número de membros poderia ser muito grande, aumentando muito o tempo necessário para o cálculo das exceções em células que o cliente talvez não estivesse interessado em pesquisar.
- OCCOM não necessita manipular todas as informações de metadados de JODI, caso seja necessário, o cliente poderá acessar JODI diretamente, portanto os objetos reescritos possuem um número menor de informações, ocupando um espaço menor na memória.

- A necessidade de OCCOM tratar medidas de forma diferente que JODI – em JODI medidas são dimensões com uma característica especial. OCCOM só admite a definição de apenas uma medida para o cálculo de exceções.
- Por questão de desempenho: caso OCCOM manipulasse objetos JODI diretamente, isso tornaria o cálculo de exceções lento, pois os objetos JODI são remotos e fatores como rede e número de usuários conectados influenciam seu desempenho.

A seguir apresentaremos a descrição detalhada das classes e interfaces do modelo OCCOM.

Na Figura 5.3 podemos ver a classe *cMiner*, que implementa a interface *iMiner*, e é responsável pela inicialização de todos os demais objetos OCCOM (*cMinerMeasure*, *cMinerDimension*, *cMinerLevel*, *cMinerMember*, *cMinerCube* e *cMinerCell*). *CMiner* implementa os métodos da Tabela B.1 presente no apêndice B.

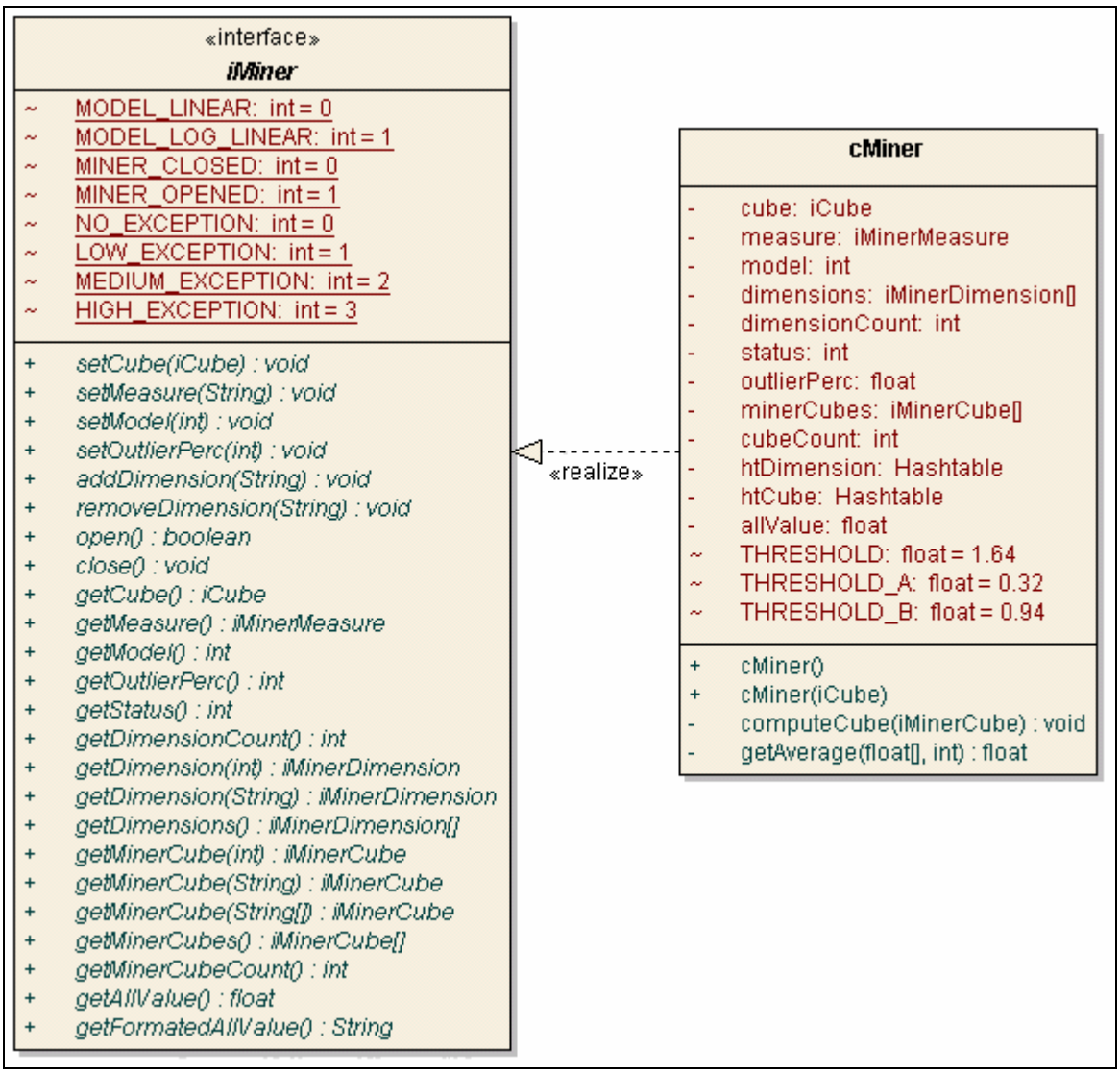


Figura 5.3 – Classe *cMiner* e a sua interface.

Para poder realizar o cálculo de exceções, o usuário deve especificar o cubo que contém os dados originais (objeto *iCube* de JODI) através do método construtor de *cMiner* ou através do método *setCube*. Feito isto, deve determinar qual a medida utilizada para o cálculo (método *setMeasure*), qual o modelo (linear ou log-linear – método *setModel*), qual o grau de sensibilidade de OCCOM a desvios (método *setOutlierPerc*) e adicionar dimensões através do método *addDimension*.

Os métodos privados *computeCube*(*iMinerCube*) e *getAverage*(*float*[]) são responsáveis pelo cálculo das exceções e cálculo da média de acordo com o grau de sensibilidade escolhido, respectivamente. Estes métodos são executados através de chamada ao método *open* de *cMiner*. Uma vez que as exceções tenham sido computadas, os cubos enriquecidos por mineração podem ser acessados através do método *getMinerCube*.

Detalhes sobre o funcionamento de cada um dos métodos citados podem ser vistos no Apêndice B.

A Figura 5.4 apresenta a classe *cMinerMeasure*, que implementa a medida sobre a qual serão computadas as exceções.

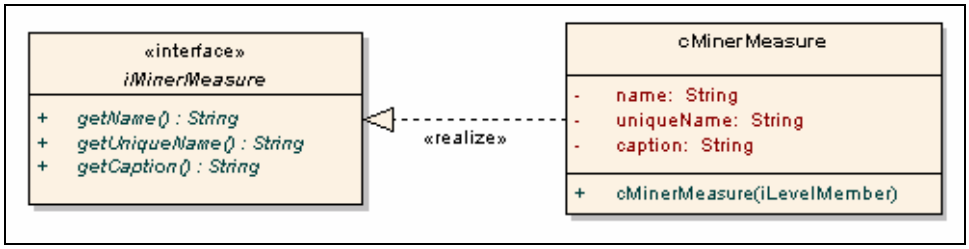


Figura 5.4 – A classe *cMinerMeasure* e a sua interface.

A classe responsável pelas informações de metadados das dimensões que serão utilizadas para o cálculo de exceções é implementada por *cMinerDimension*, conforme Figura 5.5.

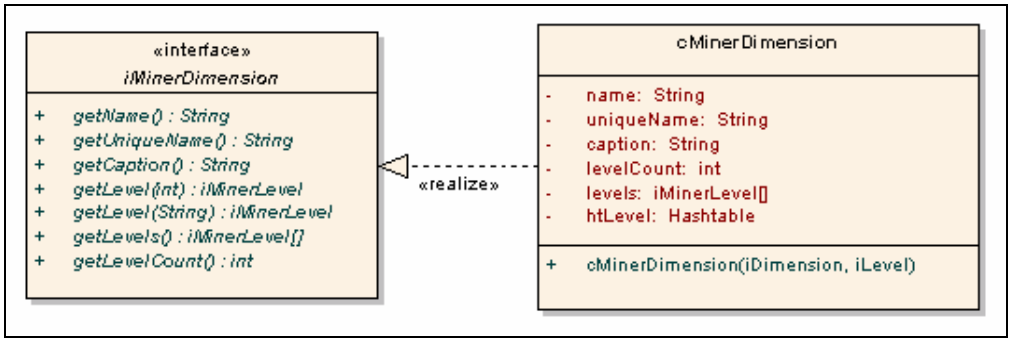


Figura 5.5 – A classe *cMinerDimension* e a sua interface.

Os principais métodos implementados por *cMinerDimension* podem ser vistos na Tabela B.3.

Na Figura 5.6 apresentamos a classe *cMinerLevel*, responsável por inicializar os objetos de membros do nível (*cMinerMember*) e que contém os metadados de níveis de uma dimensão. *cMinerLevel* implementa os métodos da Tabela B.4.

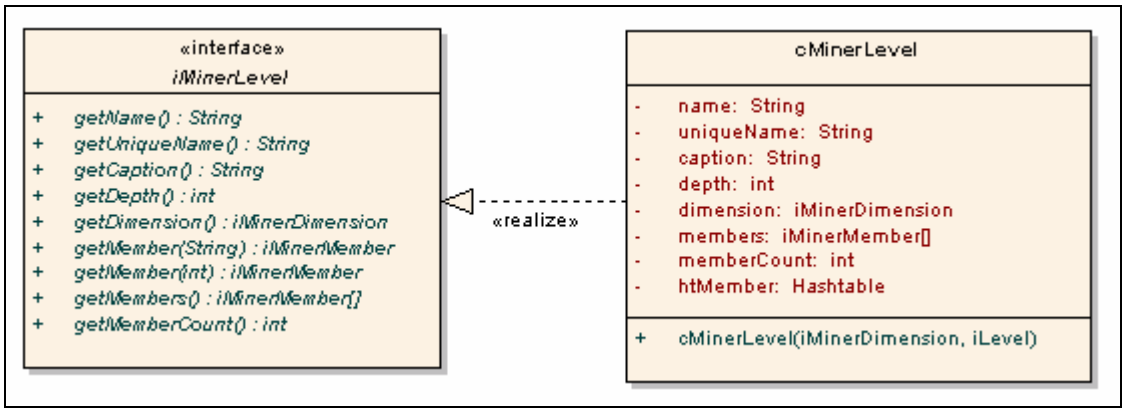


Figura 5.6 – A classe *cMinerLevel* e a sua interface.

A classe *cMinerMember* (Figura 5.7) é responsável pela reescrita de metadados dos membros dos níveis de JODI (*iLevelMember*). Também reescreve os relacionamentos pai-filho dos membros do nível.

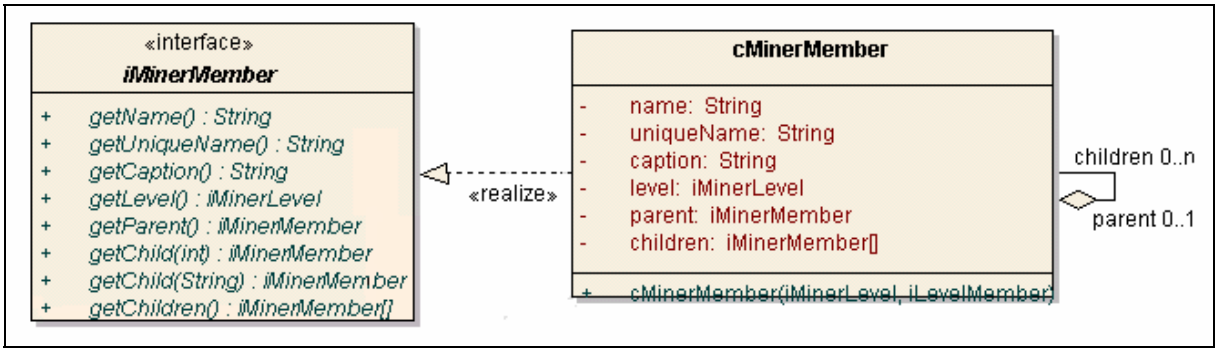


Figura 5.7 – A classe *cMinerMember* e a sua interface.

Os principais métodos implementados pela classe *cMinerMember* podem ser vistos na Tabela B.5.

A classe *cMinerCube* (Figura 5.8) é responsável por instanciar objetos do conjunto de cubóides gerados a partir das dimensões e níveis selecionados para o cálculo de exceções. Como vimos na definição de requisitos, podemos ter $\prod_{r=1}^{d}(m_r + 1)$ cubóides diferentes a partir de um cubo base, pelos quais o usuário poderá navegar em busca de células excepcionais (*cMinerCell* – Figura 5.9).

Diferentemente de JODI em suas consultas multidimensionais, que definem eixos para as células do cubo, em OCCOM, os membros das células não possuem eixos, sendo acessadas pelo seu número seqüencial ou pelo seu rótulo. A tarefa de como os dados serão apresentados fica a cargo da Interface responsável por manipular suas células.

A Tabela B.6 apresenta os principais métodos implementados por *cMinerCube*. Através de seus métodos é possível acessar os níveis (*getLevel*, *getLevels* e *getDimensionCount*) e células (*getCell*, *getCells* e *getCellCount*) que compõem o cubo; a consulta MDX a partir da qual o cubo foi gerado (*getQuery*); o grau de exceção do cubóide (*getSelfExp*), encontrado abaixo do

mesmo (*getUnderExp*) ou em algum caminho específico (*getPathExp*); e os cubóides gerados a partir de operações de d*rill down* (*getDrillDown*) ou de r*oll up* (*getRollUp*) em alguma dimensão.

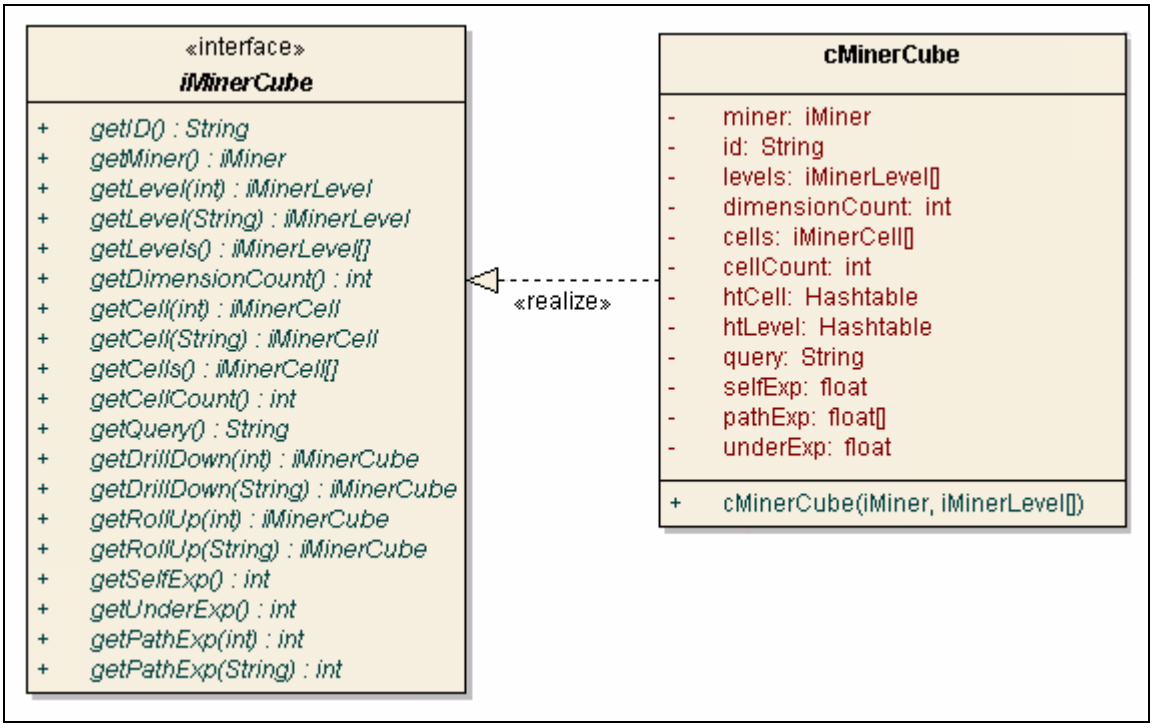


Figura 5.8 – A classe *cMinerCube* e a sua interface.

As células dos cubóides enriquecidos por mineração são geradas pela classe *cMinerCell* (Figura 5.9), responsável por instanciar as células e suas posições em cada dimensão (relacionamento com membros dos níveis – *cMinerLevel*).

As interfaces implementadas por *cMinerCell* podem ser vistas na Tabela B.7. Através delas pode-se acessar o valor da célula (*getValue* e *getFormatedValue*), o seu rótulo (*getName*), as suas posições (*getPosition*, *getPositions* e *getPositionCount*), o seu grau de exceção (*getSelfExp*), o grau de exceção encontrado abaixo dela (*getUnderExp*), o grau de exceção encontrado a partir de uma operação de *drill down* em um caminho específico (*getPathExp*) e verificar se o seu valor é nulo (*getIsNull*).

Na seção seguinte faremos considerações sobre a implementação de OCCOM.

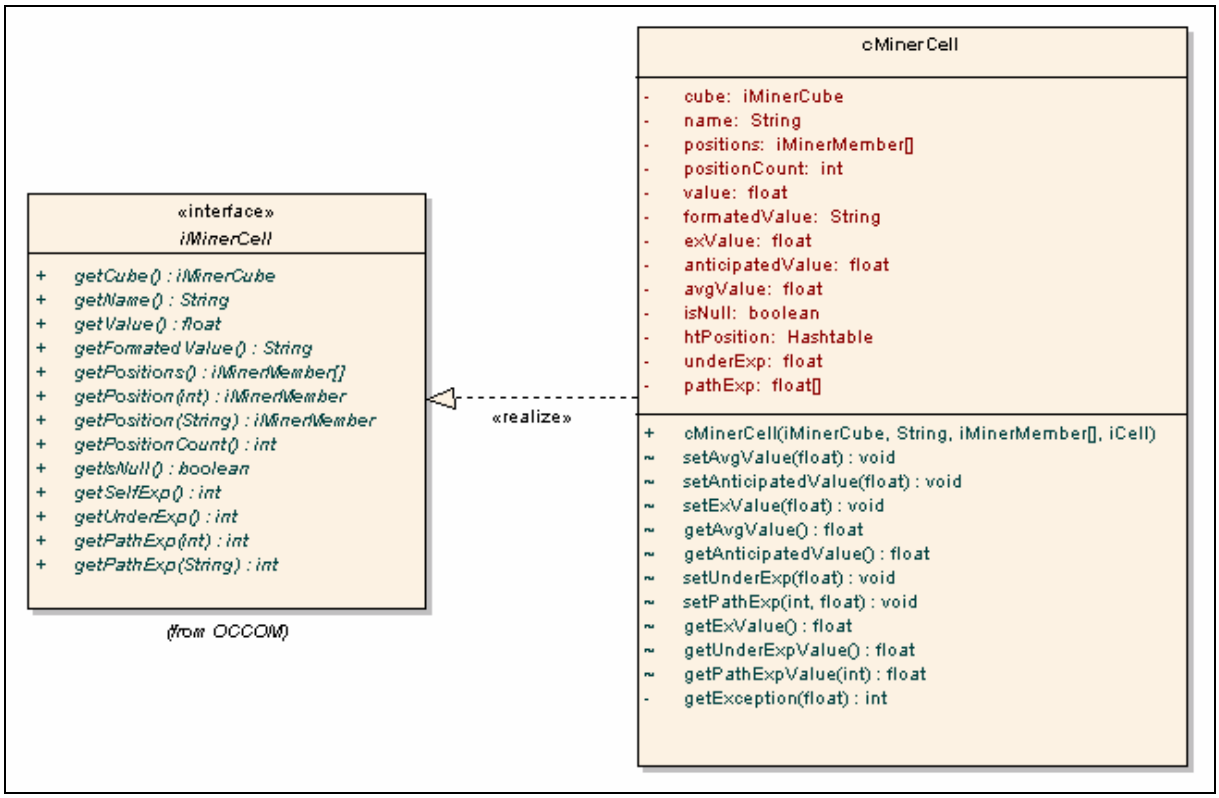


Figura 5.9 – A classe *cMinerCell* e a sua interface.

5.3 Implementação

Com a finalidade de manter os princípios do projeto MATRIKS (Seção 2.5) de desenvolvimento segundo uma arquitetura de software moderna, de baixo custo, aberta e extensível por meio de encapsulamento de serviços em componentes, comunicando-se através de uma interface de software aplicativo (API – *Application Program Interface*); e considerando o fato da escolha de JODI como interface entre OCCOM e *OLE DB for OLAP*, optou-se pelo desenvolvimento de OCCOM na linguagem Java: uma linguagem popular, robusta, portável e de baixo custo.

A arquitetura do minerador de exceções em cubos OLAP OCCOM pode ser vista na Figura 5.10.

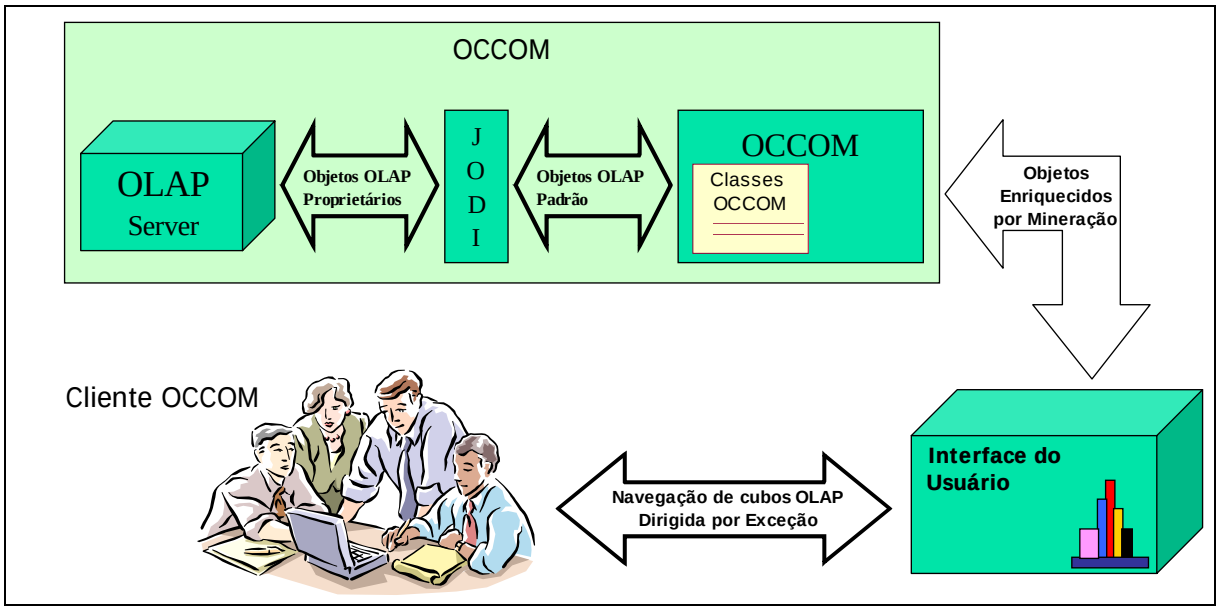


Figura 5.10 – Arquitetura OCCOM.

A função de OCCOM, dentro da arquitetura proposta, é fornecer objetos OLAP enriquecidos por mineração. Para mostrar essas informações de uma forma amigável para o usuário, faz-se necessário o uso de uma interface gráfica. Os dados, como são disponibilizados por OCCOM, são de difícil manipulação por um usuário comum, que não conheça detalhes de sua implementação.

A implementação de OCCOM foi realizada utilizando o pacote JODI para acesso aos dados do servidor OLAP, portanto todas as suas classes importam este pacote. Todas as instruções descritas na Seção 4.3 para a utilização do cliente JODI, valem para OCCOM. O pacote MATRIKS fica então, organizado como apresentado na Figura 5.11.

Para a utilização de OCCOM, além dos arquivos do cliente JODI (interfaces e classes de comunicação com o servidor RMI), são necessários os arquivos de classes e interfaces OCCOM.

O cliente OCCOM deve, então, se conectar ao servidor JODI, selecionar o cubo de dados que deseja trabalhar, e passá-lo como parâmetro para a classe *cMiner* de OCCOM. Um exemplo de código do cliente OCCOM pode ser visto na Figura 5.12.

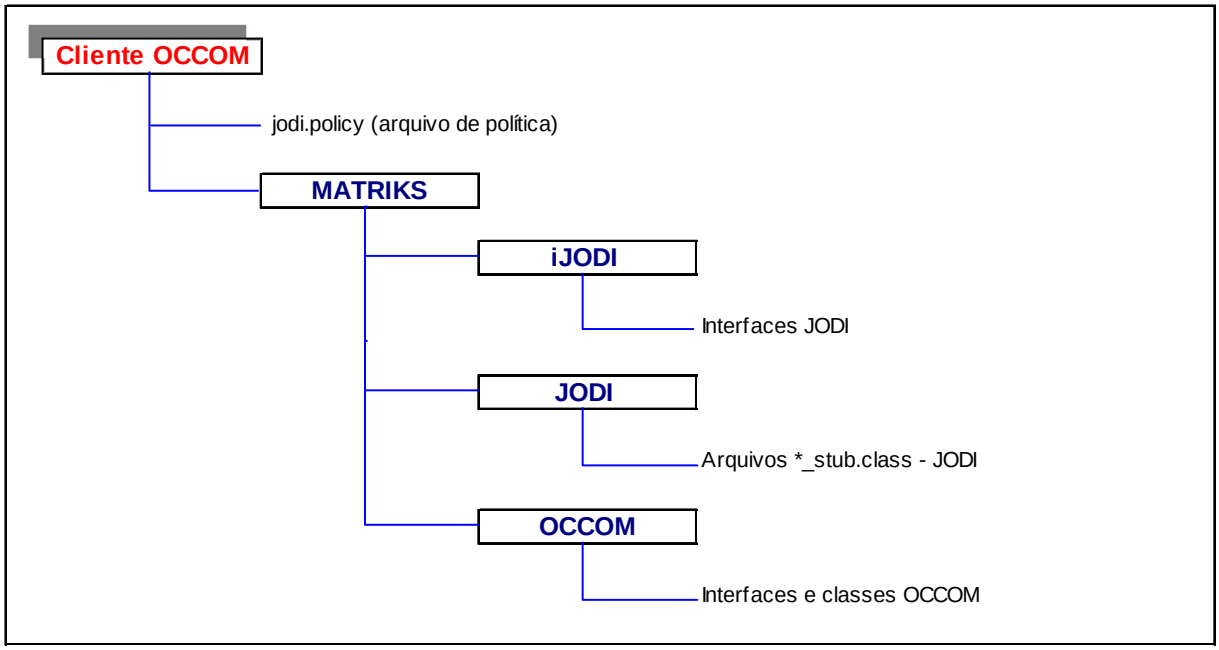


Figura 5.11 – Organização do pacote MATRIKS.

O cliente OCCOM, além de importar os pacotes da interface JODI (MATRIKS.iJODI) e de OCCOM (MATRIKS.OCCOM), deve importar o pacote RMI, necessário para comunicação com o servidor JODI. Feita a conexão com o servidor JODI e com o servidor OLAP, deve-se selecionar o cubo de dados (objeto *iCube* de JODI) sobre o qual será aplicado o algoritmo de mineração. O passo seguinte é informar a medida (nome de um membro da dimensão *[Measures]* do objeto *iCube*) e o cubóide base (quais as dimensões e até que nível em cada dimensão) para o cálculo de exceções. Pode-se ainda selecionar o modelo (linear ou log-linear) e o grau de sensibilidade a desvios que serão utilizados para cálculo. O método *open*() gera todas as agregações possíveis e realiza o cálculo das exceções, permitindo que o cliente acesse cada um dos cubóides gerados através dos métodos *getMinerCube*.

A seguir veremos como o cálculo das medidas de exceção é realizado por OCCOM.

5.3.1 Enquadramento do Modelo

Durante o desenvolvimento de OCCOM, surgiram duas alternativas para implementação do cálculo de exceções:

(1) A primeira alternativa surgiu da tentativa de implementação do cálculo de exceções totalmente através de consultas MDX, infelizmente esbarramos no momento de introduzir hierarquias no modelo, como veremos na seção seguinte.

```
// Importação do pacote RMI
import java.rmi.*;
import java.rmi.registry.*;
import java.rmi.server.*;

// Importação dos pacotes JODI/OCCOM
import MATRIKS.iJODI.*;
import MATRIKS.OCCOM.*;

public class Cliente   {
  public static void main(String[] args)   {
    String serverName = "";
    System.setSecurityManager(new RMISecurityManager());
    try     {
            // Conexão ao Servidor  JODI
            serverName = java.net.InetAddress.getLocalHost().getHostName();
            iJODIServer myServer = (iJODIServer) Naming.lookup("//" + serverName + "/JODIServer");
            iConnection myConnection = myServer.getConnection();
            myConnection.setProviderName("MSOLAP");
            myConnection.setDatasourceName("CIN06");
            myConnection.setCatalogName("FoodMart 2000");
            myConnection.openConnection();

            // Seleção do Cubo de Dados
            iCube cubo = myConnection.getCube("Sales");

            // Utilização de OCCOM
            iMiner myMiner = new cMiner(cubo);                    // Parâmetro: objeto iCube
            myMiner.setMeasure("[Measures].[Unit Sales]");        // Escolha da Medida
            myMiner.addDimension("[Product].[Product Subcategory]");      // Adicionando Dimensões
            myMiner.addDimension("[Gender].[Gender]");
            myMiner.addDimension("[Marital Status].[Marital Status]");
            myMiner.addDimension("[Time].[Quarter]");
            myMiner.setModel(myMiner.MODEL_LOG_LINEAR);           // Escolha do Modelo
            myMiner.setOutlierPerc(25);                           // Nível de sensibilidade a desvios

            // Calcula Exceções e gera cubóides e células
            myMiner.open();
            if (myMiner.getStatus() == myMiner.MINER_OPENED) {
                    // Seleciona cubóide
                    iMinerCube cbo = myMiner.getMinerCube("[Product].[Product Family]+" +
                                            "[Gender].[Gender]+[Time].[Year]");
                    System.out.println("\nCubo: " + cbo);
                    System.out.println("SelfExp Value: " + cbo.getSelfExp());
                    System.out.println("UnderExp Value: " + cbo.getUnderExp());
                    …
            }
    }
    catch (Exception e)  {
            e.printStackTrace();
    }
    …
}
```

Figura 5.12 – Exemplo de código de um cliente OCCOM.

(2) A segunda alternativa, adotada nesta primeira versão de OCCOM, utiliza consultas MDX apenas para gerar as agregações possíveis a partir do cubo base, realizando o cálculo dos valores médios através de chamadas a métodos dos cubóides gerados.

A classe *cMinerCube* implementa os γ-termos da equação do modelo (3.1), correspondentes a todas as agregações possíveis a partir do cubo base.

Para o cálculo dos valores estimados para as células do cubo, utilizamos o método de reescrita, que é definido pela equação

$$\hat{\lambda}_{ijk} = g^1_{ij} + g^2_{ik} + g^3_{jk}\text{, onde}$$

$$g^1_{ij} = avg_k(\lambda_{ijk})$$
$$g^2_{ik} = avg_j(\lambda_{ijk} - g^1_{ij})$$
$$g^3_{jk} = avg_i(\lambda_{ijk} - g^1_{ij} - g^2_{ik})$$

A seguinte seqüência de passos é realizada por OCCOM, após a chamada do seu método *open*(), responsável pelo cálculo de exceções:

(1) Geração do cubo base e de todos os cubóides (*cMinerCube*) que podem ser derivados através de agregações do cubo base;
(2) Criação do cubóide All (agregação em todos os níveis);
(3) Cálculo das exceções para todos os cubóides gerados, iniciando a partir do cubo base;
    (3.1) Cálculo do resíduo e do valor antecipado para as células do cubóide;
    (3.2) Cálculo do desvio padrão para as células do cubóide;
    (3.3) Cálculo de SelfExp;
(4) Cálculo de UnderExp e PathExp.

Para visualização dos passos envolvidos na geração dos cubóides, podemos observar o diagrama de atividades da Figura 5.13. A partir das dimensões e níveis selecionados, são gerados cubóides (de uma dimensão) para cada um dos níveis de cada uma das dimensões. Em seguida, são gerados os cubóides formados pela combinação do nível atual com os cubóides anteriormente gerados. Por exemplo, os cubóides derivados a partir do cubo da Figura 3.4, seriam gerados na seguinte ordem:

- Cubóide de 1 dimensão *[Time].[Semester]*: como não existe cubo gerado a partir de uma dimensão anterior, passa para o próximo nível da dimensão *[Time]*;
- Cubóide de 1 dimensão *[Time].[Quarter]*;
- Cubóide de 1 dimensão *[Time].[Month]*;
- Cubóide da dimensão seguinte: *[Gender].[Gender]*;

- Cubóides gerados a partir da combinação de *[Gender].[Gender]* com cubóides da dimensão anterior: *[Time].[Semester]+[Gender].[Gender]*, *[Time].[Quarter]+[Gender].[Gender]* e *[Time].[Month]+[Gender].[Gender]*;
- Cubóide da dimensão seguinte: *[Marital Status].[Marital Status]*;
- Cubóides gerados a partir da combinação de *[Marital Status].[Marital Status]* com cubóides das dimensões anteriores: *[Time].[Semester]+[Marital Status].[Marital Status]*, ..., *[Time].[Quarter]+[Gender].[Gender]+[Marital Status].[Marital Status]*;
- Cubo base: *[Time].[Month]+[Gender].[Gender]+[Marital Status].[Marital Status]*.

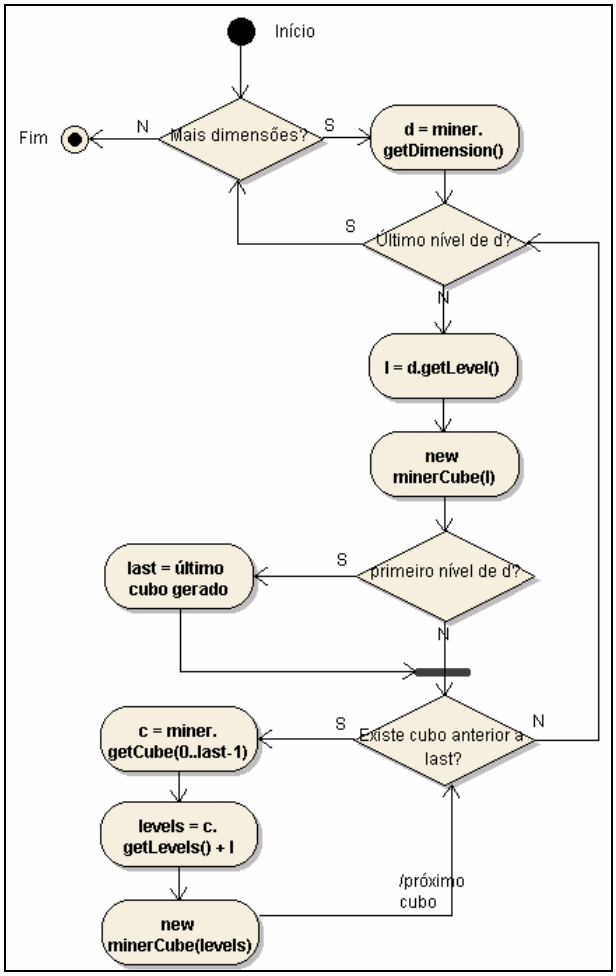


Figura 5.13 – Diagrama de atividades para geração dos cubos.

As consultas MDX para criação dos cubos são geradas a partir dos níveis, igualmente divididos em linhas e colunas, com a utilização da cláusula *CROSSJOIN* , quando necessário:

```
SELECT
    CROSSJOIN({<nome_nível_dim_1>.MEMBERS},
            {<nome_nível_dim_2>.MEMBERS}) ON COLUMNS,
    {<nome_nível_dim_3>.MEMBERS} ON ROWS
FROM <nome_cubo>
WHERE <nome_medida>
```

O cubóide *All*, que possui apenas uma célula, é gerado através da consulta MDX:

```
SELECT  <nome_medida> ON COLUMNS  FROM <nome_cubo>
```

O cálculo de exceções das células dos cubóides é divido em três etapas: cálculo do resíduo e do valor antecipado, cálculo do desvio padrão e cálculo da medida *SelfExp*.

O cálculo do resíduo, operação principal de OCCOM, é realizado através de uma seqüência de passos vistos no diagrama de atividades das Figuras 5.14 e 5.15. Tentaremos explicar cada um dos passos executados:

(1) Caso o modelo empregado seja o log-linear, os logaritmos das células do cubóide são calculados;

(2) A partir do primeiro nível, caso este possua hierarquia, o cálculo de *g* é realizado efetuando-se a média das células agregadas (Figura 5.15). O valor encontrado é então, subtraído do valor original da célula. Por exemplo, para o cubo *[Time].[Month]+[Gender].[Gender]+[Marital Status].[Marital Status]*, *g* é computado pela agregação de *[Time].[Month]* em *[Time].[Quarter]+[Gender].[Gender]+[Marital Status].[Marital Status]*;

(3) O mesmo processo do passo (2) é realizado para a agregação pela remoção da dimensão. No exemplo anterior *[Time].[Month]* em *[Gender].[Gender]+[Marital Status].[Marital Status]*;

(4) Caso o modelo log-linear esteja sendo usado, os expoentes das células são calculados, obtendo-se os valores antecipados.

Após o cálculo do resíduo, devemos calcular os desvios padrões para as células do cubóide através da fórmula 3.5. No entanto, precisamos estimar o valor de *p*, que é feito pela resolução da equação 3.6. Como *p* é calculado iterativamente, procuramos o seu valor entre 0 e 3, igualmente espaçados em 0.3, como proposto por Sarawagi et al. No entanto, obtivemos melhores resultados com um valor mais preciso de *p*. Para isso, no momento em que a equação muda de sinal, efetuamos uma nova busca por *p*, a partir do valor anterior à mudança de sinal, utilizando um intervalo de 0.01, garantindo assim, um melhor resultado. Finalmente, obtemos o valor do desvio padrão e, conseqüentemente, o valor de *SelfExp*.

Podemos agora, realizar o cálculo das outras medidas de exceção do nosso modelo. A partir do cubo base, os valores de *SelfExp* das células do cubo são transmitidos para os níveis imediatamente acima (*UnderExp*), ou para todos os níveis (*PathExp*). Procedimento similar é realizado com relação às medidas de exceção dos cubos. Assim temos, então, todas as medidas computadas.

Considerações sobre o desempenho dos procedimentos mostrados são feitos na Seção 5.4. Futuras versões de OCCOM podem ser desenvolvidas no sentido de aperfeiçoar o processo de computação das medidas, melhorando o seu desempenho.

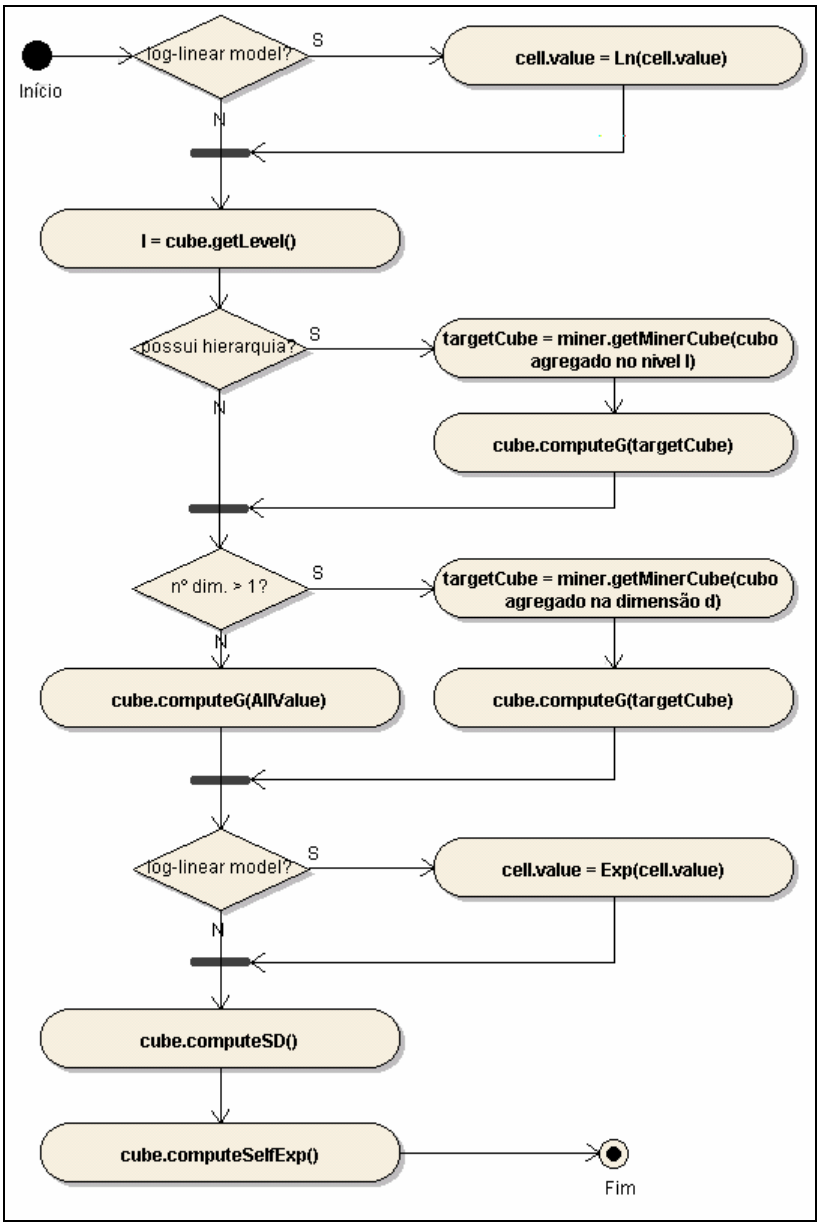


Figura 5.14 – Diagrama de atividades do cálculo de exceções.

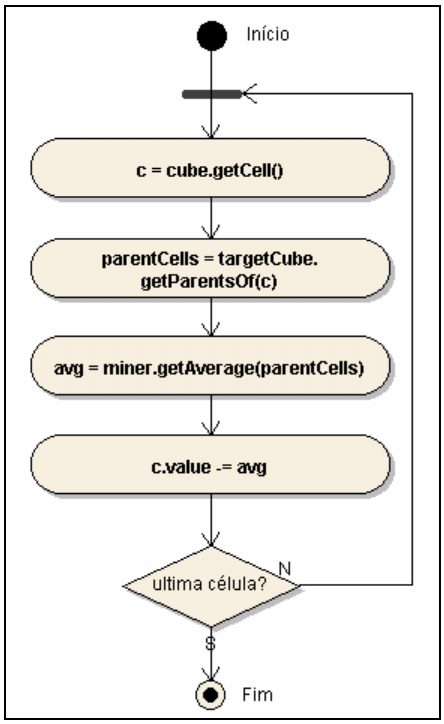


Figura 5.15 – Diagrama de atividades do cálculo dos termos *g*.

5.3.2 Cálculo de **Exceções Utilizando Consultas MDX**

Da tentativa de efetuar o cálculo de exceções em células de cubos OLAP, empregando o modelo de Sarawagi et al., utilizando consultas MDX, surgiu a instrução da Figura 5.16.

A Figura mostra o cálculo para um cubo com 3 dimensões, sem o uso de hierarquias, pelo método tradicional (o método de reescrita também pode ser facilmente implementado). A seqüência dos passos realizados é a seguinte:

- Através da cláusula WITH são geradas as novas medidas: agregações com uma dimensão (cubos *[A]*, *[B]* e *[C]*), agregações com duas dimensões (*[AB]*, *[AC]* e *[BC]*) e o cubo com valores estimados (*[ABC]*);
- Para o cálculo do desvio padrão deve-se computar o valor de *p*, que apesar de estar explicitamente declarado, pode ser calculado através de uma função desenvolvida na linguagem C++, para implementação de uma biblioteca de vínculo dinâmico (*DLL*) que estende a biblioteca de funções da Microsoft (*VBA*) e que deve estar instalada no Servidor JODI;
- Calculamos então o desvio padrão e o valor de *SelfExp* de acordo com o limiar (*[Measures].[Limiar]*) determinado.

Esbarramos no entanto, no problema de não podermos inserir hierarquias com facilidade neste modelo. Não permitindo o seu uso no trabalho proposto.

O cálculo de exceções através do uso de consultas MDX, além de simplificar o desenvolvimento de nossa ferramenta, representaria um ganho de desempenho em sua implementação.

```
WITH
  Member [Measures].[All] as
     'VBA!LOG(Avg(CrossJoin({[Product].[Product Family].Members},
      CrossJoin({[Marital Status].[Marital Status].Members},
                {[Gender].[Gender].Members})), [Measures].[Unit Sales]))'
  Member [Measures].[A] as
    'VBA!LOG(Avg(CrossJoin({[Marital Status].[Marital Status].Members},
                {[Gender].[Gender].Members}), [Measures].[Unit Sales])) - [Measures].[All]'
  Member [Measures].[B] as
    'VBA!LOG(Avg(CrossJoin({[Marital Status].[Marital Status].Members},
                {[Product].[Product Family].Members}), [Measures].[Unit Sales])) - [Measures].[All]'
  Member [Measures].[C] as
    'VBA!LOG(Avg(CrossJoin({[Gender].[Gender].Members}, {[Product].[Product Family].Members}),
                [Measures].[Unit Sales])) - [Measures].[All]'
  Member [Measures].[AB] as
    'VBA!LOG(Avg({[Marital Status].[Marital Status].Members}, [Measures].[Unit Sales])) –
                [Measures].[All] - [Measures].[A] - [Measures].[B]'
  Member [Measures].[AC] as
    'VBA!LOG(Avg({[Gender].[Gender].Members}, [Measures].[Unit Sales])) –
                [Measures].[All] - [Measures].[A] - [Measures].[C]'
  Member [Measures].[BC] as
    'VBA!LOG(Avg({[Product].[Product Family].Members}, [Measures].[Unit Sales])) –
                [Measures].[All] - [Measures].[B] - [Measures].[C]'
  Member [Measures].[ABC] as
    'VBA!EXP([Measures].[All] + [Measures].[A] + [Measures].[B] + [Measures].[C] +
                [Measures].[AB] + [Measures].[AC] + [Measures].[BC])'
  Member [Measures].[p] as '1.06'
  Member [Measures].[Limiar] as '1.64'
  Member [Measures].[Desvio Padrao] as
    'VBA!SQR([Measures].[ABC] ^ [Measures].[p])'
  Member [Measures].[CellExp] as
    'IIF(VBA!ABS([Measures].[Dif] / [Measures].[Desvio Padrao]) > [Measures].[Limiar],
                VBA!ABS([Measures].[Dif] / [Measures].[Desvio Padrao]) - [Measures].[Limiar], 0)',
                FORMAT_STRING = '0.00'
SELECT
  {[Product].[Product Family].Members} ON COLUMNS,
  CROSSJOIN ({[Gender].[Gender].Members}, {[Marital Status].[Marital Status].Members}) ON ROWS
FROM [Sales]
WHERE ([Measures].[CellExp])
```

Figura 5.16 – Cálculo de exceções utilizando consultas MDX.

5.4 Testes

A arquitetura de software orientada a objetos resulta em uma série de camadas de subsistemas que encapsulam a colaboração entre classes. Cada um destes elementos (subsistemas e

classes) executam funções que ajudam a alcançar os requisitos do sistema. É necessário testar sistemas orientados a objetos em uma variedade de níveis diferentes em um esforço para descobrir erros que podem ocorrer quando classes colaboram entre si e subsistemas se comunicam através de camadas da arquitetura (PRESSMAN, 2001).

A mesma metodologia de testes aplicadas a JODI (Seção 4.4) foi aplicada a OCCOM, ou seja, foram realizados testes funcionais, estruturais e de desempenho.

A base de dados utilizada para testes foi a base de dados de exemplo da Microsoft, FoodMart 2000, fornecida com a instalação do Analysis Services. O equipamento utilizado foi o mesmo, com as mesmas configurações, onde foram realizados os testes em JODI.

OCCOM possui aproximadamente 2.000 linhas de código, destas, a implementação das classes *cMiner* (≅ 800 linhas) e *cMinerCube* (≅ 450 linhas) são as com maior número de linhas.

A descrição detalhada sobre cada um dos testes realizados em OCCOM é o que veremos a seguir.

5.4.1 Teste Funcionais

A fim de validar as agregações e relacionamentos existentes no modelo, foram realizadas varreduras nas classes de OCCOM, nos seguintes sentidos (Figura 5.17):

- iMiner → iMinerDimension → iMinerLevel → iMinerMember
- iMinerMember → iMinerLevel → iMinerDimension → iMiner
- iMiner → iMinerCube → iMinerCell → iMinerMember → iMinerLevel → → iMinerDimension
- iMinerCell → iMinerCube → iMiner
- iMiner → iMinerCube → iMinerLevel → iMinerDimension

Para validação do cálculo de exceções, os resultados de duas metodologias diferentes foram comparados: primeiro efetuamos o cálculo utilizando o método *Up-Down* (Seção 3.1.1) e salvamos os resultados encontrados em um arquivo de texto; posteriormente utilizamos o método de reescrita, cujos resultados foram comparados aos valores lidos do arquivo gravado.

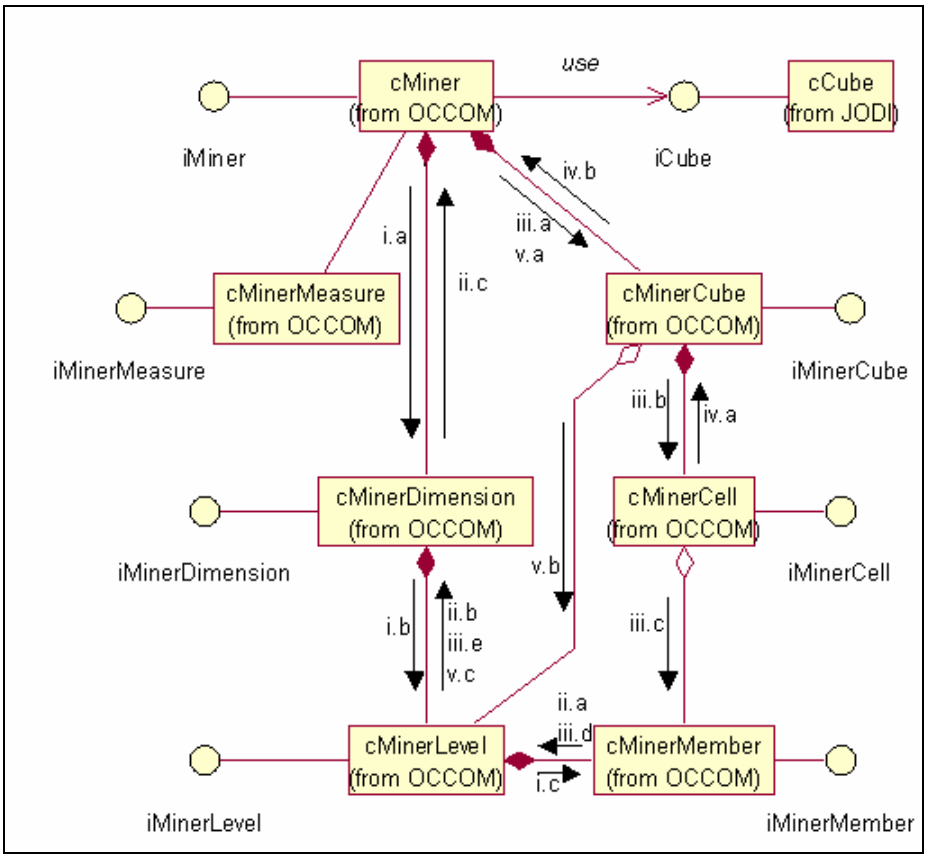


Figura 5.17 – Teste das agregações OCCOM.

5.4.2 Testes Estruturais

A estrutura interna das rotinas críticas do sistema foram examinadas em busca de erros. Consideramos rotinas críticas de OCCOM:

- Criação de cubóides: verificação se todos os cubóides gerados a partir do cubo base foram criados corretamente e verificação da comunicação com JODI para realização das consultas MDX;
- Cálculo de *All Value*: verificação do cálculo do valor do maior nível de agregação (cubo *All*), que é feito através de chamada a uma consulta MDX em JODI;
- Cálculo de exceções: verificação da rotina de cálculo de exceções, que envolve as subrotinas de cálculo do resíduo (que inclui o cálculo do valor médio das células com base no grau de sensibilidade escolhido), cálculo do valor antecipado, cálculo do desvio padrão e cálculo de *SelfExp* (exceção da célula);
- Cálculo de *UnderExp* e *PathExp*: verificação do cálculo das medidas de exceção derivadas de *SelfExp*.

5.4.3 Testes de Desempenho

Visando analisar a performance de OCCOM, criamos uma classe para guardar o tempo necessário para execução das rotinas mais robustas. A classe *Temporizador* computa o tempo gasto na execução das seguintes rotinas:

- Criação dos cubóides e cálculo de exceções (chamada ao método *open* de OCCOM);
- Comunicação com JODI (realização de consultas MDX);
- Cálculo de *All Value*;
- Cálculo de exceções;
- Cálculo dos resíduos (parte do cálculo de exceções);
- Cálculo dos valores médios das células (parte do cálculo de exceções);
- Método privado de busca das células filho que formam determinada agregação (utilizadas para o cálculo dos valores médios das células);
- Cálculo da média com base no percentual de sensibilidade a desvios (utilizadas para o cálculo dos valores médios das células);
- Cálculo do valor antecipado (parte do cálculo de exceções);
- Cálculo do desvio padrão (parte do cálculo de exceções);
- Cálculo de *SelfExp* (parte do cálculo de exceções);
- Cálculo de *UnderExp* e de *PathExp*.

Durante a realização dos testes de desempenho de OCCOM notou-se a necessidade de criação dos relacionamentos “pai-filho” para o cache dos membros dos níveis, pois a rotina de busca das células para cálculo do valor médio apresentou um baixo desempenho.

O principal fato a ser considerado dos resultados obtidos com a realização dos testes de desempenho em OCCOM é o da comunicação com JODI representar cerca de 80% do tempo necessário para o cálculo de exceções. O desafio de melhorar o desempenho de OCCOM passa necessariamente pelo aperfeiçoamento da interface JODI. A Figura 5.18 mostra um resumo dos resultados obtidos. Notamos que o tempo necessário para o cálculo de exceções está mais ligado ao número de hierarquias e, conseqüentemente, ao número de agregações possíveis, do que ao número de células do cubo base.

Fatores físicos como estrutura de rede de comunicação, memória física disponível e velocidade do processador influenciam o desempenho de OCCOM. Fatores lógicos como

número de dimensões, quantidade de níveis de hierarquia em cada dimensão e quantidade de células do cubo base também influenciam em seu desempenho. Não podemos, portanto, estabelecer um tempo preciso para o cálculo de exceções.

[Prod. Depart.]x[Quarter]x[Gd]x[MS]	Tempo (s)
Criação de Cubos	**11,016**
.. JODI	8,153
All Value	**0,031**
Cálculo de Exceções	**0,719**
.. Cálculo do Resíduo	0,704
.. Cálculo do Valor Médio	0,704
.. Rotina getParents	0,891
.. Cálculo da Média (outlier)	0,000
.. Cálculo Anticipated Value	0,000
.. Cálculo do Desvio Padrão	0,000
.. Cálculo de SelfExp	0,000
Cálculo de UnderExp	**0,234**
Abrir OCCOM	**12,000**

* Cubo base com 736 células
** Gd = Gender, MS = Marital Status

[Prod. Depart.]x[Month]x[Gd]x[MS]	Tempo (s)
Criação de Cubos	**10,782**
.. JODI	8,328
All Value	**0,015**
Cálculo de Exceções	**0,750**
.. Cálculo do Resíduo	0,719
.. Cálculo do Valor Médio	0,719
.. Rotina getParents	0,938
.. Cálculo da Média (outlier)	0,000
.. Cálculo Anticipated Value	0,000
.. Cálculo do Desvio Padrão	0,000
.. Cálculo de SelfExp	0,000
Cálculo de UnderExp	**0,219**
Abrir OCCOM	**11,776**

* Cubo base com 2.208 células
** Gd = Gender, MS = Marital Status

[Prod. Categ.]x[Quarter]x[Gd]x[MS]	Tempo (s)
Criação de Cubos	**29,781**
.. JODI	24,099
All Value	**0,016**
Cálculo de Exceções	**5,828**
.. Cálculo do Resíduo	5,780
.. Cálculo do Valor Médio	5,780
.. Rotina getParents	7,391
.. Cálculo da Média (outlier)	0,015
.. Cálculo Anticipated Value	0,000
.. Cálculo do Desvio Padrão	0,032
.. Cálculo de SelfExp	0,000
Cálculo de UnderExp	**1,641**
Abrir OCCOM	**37,282**

* Cubo base com 1.760 células
** Gd = Gender, MS = Marital Status

Figura 5.18 – Teste de Desempenho - OCCOM

5.5 Conclusão

O desenvolvimento de OCCOM baseou-se nos requisitos descritos na Seção 5.1. Testes foram realizados a fim de garantir o seu funcionamento a níveis aceitáveis, não podemos, no entanto, garantir que esteja 100% livre de erros e que o seu desempenho não possa ser melhorado. Versões futuras de OCCOM devem ser desenvolvidas a fim de que este venha a se tornar um componente de software maduro e confiável. A necessidade de incorporação de novos requisitos e desenvolvimento de novas funcionalidades também podem surgir no futuro.

É possível também, que no momento de integração com outros componentes do ambiente MATRIKS, a exemplo de HYSSOP, OCCOM venha a sofrer modificações.

6 INTERFACE OCCOM

A modelagem de interfaces está focada em três áreas de concentração: (1) o projeto de interfaces entre componentes de software, (2) o projeto de interfaces entre o software e alguma entidade externa não humana, e (3) o projeto de interfaces entre um humano (usuário) e o computador. Nesta seção o nosso foco é exclusivamente a terceira categoria de interfaces – o projeto de interface do usuário.

O uso de interfaces gráficas do usuário (*graphical user interfaces* - GUIs), com janelas, ícones e mouse eliminaram muitos dos problemas das interfaces. Mas, mesmo no “mundo de janelas” existem muitas interfaces de difícil aprendizado, difícil uso, confusas e em muitos casos, totalmente frustrantes.

Durante o processo de projeto de interfaces, muitas questões devem ser respondidas, a exemplo de: Quem é o usuário? O que o usuário deve saber para poder interagir com o sistema? Como o usuário irá interpretar as informações produzidas pelo sistema? Qual a expectativa do usuário com relação ao sistema? (PRESSMAN, 2001).

Apresentamos a seguir algumas regras importantes para o desenvolvimento de interfaces:

- Definir modos de interação que não forcem o usuário a realizar ações desnecessárias ou indesejáveis;
- A interação deve ser flexível: alguns usuários preferem enviar comandos usando o teclado, outros preferem o mouse;
- Aspectos técnicos internos devem ser mantidos ocultos dos usuários casuais;
- Permitir a interação direta do usuário com os objetos que aparecem na tela: usuários gostam de sentir que estão no controle das operações realizadas;
- Reduzir a necessidade do usuário se lembrar de tarefas realizadas a pouco tempo: quando usuários estão envolvidos em tarefas complexas, a demanda por lembranças de tarefas recentes pode ser significante. A interface deve fornecer “colas” visuais que permitam ao usuário reconhecer ações passadas, em vez de ter que recordá-las;
- Estabelecer valores padrão (*default*) que possam ser entendidos pelo usuário;
- Definir teclas de atalho que sejam intuitivas;
- O layout visual da interface deve ser baseado em uma metáfora do mundo real;
- Permita ao usuário colocar a tarefa corrente em um contexto significativo: muitas interfaces implementam camadas complexas de interação com dezenas de telas. É

importante fornecer indicadores (títulos de janelas, ícones gráficos, código de cores consistentes) que permitam ao usuário conhecer o contexto do trabalho que está sendo realizado.

Buscando desenvolver uma interface dentro das características listadas, e tendo como alvo usuários responsáveis pelo processo de tomada de decisão dentro das organizações, que nem sempre possuem conhecimentos avançados na operação de computadores, procuramos simplificar a interface gráfica de OCCOM, que possui praticamente duas telas:

- A tela inicial, onde o usuário poderá determinar o servidor JODI a que irá se conectar, o servidor OLAP, e setar parâmetros de configuração, além de escolher o cubo de dados e as dimensões sobre as quais aplicará o algoritmo de mineração. As informações de configuração devem ser salvas em um arquivo de inicialização, de maneira que o usuário não tenha que efetuar as configurações todas as vezes que executar a aplicação.
- A tela de navegação nas informações dos cubos OLAP, onde o usuário poderá verificar as exceções encontradas nas células do cubo e realizar operações de *drill-down*, *roll-up* e *pivot*.

Na Figura 6.1 podemos observar a tela inicial de OCCOM. O usuário, após se conectar ao servidor JODI e OLAP, poderá adicionar as dimensões com as quais deseja trabalhar e definir os parâmetros para o cálculo de exceções, que será efetuado ao se pressionar o botão para processamento. Neste caso foi selecionado o cubo *Sales*; a medida *UnitSales*; um percentual de sensibilidade a desvios de 25%; o modelo *log-linear*, já que se trata de uma unidade do tipo *count*; e foram selecionadas as dimensões *Gender*, *Marital Status*, *Product* até o nível *Product Department*, e *Time* até o nível *Month*.

Uma tela no formato da apresentada na Figura 6.2, será então mostrada ao usuário, que poderá navegar pelas dimensões, efetuando operações de *drill-down* e *roll-up*. Nesta tela, retirada da interface Web de OCCOM, podemos notar que a cor da célula mostra o seu grau de exceção (*SelfExp*), e a cor do texto o grau de exceção encontrado imediatamente abaixo da célula (*UnderExp*). A cor do nome da dimensão mostrada na parte superior da tela irá mostrar o valor da medida *PathExp* para aquela dimensão. Na parte inferior da tela é mostrado o identificador do cubo selecionado. O grau de exceção do cubo (*MaxSelfExp*) é mostrado pela cor do cubo e o grau de exceção que pode ser encontrado imediatamente abaixo dele (*UnderExp*), pela cor de sua letra.

No exemplo da Figura 6.2, podemos visualizar que, apesar de nenhuma célula ser excepcional, as células da linha *[Food]+[1997]* possuem exceções em algum nível abaixo, tanto na dimensão *[Time]*, como na dimensão *[Product]*. Caso fosse realizada uma operação

de *drill-down* nestes dois caminhos, a interface iria mostrar então, o cubo da Figura 6.3, que possui algumas células excepcionais (linha *[Beverages]+[Q4]*).

Note que poderemos ainda encontrar exceções abaixo das células das linhas *[Alcoholic Beverages]+[Q1]* e *[Alcoholic Beverages]+[Q1]*, levando o usuário a realizar uma nova operação de *drill-down* na dimensão *[Time]*, levando ao cubo base *[Product].[Product Department]+[Time].[Month]+ [Gender] +[Marital Status]*, que possui algumas células excepcionais de nível baixo (verde) e médio (azul).

O usuário poderá ainda, visualizar o comando MDX utilizado para a geração do cubo e realizar novo processamento, com outras dimensões, retornando à tela inicial.

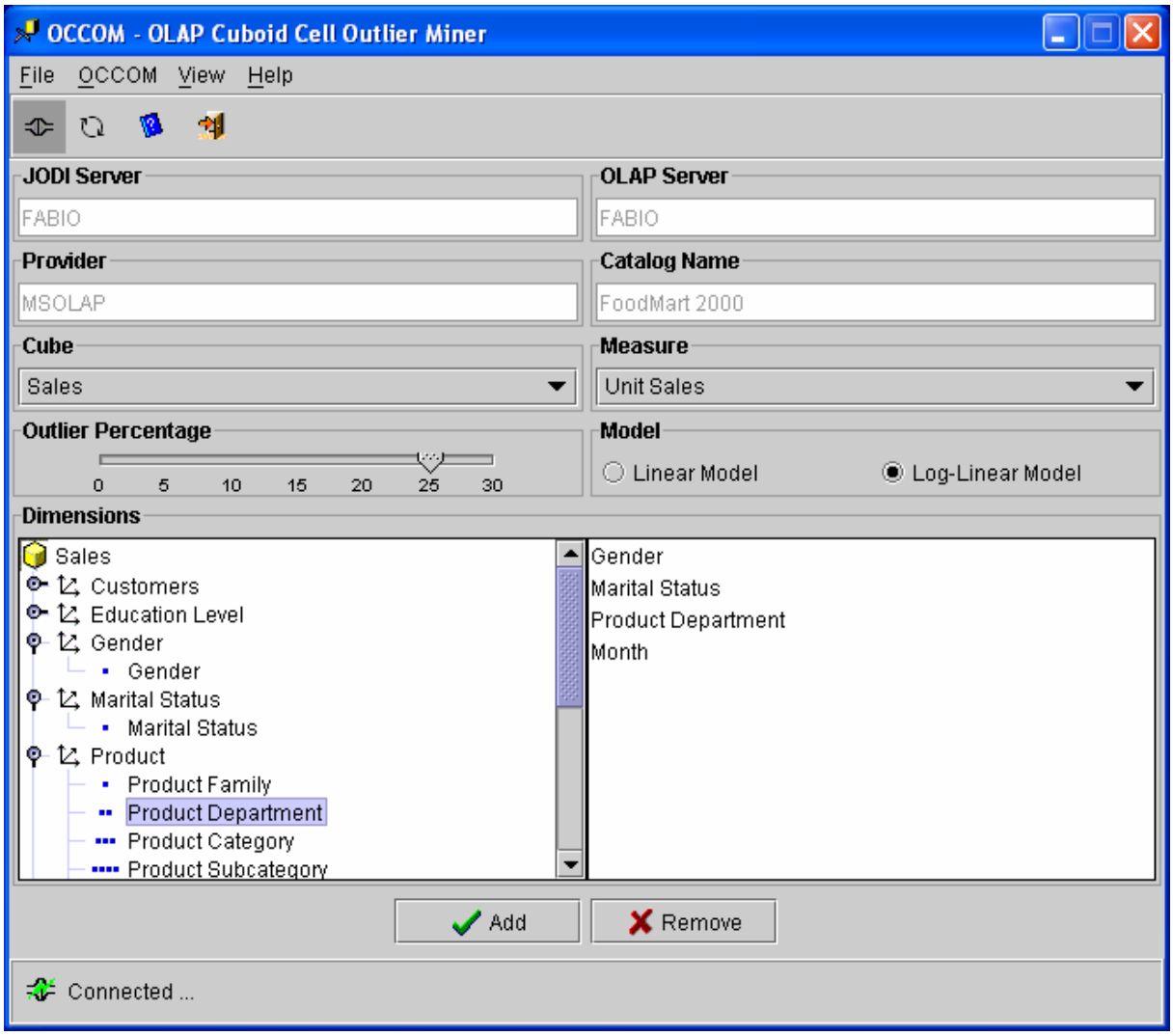
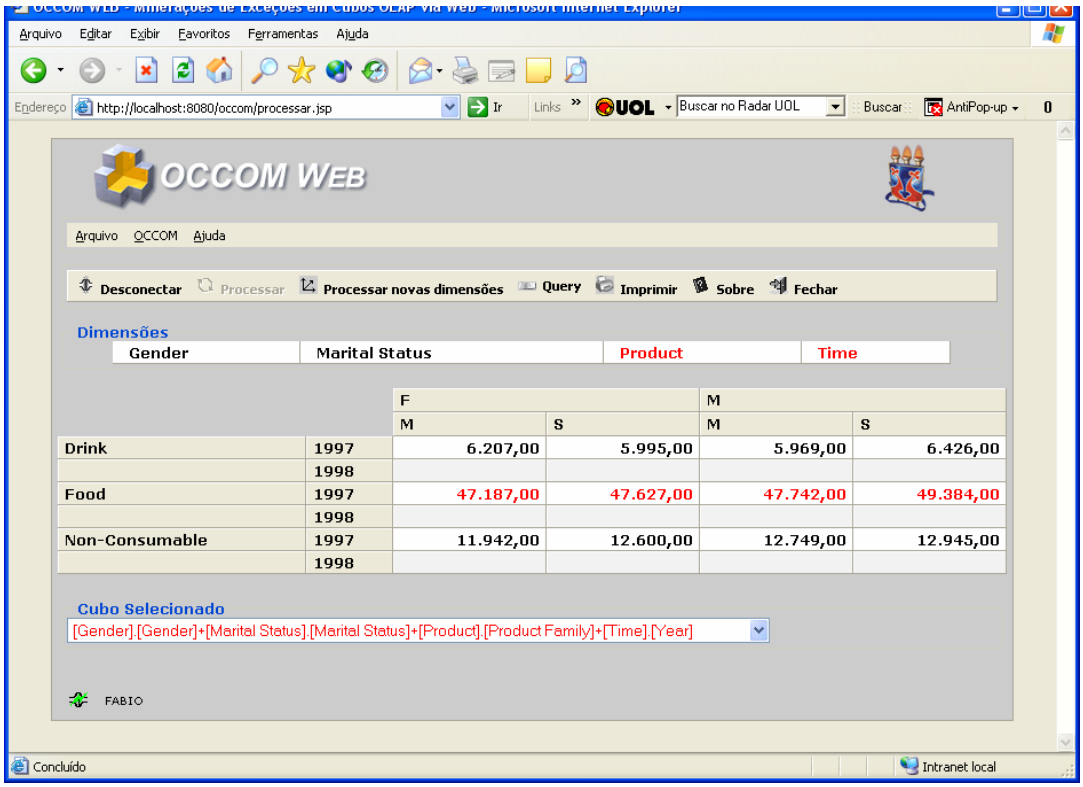
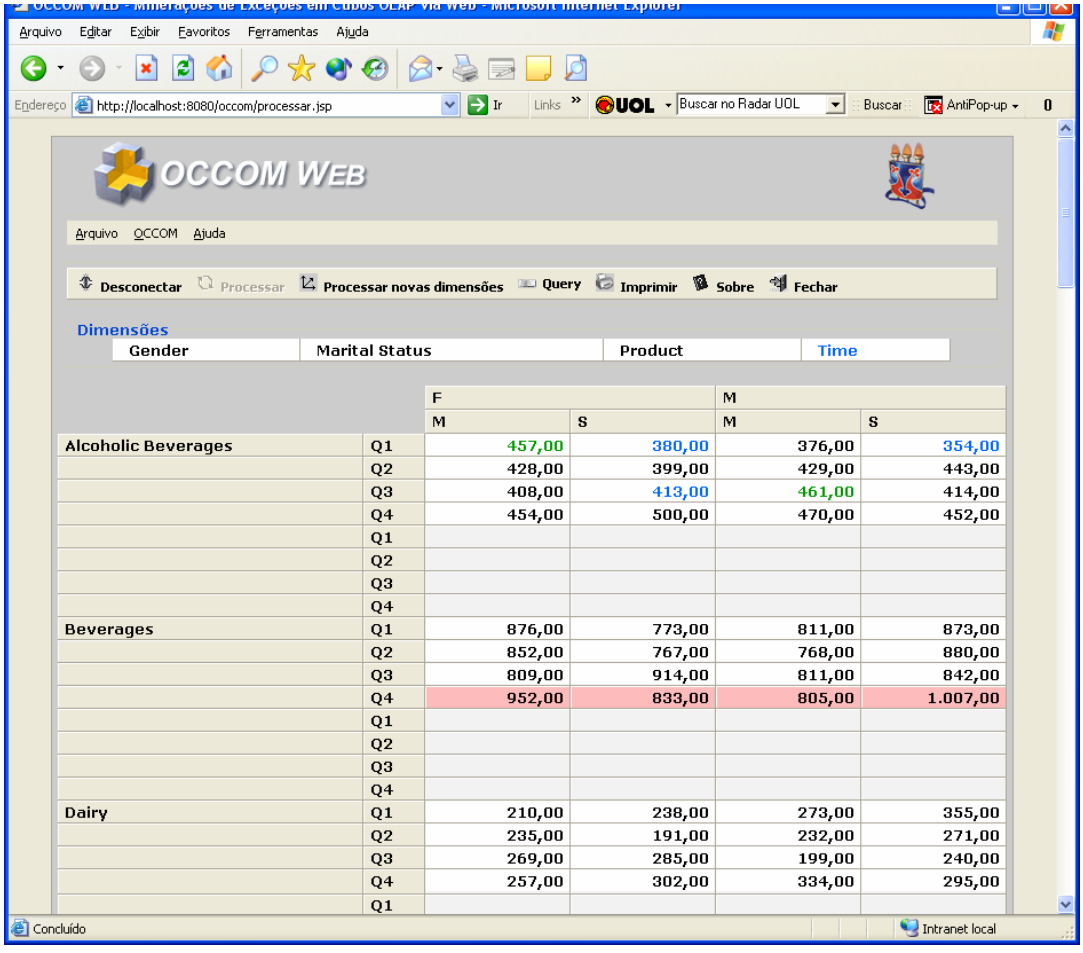


Figura 6.1 – Tela inicial da Interface OCCOM.



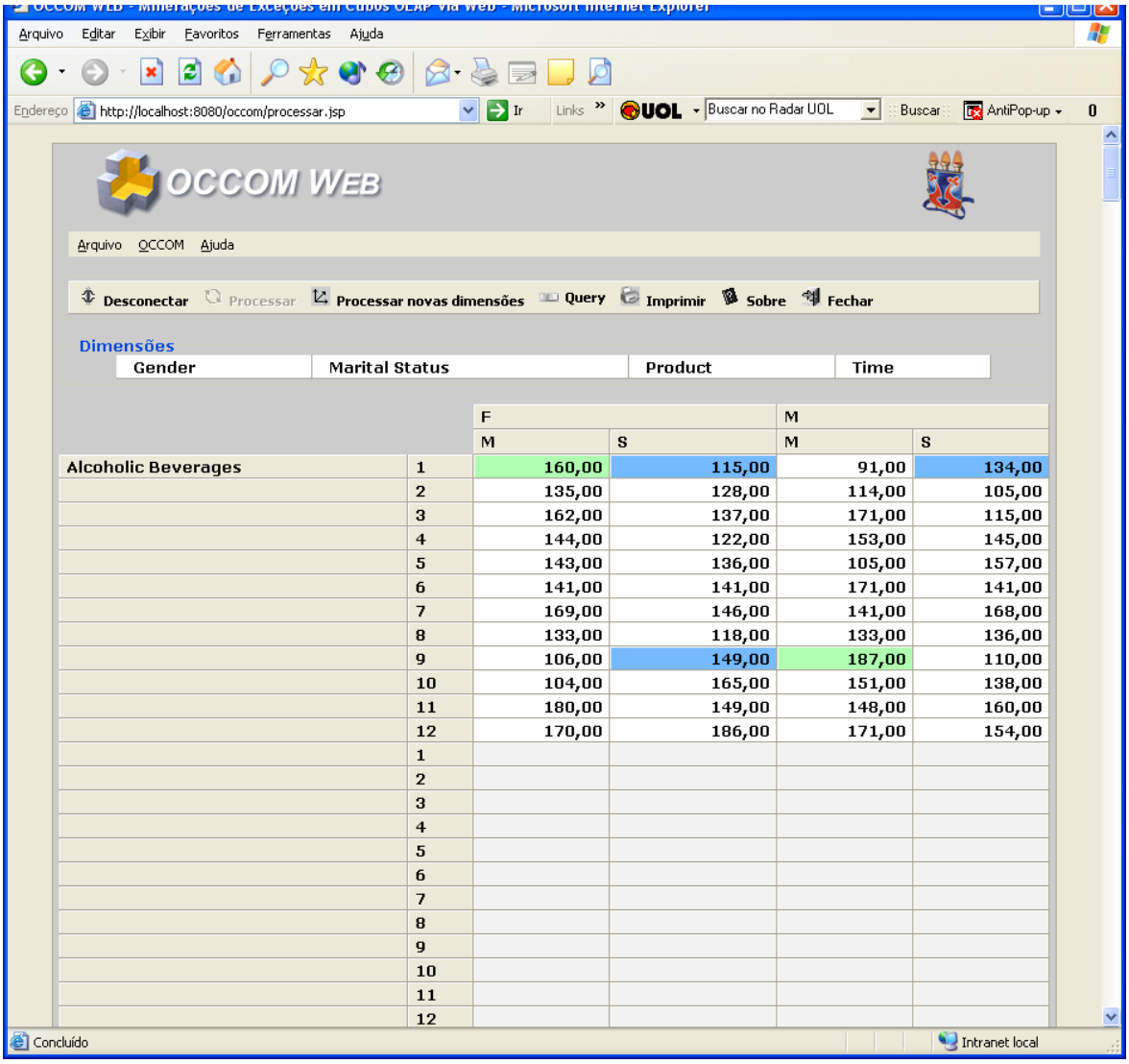
		F		M	
		M	S	M	S
Drink	1997	6.207,00	5.995,00	5.969,00	6.426,00
	1998				
Food	1997	47.187,00	47.627,00	47.742,00	49.384,00
	1998				
Non-Consumable	1997	11.942,00	12.600,00	12.749,00	12.945,00
	1998				

Figura 6.2 – Tela de navegação em cubos OLAP – Interface OCCOM.



		F		M	
		M	S	M	S
Alcoholic Beverages	Q1	457,00	380,00	376,00	354,00
	Q2	428,00	399,00	429,00	443,00
	Q3	408,00	413,00	461,00	414,00
	Q4	454,00	500,00	470,00	452,00
	Q1				
	Q2				
	Q3				
	Q4				
Beverages	Q1	876,00	773,00	811,00	873,00
	Q2	852,00	767,00	768,00	880,00
	Q3	809,00	914,00	811,00	842,00
	Q4	952,00	833,00	805,00	1.007,00
	Q1				
	Q2				
	Q3				
	Q4				
Dairy	Q1	210,00	238,00	273,00	355,00
	Q2	235,00	191,00	232,00	271,00
	Q3	269,00	285,00	199,00	240,00
	Q4	257,00	302,00	334,00	295,00
	Q1				

Figura 6.3 – Tela resultante de operação de *drill-down* nas dimensões *[Product]* e *[Time]*.



		F		M	
		M	S	M	S
Alcoholic Beverages	1	160,00	115,00	91,00	134,00
	2	135,00	128,00	114,00	105,00
	3	162,00	137,00	171,00	115,00
	4	144,00	122,00	153,00	145,00
	5	143,00	136,00	105,00	157,00
	6	141,00	141,00	171,00	141,00
	7	169,00	146,00	141,00	168,00
	8	133,00	118,00	133,00	136,00
	9	106,00	149,00	187,00	110,00
	10	104,00	165,00	151,00	138,00
	11	180,00	149,00	148,00	160,00
	12	170,00	186,00	171,00	154,00
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	10				
	11				
	12				

Figura 6.4 – Cubo base *[Product].[Product Department]+[Time].[Month]+[Gender]+[Marital Status]*.

7 CONCLUSÃO

Com o crescimento explosivo das informações dentro das organizações, novas tecnologias surgiram no intuito de auxiliar o processo de tomada de decisão. Neste contexto, a construção de *data warehouses* vem se destacando no papel de converter uma grande quantidade de dados em informação que possa ser utilizada de maneira estratégica.

A fim de facilitar a realização de consultas analíticas a dados armazenados em *data warehouses* surgiu a tecnologia de processamento analítico on-line (OLAP), que inclui técnicas de análise com funcionalidades para agregação dos dados, permitindo visões por diferentes ângulos.

Outra técnica que emergiu com grande sucesso para suporte à tomada de decisão é a Mineração de Dados, que fornece um serviço *automático* para descoberta de tendências gerais ou de dados atípicos, fornecendo *insights* sobre o domínio abordado.

No intuito de realizar uma integração sinérgica entre as tecnologias de OLAP e Mineração de dados, aproveitando a complementaridade entre elas, desenvolvemos OCCOM (*OLAP Cuboid Cell Outlier Miner*), um componente Java para mineração de exceções em células de cubos OLAP, que busca guiar usuários a explorarem cubos de maneira eficiente.

Componente do projeto MATRIKS, que tem como alvo a integração de tecnologias de *data warehousing*, bancos de dados multidimensionais, mineração de dados e geração automática de hipertextos em linguagem natural, OCCOM é um sistema cliente-servidor, de código aberto, independente de plataforma, multi-usuário e com uma interface gráfica amigável.

OCCOM foi desenvolvido a partir da integração de alguns algoritmos propostos para mineração de exceções em dados OLAP, e projetado para se comunicar com o servidor MSOLAP através de JODI (*Java OLAP Data Interface*), uma interface de programação que disponibiliza e integra serviços OLAP providos pelo servidor *OLE DB for OLAP*.

7.1 Contribuições

Destacamos as seguintes contribuições do trabalho realizado:

- OCCOM é o único componente de código aberto para mineração de exceções em células de cubos OLAP com características de ambiente cliente-servidor, multi-plataforma e multi-usuário;
- A combinação de OCCOM com o previamente desenvolvido gerador de hipertexto em linguagem natural HYSSOP, permite a geração automática de resumos, no formato de páginas web, de células de dados que são estatisticamente consideradas exceção em um contexto multidimensional e multi-granular de um *datawarehouse* OLAP;
- OCCOM cobre a principal lacuna na arquitetura do projeto MATRIKS, possibilitando a sua implementação;
- A interface JODI, buscou corrigir algumas limitações de JDCI (*Java Data Cube Interface*), implementação anterior do modelo de acesso a dados ODCI (FIDALGO, 2000): acesso concorrente e distribuído, submissão a testes de desempenho e integração com mineração de dados.

7.2 Trabalhos Futuros

A seguir apresentamos algumas tarefas que podem ser realizadas para aperfeiçoar o componente de software OCCOM:

- As células enriquecidas por mineração de OCCOM podem ser tornadas persistentes, a fim de evitar que o cálculo de exceções seja efetuado todas as vezes que o usuário acessar o sistema. Destacamos que esta é uma limitação do servidor MSOLAP, que não permite escrita em sua base;
- Durante o desenvolvimento deste trabalho surgiu uma nova tentativa de padronização de interfaces para disponibilização de serviços OLAP: a integração de OCCOM com CWM (*Common Warehouse Metamodel* – Seção 3.2.3) pode ser uma das alternativas no futuro;
- Apesar de OCCOM gerar entradas para HYSSOP, o mesmo foi desenvolvido na linguagem de programação LIFE, a construção de uma interface LIFE/Java permitiria a integração dessas duas ferramentas;
- A aplicação de OCCOM a casos concretos deve ser realizada a fim de torná-lo uma ferramenta confiável;
- Um estudo mais aprofundado dos algoritmos utilizados para cálculo de exceções deve ser realizado com a finalidade de melhorar o desempenho de OCCOM;

- Como JODI se constitui um dos fatores que prejudicam o desempenho de OCCOM, estudos devem ser realizados no intuito de desenvolver uma versão de JODI com desempenho similar ao de ADOMD, principalmente com relação à realização de consultas MDX.

8 REFERÊNCIAS

AGARWAL, S.; AGRAWAL, R.; DESHPANDE, P. M.; GUPTA, A.; NAUGHTON, J. F.; RAMAKRISHNAN, R.; SARAWAGI, S. **On the computation of multidimensional aggregates**. In Proc. of the 22nd Int´l Conference on Very Large Databases, pages 506-521, Mumbai (Bombay), India, September 1996.

AÏT-KACI, H.; LINCOLN, P. **LIFE: A natural language for natural language**. T. A. Informations, 30(1-2):37-67, Association pour le Traitement Automatique des Langues, Paris, France 1989.

BISHOP, Y.; FIENBERG, S.; HOLLAND, P. **Discrete Multivariate Analysis theory and practice**. The MIT Press, 1975.

BLAKELEY, J. A.; PIZZO, M. **Microsoft Universal Data Access Platform**. In ACM SIGMOD international conference on Management of data, pag 502-503, 1998.

BOOCHI, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário**. Campus, 2000.

CABIBBO, L.; TORLONE R. **Data Independence in OLAP Systems**. In Settimo Convegno Nazionale su Sistemi Evoluti per Basi di Dati, pages 35-49, 1999.

CAMPOS, M. L.; FILHO, A. V. R. **Data Warehouse Tutorial,** Url: http://genesis.nce.ufrj.br/dataware/tutorial/tutorial.html.

CHAUDHURI, S.; DAYAL, U. **Decision support, Data Warehouse, and OLAP,** in Tutorials of the Twenty-Second international Conf. On Very Large Data Base, Bombay, pages 295-306, 1996.

CHEN, Qing. **Mining Exceptions and Quantitative Association Rules in OLAP Data Cube**. Master of Science Thesis. Simon Fraser University, 1999.

CODD, E. F.; CODD, S.B.; SALLEY, C.T. **Providing OLAP to user Analysts: An IT Mandate,** White Paper, Arbor Software Corporation, 1993.

COOLEY, William W.; LOHNES, Paul R. **Multivariate data analysis**. Robert E. Krieger publishers, 1986.

FAVERO, E.L. **Generating hypertext summaries of data mining discoveries in multidimensional databases**. Tese de doutorado. CIn, Universidade Federal de Pernambuco, Recife, 2000.

FAYYAD, U. M.; SHAPIRO, G. Piatetsky; SMYTH, P.;UTHURUSAMY, R. **Advances in Knowledge Discovery and Data Mining.** AAAI/MIT Press, 1996.

FIDALGO, R. N. **JDCI: Uma API Java para disponibilização e integração de serviços OLAP**. Dissertação de Mestrado. CIn, Universidade Federal de Pernambuco, Recife, 2000.

GAMMA, E., et al. **Design Patterns**. Addison-Wesley, 1995.

GINGRAS, F.; LAKSHMANAN, L. V. S. **nD-SQL: A Multi-dimensional Language for Interoperability and OLAP**. Proc. 24th International Conference on Very Large Databases, New-York, NY, 1998.

GRAY, J.; BOSWORTH, A.; LAYMAN, A.; PIRAHESH, H. **Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals**. Technical Report, Microsoft Research, Microsoft Corporation, 1995.

GYSSENS, M.; LAKSHMANAN, L. V. S. **A foundation for multi-dimensional databases**. In Twenty-third International Conf. On Very Large Bata Bases, Athens, pages 106-115, 1997.

HAN, J.; CHIANG, J.; CHEE, S.; CHEN, J.; CHEN, Q.; CHENG, S.; GONG, W.; KAMBER, M.; KOPERSKI, K.; LIU, G.; LU, Y.; STEFANOVIC, N.; WINSTONE, L.; XIA, B.;ZAIANE, O. R.; ZHANG, S.; ZHU, H. **DBMiner: A System for Data Mining in Relational Database and Data Warehouses**. Proc. CASCON'97: Meeting of Minds, Toronto, Canada, November 1997.

HAN, Jiawei; KAMBER, Micheline. **Data Mining: Concepts and Techniques**. Morgan Kaufmann Publishers, 2001.

HORSTMANN, CAY S.; CORNELL, Gary. **Core Java 2 – Volume I – Fundamentos**. Makron Books, 2001.

HORSTMANN, CAY S.; CORNELL, Gary. **Core Java 2 – Volume II – Recursos Avançados**. Makron Books, 2001.

IETF Org. **Hypertext Transfer Protocol - HTTP/1.1.** Disponível em: <http://www.ietf.org/rfc/rfc2616.txt>, 2000.

INMON, W. H. **Build the Data Warehouse.** John Wiley, second edition, 1996.

KINBALL, R. **The Data Warehouse Toolkit**. John Wiley, 1996.

KNORR, Edwin M.; NG, Raymond T. **A Unified Notion of Outliers: Properties and Computation**. University of British Columbia. Vancouver, Canada, 1998.

KNORR, Edwin M.; NG, Raymond T. **Algorithms for Mining Distance-Based Outliers in Large Datasets**. University of British Columbia. Vancouver, Canada, 1998.

LEWANDOWSKI, S. M. **Frameworks for Component-Based Client/Server Computing**. ACM Computing Surveys, Vol. 30, No. 1, March 1998.

LINO, N. C. Q. **DOODI: Uma API para Integração entre Bancos de Dados Multidimensionais e Sistemas Dedutivos**. Dissertação de Mestrado. CIn, Universidade Federal de Pernambuco, Recife, 2000.

LISKOV, Barbara; GUTTAG, John. **Program Development in Java: abstraction, specification, and object-oriented design**. Addison-Wesley, 2001.

McCABE, M. Catherine; LEE, Jinho; CHOWDHURY, Abdur; GROSSMAN, David; FRIEDER, Ophir. **On the Design and Evaluation of a Multi-dimensional Approach to Information Retrieval**. U.S. Government, Illinois Insttitute of Technology, 2000.

Microsoft and Digital Equipment Corporation. **The Component Object Model Specification**. Disponível em: <http://www.msdn.microsoft.com/library/specs/S1D137.htm>, 1995.

Microsoft Corporation. **ADO Web Site**. Disponível em: <http://www.microsoft.com/data/ado/>. Acesso em: 2002.

Microsoft Corporation. **DCOM Web Site**. Disponível em: <http://www.microsoft.com/com/tech/dcom.asp>. Acesso em: 2001.

Microsoft Corporation. **Introduction to OLE DB for OLAP**. Disponível em: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/oledb/htm/>. Acesso em: 2002.

Microsoft Corporation. **Microsoft ADO MD Programmer's Reference**. Manual de Referências, 1998.

Microsoft Corporation. **Microsoft ADO Programmer's Reference**. Manual de Referência, 1998.

Microsoft Corporation. **OLE DB Programmer's Reference**. Manual de Referência, 1998.

Microsoft Corporation. *Universal Data Access Web Site.* Url: <http://www.microsoft.com/data/>, 2001.

Microsoft. **Overview of the .NET framework**, 2001. Disponível em: <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpovrintroductiontonetframeworksdk.asp>. Acesso em: Fevereiro/2003.

OLAP Council. **MDAPI (TM) the OLAP Application Program Interface Version 2.0 – Programmer's Guide**, 1997.

OLAP Council.**OLAP Council Web Site**, 1999. Disponível em: <http://www.olapcouncil.org/>, 1999. Acesso em: outubro/2001.

OMG - Object Management Group. **Common Warehouse Metamodel (CWM) Specification**, 2001.

OMG - Object Management Group. **CORBA Basics**, 2002. Disponível em: <http://www.omg.org/gettingstarted/corbafaq.htm>. Acesso em: Janeiro/2003.

OMG - Object Management Group. **Introduction To OMG's Unified Modeling Language (UML)**, 2002. Disponível em: <http://www.omg.org/gettingstarted/what_is_uml.htm>. Acesso em: Fevereiro/2003.

OMG - Object Management Group. **The Common Object Request Broker: Architecture and Specification**, 1998.

PENDSE, N. **OLAP Architectures.** Url: http://www.olapreport.com/Architectures.htm; 2000.

PETERSON, Timothy; PINKELMAN, James. **Microsoft OLAP Unleashed**. Sams, 1999.

PRESSMAN, Roger S. **Software engineering: a practitioner's approach**. 5th ed. McGraw-Hill, 2001.

PUC/RS. **Teste de Software**. Site da Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul. Disponível em: <http://inf.pucrs.br/~flavio/teste/>. Acesso em: janeiro/2003.

ROBIN, J.; FAVERO, E. **Content Aggregation in Natural Language Hypertext Summarization of OLAP and Data Mining Discoveries**. In First International Conference of Natural Language Generation, 2000.

ROBIN, J.; FAVERO, E. **HYSSOP: Natural Language Generation Meets Knowledge Discovery in Databases**. In Third International Conference on Information Integration and Web-Based Applications and Services, 2001, p. 243-256.

SARAWAGI, Sunita; AGRAWAL, Rakesh; MEGIDDO, Nimrod. **Discovery-driven Exploration of OLAP Data Cubes**. IBM Research Division, 1998.

SOARES, José F.; FARIAS, Alfredo A.; CÉSAR, Cibele C. **Introdução à Estatística**. Guanabara Koogan, 1991.

SUN Microsystems. **Enterprise JavaBeans Specification, Version 2.1**, 2002.

Sun Microsystems. **RMI – Remote Method Invocation Web Site**. Disponível em <http://java.sun.com/products/jdk/1.1/docs/guide/rmi/index.html>. Acesso em: março/2002.

SZYPERSKI, Clemens. **Component Software: Beyond Object-Oriented Programming**. Addison-Wesley, 1998.

TORLONE, Riccardo; CABIBBO, Luca. **A Logical Framework for Querying Multidimensional Data**. Dipartimento di Informatica e Automazione. Università di Roma. Rome, Italy, 1998.

W3C. **Extensible Markup Language (XML) 1.0 (Second Edition)**, 2000. Disponível em: <http://www.w3.org/TR/REC-xml>. Acesso em: Fevereiro/2003.

WITTEN, Ian H.; FRANK, Eibe. **Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations**. Morgan Kaufmann Publishers, 2000.

APÊNDICES

O Apêndice A fornece detalhes adicionais (para fins de consulta) sobre as classes JODI, seus métodos, sintaxe e funcionamento. O Apêndice B faz o mesmo com relação às classes OCCOM.

A. Classes JODI

A.1 JODIServer

Classe responsável pela conexão ao servidor JODI. A Tabela A.1 mostra a descrição dos métodos implementados por *JODIServer*.

Nome do Método	**Descrição**
JODIServer()	método construtor do servidor JODI, instancia o servidor e aguarda conexões.
getConnection()	retorna um novo objeto do tipo *iConnection*, que realiza a conexão a um servidor OLAP.
getConnection(*String*)	retorna um novo objeto do tipo *iConnection*, conectado a um servidor OLAP determinado pela string de conexão passada como parâmetro. Detalhes sobre a string de conexão podem ser vistos na próxima classe.
getName()	Retorna o nome do servidor: "JODI Server".
getServerName()	Retorna o nome da máquina (*host*) onde está instalado o servidor JODI.
getVersion()	Retorna o número da versão do servidor JODI.

Tabela A.1 – Métodos implementados por *JODIServer*.

A.2 cConnection

Classe responsável pela conexão ao servidor OLAP. A tabela A.2 mostra a descrição dos métodos implementados por *cConnection*.

Devido a grande quantidade de métodos implementados por essa classe, resolvemos agrupá-los por funcionalidade:

(1) Métodos construtores da classe;

(2) Métodos para conexão ao servidor OLAP;

(3) Métodos para manipulação de metadados do esquema multidimensional; e

(4) Métodos para manipulação de consultas multidimensionais.

Grupo	Nome do Método	Descrição
(1)	*cConnection* (*iJODIServer*)	método construtor da classe *cConnection*, recebe como parâmetro o servidor ODCI/JODI que manipula as conexões.
(1)	*cConnection*(*iJODIServer*,*String*)	método construtor da classe *cConnection*, recebe como parâmetro o servidor JODI e um texto de conexão. Abre automaticamente uma conexão a um servidor OLAP. O texto de conexão passado como parâmetro do construtor de *cConnection* é o mesmo parâmetro *arg0* usado pelo método *setConnection*() da classe *Catalog* de ADO MD (Figura 3.11). Devem ser informados: (1) o nome da máquina onde está instalado o servidor OLAP (*Data Source*), (2) o nome do esquema multidimensional (*Catalog*), e (3) o nome do servidor *OLE DB for OLAP* (*Provider*), que serão usados para abrir a conexão. Como informações opcionais, temos o nome do usuário (*Username*) e a senha (*Password*). Ao abrir uma conexão, a classe *cConnection* automaticamente irá iniciar o processo de re-escrita dos objetos COM de ADO MD[1], chamando o construtor de *cCube*. Um exemplo de string de conexão é "*Data Source=localhost; Initial Catalog = FoodMart 2000; Provider=MSOLAP*".
(2)	*openConnection*()	abre a conexão. Note que as informações de nome da máquina, esquema multidimensional, servidor *OLE DB for OLAP*, nome do usuário e senha, devem previamente ser informados através dos métodos *set* correspondentes.
(2)	*openConnection* (*String*)	abre a conexão. Recebe como parâmetro o texto de conexão, conforme descrito anteriormente no construtor da classe.
(2)	*closeConnection*()	fecha a conexão com o servidor OLAP.
(2)	*getConnectionStatus*()	retorna um inteiro informando o status da conexão. O valor retornado pode ser comparado às constantes *CONNECTION_CLOSED* (cubo fechado) e *CONNECTION_OPENED* (cubo aberto).
(2)	*setConnectionString* (*String*)	determina o texto de conexão utilizado para abrir a conexão com o servidor OLAP.
(2)	*setDatasourceName* (*String*)	determina o nome da máquina onde está instalado o servidor OLAP.
(2)	*setCatalogName* (*String*)	determina o nome do esquema multidimensional para conexão.
(2)	*setProviderName* (*String*)	determina o nome do servidor *OLE DB for OLAP*, geralmente "*MSOLAP*".
(2)	*setUsername*(*String*)	determina o nome do usuário para conexão.
(2)	*setPassword*(*String*)	determina a senha do usuário.
(3)	*getCubeCount*()	retorna a quantidade de cubos existentes no esquema multidimensional.
(3)	*getCube*(*int*)	retorna pelo índice (parâmetro), um objeto do arranjo de cubos do esquema multidimensional, note que este índice sempre inicia com o valor "0".
(3)	*getCube*(*String*)	retorna pelo nome (parâmetro), um cubo do esquema multidimensional.
(3)	*getCubes*()	retorna o conjunto de cubos existentes no esquema multidimensional.
(4)	*getMDQuery*()	retorna um objeto para realização de uma consulta MDX.
(4)	*getMDQuery*(*String*)	retorna um objeto de consulta MDX. O texto da consulta deve ser passado como parâmetro. Automaticamente instancia os objetos do resultado da consulta multidimensional.

Tabela A.2 – Métodos implementados por *cConnection*.

[1] ADO MD e *OLE DB for OLAP* são dependentes da plataforma COM e por isso, para superar esta limitação, deve-se re-escrever seus objetos de forma que após re-escritos, se tornem independentes desta plataforma.

A.3 cCube

Classe que implementa a interface *iCube* e re-escreve os objetos COM da interface *CubeDef* de ADO MD.

Nome do Método	Descrição
CCube(*CubeDef, cConnection, int*)	método construtor que manipula as operações da interface *CubeDef* de ADO MD (parâmetro) para re-escrever a classe *cCube*. Este construtor também é responsável por: (1) registrar o objeto *cConnection* que o chamou (parâmetro) e instanciar os objetos da classe *cDimension*. O parâmetro inteiro representa o número (*number*) ordinal de *cCube*
getType()	retorna o inteiro que denota o tipo do cubo, que pode ser igual ao valor de alguma das constantes *MB_CBTYPE_CUBE* (cubo) e *MB_CBTYPE_VIRTUAL_CUBE* (cubo virtual). O primeiro identifica que o cubo foi gerado a partir dos dados de um *data warehouse* e o segundo, que o cubo foi gerado a partir de cubos pré-existentes. Ou seja, fazendo analogia com um banco de dados relacional, um cubo seria uma tabela e um cubo virtual seria uma visão materializada de uma ou mais tabelas
getDimension(*int*)	retorna, pelo índice (parâmetro), um objeto do arranjo de dimensões do cubo
getDimension(*String*)	retorna, pelo nome único (parâmetro), um objeto do arranjo de dimensão do cubo
getDimensions()	retorna o arranjo de dimensões que formam o cubo
getDimensionCount()	retorna a quantidade de objetos do arranjo de dimensões do cubo
getCubeNumber()	retorna um número inteiro que indica o ordinal do cubo

Tabela A.3 – Métodos implementados por *cCube*.

A.4 cDimension

Classe responsável por reescrever os objetos COM da interface *Dimension* de ADO MD.

Nome do Método	Descrição
cDimension(*cCube, Dimension, int*)	método construtor que manipula as operação da interface *Dimension* (parâmetro) de ADO MD para re-escrever a classe *cDimension*. Este construtor também é responsável por: (1) registra o objeto *cCube* que o chamou (parâmetro) e (2) instanciar a classe *cHierarchy*. O parâmetro inteiro representa o número (*number*) ordinal de *cDimension*
getType()	retorna o inteiro que denota o tipo da dimensão. Pode ser igual ao valor das constantes *MD_DIMTYPE_TIME* (para dimensões do tipo tempo), *MD_DIMTYPE_MEASURE* (para medidas – em *OLE DB for OLAP* medidas são tratadas como um dimensão de um tipo específico), *MD_DIMTYPE_OTHER* (outro tipo de dimensão) ou *MD_DIMTYPE_UNKNOWN* (tipo de dimensão desconhecido)
getCardinality()	retorna o número de membros na dimensão (*cLevelMember*)
getHierarchyCount()	retorna a quantidade de objetos do arranjo hierarquia da dimensão
getHierarchy(*int*)	retorna, pelo índice (parâmetro), um objeto do arranjo de hierarquias da dimensão
getHierarchy(*String*)	retorna, pelo nome único (parâmetro), um objeto do arranjo de hierarquias da dimensão
getHierarchies()	retorna o conjunto de hierarquias da dimensão
getDefaultHierarchy()	retorna a hierarquia padrão (*default*) da dimensão
getDimensionNumber()	retorna o ordinal que identifica a dimensão

Tabela A.4 – Métodos Implementados por *cDimension*.

A.5 cHierarchy

Classe responsável por reconstruir os objetos COM da interface *Hierarchy* de ADO MD.

Nome do Método	Descrição
cHierarchy(*cDimension, Hierarchy, int*)	método construtor que manipula as operações da interface *Hierarchy* (parâmetro) de ADO MD para re-escrever a classe *cHierarchy*. Este construtor também é responsável por: (1) registrar o objeto *cDimension* (parâmetro) que o chamou e (2) instanciar a classe *cLevel*. O parâmetro inteiro representa o número (*number*) ordinal de *cHierarchy*
getType()	retorna o inteiro que denota o tipo de estrutura da hierarquia. Pode ser igual ao valor das constantes *MD_STRUCTURE_FULLYBALANCED* (completamente balanceada), *MD_STRUCTURE_RAGGEDBALANCED* (parcialmente balanceada ou não definido), *MD_STRUCTURE_UNBALANCED* (não balanceada) ou *MD_STRUCTURE_ NETWORK* (estrutura de rede). Em uma hierarquia balanceada, todos os ramos da hierarquia descendem do mesmo nível e cada pai lógico de um membro é um membro de um nível imediatamente acima (por exemplo, em uma dimensão geográfica, os membros do nível país são limitados a Alemanha e França; a hierarquia da dimensão é balanceada e todos os ramos da hierarquia terminam em uma cidade no nível Cidade; o pai de cada cidade deve necessariamente ser Alemanha ou França, e Europa é o pai tanto de França como de Alemanha. Em uma hierarquia não balanceada, ramos da hierarquia podem descender de diferentes níveis (por exemplo, um dimensão Organização contém um membro para cada empregado em uma empresa; o CEO é o membro mais alto da hierarquia, e os gerentes de divisão e secretário executivo estão imediatamente abaixo do CEO; os gerentes de divisão possuem membros subordinados, mas o secretário executivo não possui). Em uma hierarquia parcialmente balanceada, o pai lógico de pelo menos um membro não está no nível imediatamente acima do membro, isto pode fazer com que ramos de uma hierarquia descendam de diferentes níveis. A estrutura de hierarquias em rede não são implementadas na prática pelo MS OLAP Server.
getCardinality()	retorna o número de membros na hierarquia
getLevel(*int*)	retorna, pelo índice (parâmetro), um objeto do arranjo de níveis da hierarquia
getLevel(*String*)	retorna, pelo nome único (parâmetro), um objeto do arranjo de níveis da hierarquia
getLevels()	retorna o arranjo de níveis da hierarquia
getLevelCount()	retorna a quantidade de objetos do arranjo de níveis da hierarquia
getDefaultMember()	retorna o membro padrão da hierarquia. Representa o valor padrão que o sistema usa quando o usuário não especifica membros na consulta MDX
getAllMember()	retorna o membro geral da hierarquia. É o membro do primeiro nível da hierarquia, representa o pai de todos os membros dessa hierarquia
getHierarchyNumber()	retorna o ordinal que identifica a hierarquia

Tabela A.5 – Métodos Implementados por *cHierarchy*.

A.6 cLevel

Classe responsável por re-escrever os objetos COM da interface *Level* de ADO MD.

Nome do Método	Descrição
cLevel(*cHierarchy*, *Level*, *int*)	método construtor que manipula as operações da interface *Level* de ADO MD (parâmetro) para re-escrever a classe *cLevel*. Também registra o objeto *cHierarchy* que o chamou (parâmetro). O parâmetro inteiro representa o número (*number*) ordinal de *cLevel*. Diferentemente das outras classes de metadados vistas até aqui, *cLevel* não instancia automaticamente os objetos da classe *cLevelMember* por motivos que veremos na seção 4.3
makeMembers()	método privado que instancia os objetos da classe *cLevelMember* (membros do nível)
getMember(*int*)	retorna, pelo índice (parâmetro), um objeto do arranjo de membros do nível
getMember(*String*)	retorna, pelo nome único (parâmetro), um objeto do arranjo de membros do nível
getMembers()	retorna o arranjo de membros do nível
getMemberCount()	retorna a quantidade de objetos do arranjo de membros do nível
getCardinality()	retorna o número de membros do nível
getDepth()	retorna um inteiro que denota a profundidade do nível - distância do nível à raiz da hierarquia (primeiro nível)
getType()	retorna o inteiro que denota o tipo do nível. Pode ser igual ao valor das constantes *MD_LEVEL_TYPE_REGULAR* (nível que não requer nenhum formato especial), *MD_LEVEL_TYPE_ALL* (primeiro nível da hierarquia – maior granularidade), *MD_LEVEL_TYPE_CALCULATED* (nível calculado - gerado por uma fórmula), *MD_LEVEL_TYPE_TIME* (tempo), *MD_LEVEL_TYPE_TIME_YEARS* (anos), *MD_LEVEL_TYPE_TIME_HALF_YEAR* (semestres), *MD_LEVEL_TYPE_TIME_QUARTERS* (trimestres), *MD_LEVEL_TYPE_TIME_MONTHS* (meses), *MD_LEVEL_TYPE_TIME_WEEKS* (semanas), *MD_LEVEL_TYPE_TIME_DAYS* (dias), *MD_LEVEL_TYPE_TIME_HOURS* (horas), *MD_LEVEL_TYPE_TIME_MINUTES* (minutos), *MD_LEVEL_TYPE_TIME_SECONDS* (segundos), *MD_LEVEL_TYPE_TIME_UNDEFINED* (indefinido) ou *MD_LEVEL_TYPE_UNKNOWN* (desconhecido – não classificado pelo servidor *OLE DB for OLAP*)
getLevelNumber()	retorna o ordinal que identifica o nível

Tabela A.6 – Métodos Implementados por *cLevel*.

A.7 cLevelMember

A classe *cLevelMember* implementa a interface *iLevelMember* e re-escreve no seu método construtor os objetos COM da interface *Member* de ADO MD.

Nome do Método	Descrição
cLevelMember(*cLevel, Member, int*)	método construtor que manipula as operação da interface *Member* de ADO MD (parâmetro) para re-escrever a classe *cLevelMember*. Por motivos que veremos adiante, *cLevelMember* não instancia automaticamente os seus filhos. Também é responsável por registrar o objeto *cLevel* que o chamou (parâmetro). O parâmetro inteiro representa o número (*number*) ordinal de *cLevelMember*
makeChildren()	método privado que instancia os filhos de um objeto *cLevelMember*
getChild(*int*)	retorna, pelo índice (parâmetro), um objeto do arranjo de filhos do membro
getChild(*String*)	retorna, pelo nome único (parâmetro), um objeto do arranjo de filhos do membro
getChildren()	retorna o arranjo de filhos do membro
getChildCount()	retorna a quantidade de objetos do arranjo de filhos do membro
getHasParent()	retorna falso (*false*) se o membro é um membro raiz, ou verdadeiro (*true*) se não é um membro raiz e portanto, possui pai
getParent()	retorna o membro pai (de algum nível acima)
getType()	retorna o inteiro que denota o tipo do membro. Pode ser igual ao valor das constantes *MDMEMBER_TYPE_REGULAR* (membro normal), *MDMEMBER_TYPE_ ALL* (membro pai de todos os membros da hierarquia), *MDMEMBER_TYPE_FORMULA* (membro calculado por uma fórmula), *MDMEMBER_TYPE_MEASURE* (membro da dimensão medida) ou *MDMEMBER_TYPE_UNKNOWN* (membro não classificado pelo servidor *OLE DB for OLAP*)
getMemberNumber()	retorna o ordinal que identifica o membro

Tabela A.7 – Métodos Implementados por *cLevelMember*.

A.8 cMDQuery

Classe que permite a manipulação de consultas multidimensionais (MDX). Os métodos da classe *cMDQuery* podem ser vistos na Tabela A.8.

Nome do Método	Descrição
cMDQuery(*iConnection*)	método construtor chamado pelo método *getMDQuery*() de *cConnection* que definirá a conexão aberta (parâmetro) que será utilizada pelo método *open*()
cMDQuery(*iConnection, String*)	método construtor chamado pelo método *getMDQuery*(*String*) de *cConnection*. Define a conexão aberta (parâmetro), o texto da consulta MDX (parâmetro) e executa a consulta (chamada ao método *open*())
setQuery(*String*)	Define uma nova consulta MDX (parâmetro)
getQuery()	retorna a declaração da consulta MDX executada
open()	executa a consulta MDX. Instancia um objeto da classe *cMDResult*
close()	fecha a consulta MDX atual, para que uma nova consulta possa ser realizada
getStatus()	retorna o estado da consulta. Pode ser igual ao valor das constantes *MDQUERY_CLOSED* (consulta fechada) ou *MDQUERY_OPENED* (consulta aberta)
getMDResult()	retorna uma interface de um objeto de resultado de uma consulta MDX (*iMDResult*)

Tabela A.8 – Métodos Implementados por *cMDQuery*.

A.9 cMDResult

A classe *cMDResult* re-escreve os objetos COM da classe *Cellset* e das interfaces *Axis*, *Cell* e *Position* de ADO MD. Implementa os métodos listados na Tabela A.9.

		Products		
Gender	Marital Stat.	**Drink**	**Food**	**Non-Cons.**
F	**M**	6.207,00	47.187,00	11.942,00
F	**S**	5.995,00	47.627,00	12.600,00
M	**M**	5.969,00	47.742,00	12.749,00
M	**S**	6.426,00	49.384,00	12.945,00

Figura A.1 – Cubo de exemplo para consultas MDX.

Nome do Método	**Descrição**
cMDResult(*Cellset, cMDQuery*)	método construtor que manipula as operações da interface *Cellset* de ADO MD (parâmetro) para re-escrever a classe *cMDResult*. Este construtor também é responsável por: (1) registrar o objeto *cMDQuery* que o chamou (parâmetro) e (2) instanciar os objetos da classe *cAxis* (eixo)
getCell(*int*)	retorna, pelo índice (parâmetro), um objeto célula do resultado multidimensional (*iCell*). Note que, diferentemente dos eixos, previamente instanciados, a célula somente é instanciada no momento em que é solicitada através de um método de *cMDResult*. Por exemplo, a 3ª célula do cubo da Figura A.1 (*Non-Consumable+F+M*, cujo valor é 11.942,00) pode ser acessada através do comando *myMDResult.getCell*(*2*)
getCell(*int, int*)	retornar, pelas coordenadas (parâmetros coluna e linha), um objeto célula do resultado multidimensional. Ex.: a célula *Food+M+M* (valor de 47.742,00) pode ser acessada através do comando *myMDResult.getCell*(*1, 2*)
getCell(*String*)	retornar, pelo rótulo (parâmetro), um objeto célula do resultado multidimensional. O rótulo da célula é formado pelos rótulos dos membros (*cPositionMember*) que formam as coordenadas da célula da seguinte forma: rótulos entre colchetes "[rótulo]", com membros de um mesmo eixo separados pelo sinal "." ([A].[B]) e membros de eixos diferentes separados pelo sinal de "+" ([A].[B]+[C].[D]). Por exemplo, para acessar a célula 10 da Figura A.1 – coordenadas (1, 3) – utiliza-se *myMDResult.getCell*(*"[Food]+[M].[S]"*), ou *myMDResult.getCell*(*10*), ou *myMDResult.getCell*(*1, 3*), ou ainda *myMDResult.getCell*(*"[Food]", "[M].[S]"*), como será visto no próximo método
getCell(*String, String*)	retorna, pelos rótulos das coordenadas (parâmetros), um objeto célula do resultado multidimensional. Por exemplo, a célula *Drink+M+S* (valor de 6.426,00) da Figura 4.13 pode ser acessada através do comando *myMDResult.getCell*(*"[Drink]", "[M].[S]"*)
getCellCount()	retorna a quantidade de células do resultado da consulta multidimensional. No nosso exemplo, 12 células.
getAxisCount()	retorna a quantidade de eixos do resultado multidimensional (máximo de 2)
getAxis(*int*)	retorna, pelos ordinais 0 ou 1 (parâmetro), um objeto eixo coluna ou eixo linha do resultado multidimensional. Por exemplo, o comando *myMDResult.getAxis*(*0*) retornaria o eixo formado pela dimensão *Product*, no cubo da Figura A.1.
getAxis(*String*)	retorna, pelas palavras *Column* (coluna) ou *Row* (linha), um dos eixos do resultado multidimensional. Por exemplo, *myMDResult.getAxis*(*"Row"*)
getAxes()	retorna o arranjo de eixos do resultado multidimensional
getFilterAxis()	retorna o eixo restritivo definido pela cláusula WHERE de MDX

Tabela A.9 – Métodos Implementados por *cMDResult*.

A.10 cAxis

A classe cAxis ...

Nome do Método	Descrição
cAxis(*Axis, cMDResult*)	método construtor que manipula as operações da interface *Axis* de ADO MD (parâmetro) para re-escrever a classe *cAxis*. Também é responsável por registrar o objeto *cMDResult* que o chamou (parâmetro) e instanciar os objetos da classe *cPosition*
getName()	retorna o nome do eixo (*Column* ou *Row*)
getDimensionCount()	retorna a quantidade de dimensões existentes no eixo. Por exemplo, o eixo 1 (linha) da Figura 4.13, possui 2 dimensões: *[Gender]* e *[Marital Status]*
getPosition(*int*)	retorna, pela sua posição (parâmetro), um objeto *cPosition* (posição) que forma o eixo
getPosition(*String*)	retorna, pelo seu rótulo (parâmetro), um objeto *cPosition* que forma o eixo. Por exemplo, o eixo 0 (*Column*) da Figura 3.3 possui as posições "*[Drink]*", "*[Food]*" e "*[Non-Consumable]*", e o eixo 1 (*Row*), possui as posições "*[F].[M]*", "*[F].[S]*", "*[M].[M]*" e "*[M].[S]*"
getPositions()	retorna o arranjo de posições do eixo
getPositionCount()	retorna a quantidade de posições existentes no eixo

Tabela A.10 – Métodos Implementados por *cAxis*.

A.11 cCell

A classe *cCell* é responsável por re-escrever os objetos COM da interface *Cell* de ADO MD.

Nome do Método	Descrição
cCell(*Cell, cMDResult, Variant[]*)	método construtor que manipula as operações da interface *Cell* de ADO MD (parâmetro) para re-escrever a classe *cCell* (célula). Este construtor também é responsável por registrar o objeto *cMDResult* que o chamou (parâmetro) e referenciar os seus objetos da classe *cPosition* que foram instanciados pelo construtor da classe eixo (*cAxis*). Recebe como parâmetro, ainda, os números de ordem (coordenadas) que indicam a posição da célula em cada um dos eixos
getCaption()	retorna o rótulo da célula. Por exemplo, a célula 10 da Figura 4.13 possui o rótulo "*[Food]+[M].[S]*"
getIsNull()	retorna verdadeiro caso do valor da célula seja nulo (não possui valor) ou falso caso contrário
getValue()	retorna o valor da célula no formato numérico (*float*). A instrução *myMDResult.getCell*(*"[Food]+[M].[S]"*).*getValue*() retornaria o valor numérico 49.384,00, para o cubo da Figura 4.13.
getFormatedValue()	retorna o valor formatado da célula. A instrução *myMDResult.getCell* (*"[Food]+[M].[S]"*).*getFormatedValue*() retornaria o texto "R$ 49.384,00", para o cubo da Figura 4.13.
getPosition(*int*)	retorna, pela número da posição (parâmetro), um objeto *cPosition* da célula (valores 0 – coluna, ou 1 – linha). Por exemplo, a primeira célula do cubo da Figura 4.13 possui as posições *[Drink]* (coluna) e *[F].[M]* (linha)
getPosition(*String*)	retorna, pelo rótulo da posição (parâmetro) ou pelos nomes "*Column*" ou "*Row*", um objeto *cPosition* da célula. Por exemplo, *myCell.getPosition*(*"[Drink]"*) ou *myCell.getPosition*(*"Column"*)
getPositions()	retorna o arranjo de posições da célula
getPositionCount()	retorna a quantidade de posições (ordenadas) existentes na célula (1 ou 2)

Tabela A.11 – Métodos Implementados por *cCell*.

A.12 cPosition

A classe *cPosition* é responsável por re-escrever os objetos COM da interface *Position* de ADO MD.

Nome do Método	Descrição
cPosition(*Position, cAxis*)	método construtor que manipula as operações da interface *Position* de ADO MD (parâmetro) para re-escrever a classe *cPosition*. Este construtor também é responsável por registrar o objeto *cAxis* que o chamou e por instanciar os seus objetos da classe *cPositionMember*
getName()	retorna o nome da posição
getMember(*int*)	retorna, pela posição do membro (parâmetro), um objeto da classe *cPositionMember* (membro da posição no eixo). Por exemplo, a instrução *myMDResult.getAxis*(*"Row"*).*getPosition*(3).*getMember*(0) retornaria o membro *[M]*, para o caso do cubo da Figura 4.13.
getMember(*String*)	retorna, pelo rótulo do membro (parâmetro), um objeto da classe *cPositionMember*
getMembers()	retorna o arranjo de membros existentes em uma determinada posição do eixo
getMemberCount()	retorna a quantidade de membros existentes na posição. Corresponde ao número de dimensões do eixo.

Tabela A.12 – Métodos Implementados por *cPosition*.

A.13 cPositionMember

A classe *cPositionMember* é responsável por re-escrever os objetos COM da interface *Member* de ADO MD.

Nome do Método	Descrição
cPositionMember(*Member, cPosition*)	método construtor que manipula as operação da interface *Member* de ADO MD (parâmetro) para re-escrever a classe *cPositionMember*. Também é responsável por registrar o objeto da classe *cPosition* que o instanciou
getName()	retorna o nome do membro
getCaption()	retorna o rótulo do membro
getDrilledDown()	retorna verdadeiro (*true*) ou falso (*false*) quanto a propriedade *DrilledDown*
getParentSameAsPrev()	retorna verdadeiro (*true*) quando o membro está "*drilleddown*" e possui o mesmo pai, ou falso (*false*) quanto isto não ocorre

Tabela A.13 – Métodos Implementados por *cPositionMember*.

B Classes OCCOM

B.1 cMiner

A classe *cMiner* é responsável pelo cálculo de exceções e inicialização dos demais objetos OCCOM.

Nome do Método	Descrição
cMiner()	construtor da classe, todos os atributos são inicializados com os seus valores padrão
cMiner(*iCube*)	construtor da classe, os atributos são inicializados com valores padrão e o cubo sobre o qual serão computadas as exceções é indicado através do parâmetro, ou seja, o cliente já deve ter se conectado ao servidor JODI e posteriormente ao servidor *OLE DB for OLAP*, e selecionado com qual cubo irá trabalhar. Este procedimento é semelhante a criação do objeto com o construtor *cMiner*() e posterior execução do método *setCube*(*iCube*)
setCube(*iCube*)	indica qual cubo de dados (interface *iCube* de JODI) será utilizado para cálculo das exceções
setMeasure(*String*)	indica qual o nome da medida (nome do nível em uma dimensão do tipo medida em JODI), por exemplo *myMiner.setMeasure*(*"[Measures].[Unit Sales]"*);. Ao executar este método, um objeto do tipo *cMinerMeasure* é criado e inicializado por *cMiner*
setModel(*int*)	indica qual modelo será aplicado para o cálculo de exceções. Pode ser acessado através das constantes *MODEL_LINEAR* e *MODEL_LOG_LINEAR*
setOutlierPerc(*int*)	indica o grau de sensibilidade de OCCOM a desvios (*outliers*). Sarawagi, Agrawal e Megiddo (1998) trabalham com um percentual de 25%, ou seja, as médias para o cálculo de exceções são calculadas com 75% dos valores, 25% dos valores extremos são descartados. Em OCCOM este valor é parametrizado.
addDimension(*String*)	adiciona uma dimensão (e em até que nível) que será utilizada para cálculo de exceções. Através do método *addDimension* o cliente pode determinar quais dimensões formarão o cubóide base. Caso a dimensão já exista e um novo nível for indicado, este substituirá o anterior. Na verdade o usuário deve passar como parâmetro uma String contendo o nome da dimensão e o nível, como no exemplo: *myMiner.addDimension*(*"[Product].[Product Subcategory]"*); ou *myMiner.addDimen sion*(*"[Time].[Quarter]"*);. Ao executar este método, objetos do tipo *cMinerDimension*, *cMinerLevel* e *cMinerMember* são criados e inicializados por *cMiner*
removeDimension (*String*)	remove uma dimensão para o cálculo de exceções. O cliente deve passar como parâmetro o nome da dimensão a ser removida
open()	inicializa os objetos *cMinerCube*, cubóides gerados a partir do cubo base, e suas células – objetos do tipo *cMinerCell*. Observe, através do diagrama de seqüência da Figura 5.1, que os cubóides e suas células são gerados através de chamadas a consultas MDX, que posteriormente podem ser acessadas através do método *getQuery*() de *cMinerCube*. Após gerar os cubos, *cMiner* realiza o cálculo de exceções das células utilizando o método de computação escolhido através de *setModel*(*int*). Terminado este processo, as outras medidas de exceção são computadas (*UnderExp* e *PathExp* das células e *SelfExp*, *UnderExp* e *PathExp* dos cubos)
close()	fecha o cubo, permitindo que parâmetros sejam alterados para uma nova tarefa de cálculo de exceções
getStatus()	retorna a situação atual de *cMiner*. Pode ser comparada às constantes *MINER_CLOSED* (igual a 0), que indica que as exceções não foram computadas; e *MINER_OPENED* (igual a 1), que indica que as exceções já foram computadas
getDimensionCount()	retorna o número de dimensões que será utilizada para cálculo de exceções
getDimension(*int*)	retorna um objeto de dimensão (*iMinerDimension*), recebendo como parâmetro o seu número

getDimension(*String*)	retorna um objeto de dimensão (*iMinerDimension*), recebendo como parâmetro o seu rótulo
getDimensions()	retorna o conjunto de dimensões que serão utilizadas para o cálculo de exceções
getMinerCube(*int*)	retorna um objeto do tipo cubo de dados enriquecido por mineração (*iMinerCube*), realizando a busca pelo seu número (parâmetro)
getMinerCube(*String*)	retorna um objeto do tipo cubo de dados enriquecido por mineração (*iMinerCube*), realizando a busca pelo seu rótulo (parâmetro). Ao gerar os cubos (método *open*), *cMiner* forma seus rótulos pelos nomes das suas dimensões separados pelo sinal de "+", por exemplo, *myMiner.getMinerCube*("*[Product]*. *[Product Family]+[Gender]*. *[Gender]+[Time]*.*[Year]"*); retornaria o cubóide enriquecido por mineração formado pelas dimensões *[Product]*, em seu nível *[Product Family]*; *[Gender]*, no nível *[Gender*]; e *[Time]*, no nível *[Year]*
getMinerCube (*String[]*)	em vez de receber como parâmetro os rótulos das dimensões separados pelo sinal de "+", *cMiner* também pode realizar a busca do cubo enriquecido por mineração recebendo como parâmetro uma matriz formada pelos rótulos de cada uma das dimensões
getMinerCubes()	retorna o conjunto de cubóides enriquecidos por mineração gerados por *cMiner*
getMinerCubeCount()	retorna a quantidade de cubóides gerados por *cMiner*
getAllValue()	retorna o valor do cubo *All* (agregação total de todas as dimensões);
getFormatedAllValue()	retorna o valor formatado do cubo *All*

Tabela B.1 – Métodos Implementados por *cMiner*.

B.2 cMinerMeasure

A classe *cMinerMeasure* implementa a medida utilizada para o cálculo de exceções.

Nome do Método	**Descrição**
cMinerMeasure (*iLevelMember*)	método construtor da classe. Recebe como parâmetro a classe JODI correspondente à medida (*iLevelMember*) que será reescritas por *cMinerMeasure*
getName()	retorna o nome do membro que representa a medida
getUniqueName()	retorna o nome único do membro que representa a medida
getCaption()	retorna o rótulo do membro que representa a medida

Tabela B.2 – Métodos Implementados por *cMinerMeasure*.

B.3 cMinerDimension

A classe *cMinerDimension* implementa as informações de metadados das dimensões utilizadas para o cálculo de exceções.

Nome do Método	Descrição
cMinerDimension (*iDimension, iLevel*)	método construtor da classe, é responsável pela inicialização da dimensão, seus níveis (*cMinerLevel*) e membros (*cMinerMember*). Recebe como parâmetro as classes JODI correspondentes à dimensão (*iDimension*) e ao nível (*iLevel*), que serão reescritas por *cMinerDimension*
getLevel(*int*)	retorna o nível da dimensão correspondente à profundidade indicada (parâmetro). Note que esta profundidade vai até o limite indicado pelo método *addDimension* de *cMiner*
getLevel(*String*)	retorna o nível da dimensão que possua o rótulo indicado como parâmetro
getLevels()	retorna o conjunto de níveis da dimensão
getLevelsCount()	retorna a quantidade de níveis da dimensão, correspondente à profundidade que será utilizada para o cálculo de exceções

Tabela B.3 – Métodos Implementados por *cMinerDimension*.

B.4 cMinerLevel

A classe *cMinerLevel* implementa os objetos que contém os metadados de níveis de uma dimensão.

Nome do Método	Descrição
CMinerLevel (*iMinerDimension, iLevel*)	Construtor da classe, reescreve os objetos de nível do cubo de JODI (*iLevel*) e inicializa os objetos de membros do nível (*iMinerMember*) utilizados para o cálculo de exceções. Recebe ainda, como parâmetro, a classe de dimensão (*iMinerDimension*) que a inicializou. Além disso, cria os relacionamentos pai-filho entre os membros do nível
getDepth()	retorna a profundidade do nível
getMember(*String*)	retorna o membro do nível, recebendo como parâmetro o seu rótulo
getMember(*int*)	retorna o membro do nível, recebendo como parâmetro o seu índice
getMembers()	retorna o conjunto de membros do nível
getMemberCount()	retorna a quantidade de membros do nível

Tabela B.4 – Métodos Implementados por *cMinerLevel*.

B.5 cMinerMember

A classe *cMinerMember* re-escreve os metadados dos membros dos níveis de JODI e seus relacionamentos pai-filho.

Nome do Método	Descrição
cMinerMember (*iMinerLevel, iLevelMember*)	método construtor da classe, reescreve objetos de membros do nível de JODI (*iLevelMember*). Também recebe como parâmetro o nível da dimensão que o instanciou (*iMinerLevel*)
getParent()	retorna o membro que é o pai do membro do nível dentro de uma hierarquia
getChild(*int*)	retorna o membro filho do membro do nível pelo seu índice (parâmetro)
getChild(*String*)	retorna o membro filho do membro do nível pelo seu rótulo (parâmetro)
getChildren()	retorna o conjunto de membros filhos do membro do nível
getChildCount()	retorna a quantidade de filhos de um membro do nível

Tabela B.5 – Métodos Implementados por *cMinerMember*.

B.6 cMinerCube

A classe *cMinerCube* instancia objetos do cojunto de cubóides gerados a partir das dimensões e níveis selecionados para o cálculo de exceções. Corresponde aos termos da equação para cálculo de exceções (equação 3.1).

Nome do Método	Descrição
cMinerCube(*iMiner*, *iMinerLevel*[])	construtor da classe responsável por criar o cubóide de dados e instanciar as suas células (*iMinerCell*). Recebe como parâmetros a classe que o criou (*iMiner*) e o conjunto de níveis que formam o cubóide (*iMinerLevel*[]). Note que as suas células são geradas a partir da execução de uma consulta MDX, que distribui os níveis em colunas e linhas, e utiliza a medida definida em *iMinerMeasure*
getID()	retorna o identificador do cubo, formado pelos rótulos dos níveis que o compõe separados pelo sinal de "+"
getLevel(*int*)	retorna um dos níveis que o compõe, pelo seu índice (parâmetro)
getLevel(*String*)	retorna um dos níveis que o compõe, recebendo como parâmetro o seu rótulo
getLevels()	retorna o conjunto de níveis que compõe o cubóide
getDimensionsCount()	retorna a quantidade de dimensões que formam o cubóide
getCell(*int*)	retorna uma das células que formam o cubóide de dados. Recebe como parâmetro o índice da célula
getCell(*String*)	retorna uma das células que formam o cubóide, recebendo como parâmetro o seu rótulo, formado pelos rótulos do conjunto de membros que a compõe, separados pelo sinal "+"
getCells()	retorna o conjunto de células que formam o cubóide de dados
getCellCount()	retorna a quantidade de células que compõem o cubóide
getQuery()	retorna o texto da consulta MDX utilizada para criar o cubóide
getDrillDown(*int*)	retorna o cubóide gerado a partir da operação de *drill-down* (Seção 2.2) na dimensão de índice indicado (parâmetro)
getDrillDown(*String*)	retorna o cubóide gerado a partir da operação de *drill-down* na dimensão com o rótulo indicado (parâmetro)
getRollUp(*int*)	retorna o cubóide gerado a partir da operação de *roll-up* (Seção 2.2) na dimensão de índice indicado (parâmetro)
getRollUp(*String*)	retorna o cubóide gerado a partir da operação de *roll-up* na dimensão com o rótulo indicado (parâmetro)
getSelfExp()	retorna o valor da maior exceção encontrada nas células do cubo
getUnderExp()	retorna o valor da maior exceção encontrada em alguma célula de algum cubo gerado a partir de operações de *drill-down* no cubo atual
getPathExp(*int*)	retorna o valor da maior exceção encontrada em alguma célula de algum cubo gerado a partir de operações de *drill-down* no cubo atual ao longo da dimensão indicada pelo índice (parâmetro)
getPathExp(*String*)	retorna o valor da maior exceção encontrada em alguma célula de algum cubo gerado a partir de operações de *drill-down* no cubo atual ao longo da dimensão cujo rótulo seja igual ao passado como parâmetro

Tabela B.6 – Métodos Implementados por *cMinerCube*.

B.7 cMinerCell

A classe *cMinerCell* instancia as células dos cubóides enriquecidas por mineração.

Nome do Método	Descrição
cMinerCell(*iMinerCube*, *String*, *iMinerMember*[], *iCell*)	método construtor da classe, recebe como parâmetro o cubóide ao qual ela pertence (*iMinerCube*), o seu rótulo, as suas posições (*iMinerMember*[]) e a célula da consulta MDX que irá guardar no cache (*iCell*)
getName()	retorna o rótulo da célula (conjunto de membros separados pelo sinal "+")
getValue()	retorna o valor da célula
getFormatedValue()	retorna o valor formatado da célula
getPositions()	retorna o conjunto de posições da célula (relacionamento com os membros de cada um dos níveis)
getPosition(*int*)	retorna uma posição da célula pelo seu índice (parâmetro)
getPosition(*String*)	retorna uma posição da célula pelo seu rótulo (parâmetro)
getPositionCount()	retorna a quantidade de posições da célula, que é igual ao número de dimensões que formam o cubóide a que ela pertence
getIsNull()	retorna verdadeiro (*True*) se o valor da célula é nulo. Note que este método é importante pois ao computar as exceções, *cMiner* ignora valores nulos
getSelfExp()	retorna o grau de exceção da célula (de 0 a 3)
getUnderExp()	retorna o grau de exceção encontrado em alguma célula filho, caso realizemos uma operação de *drill-down* no cubo
getPathExp(*int*)	retorna o grau de exceção encontrado em alguma célula filho, caso realizemos uma operação de *drill-down* no cubo ao longo da dimensão com o índice indicado (parâmetro)
getPathExp(*String*)	retorna o grau de exceção encontrado em alguma célula filho, caso realizemos uma operação de *drill-down* no cubo ao longo da dimensão com o rótulo indicado (parâmetro)

Tabela B.7 – Métodos Implementados por *cMinerCell*.

Dissertação apresentada por **Fábio Moura Pereira** a Pós-Graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco, sob o título **"Um Componente de Mineração de Exceções em Cubos OLAP"** orientada pelo Prof. Jacques Pierre Louis Robin e aprovada pela Banca Examinadora formada pelos professores:

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

Prof. Ulrich Schiel
Departamento de Sistemas e Computação / UFCG

Prof. Jacques Pierre Louis Robin
Centro de Informática / UFPE

Visto e permitida a impressão.
Recife, 28 de fevereiro de 2003.

Prof. AUGUSTO CESAR ALVES SAMPAIO
Coordenador da Pós-Graduação em Ciência da Computação do
Centro de Informática da Universidade Federal de Pernambuco.