Pós-Graduação em Ciência da Computação

"Uma Solução de Metadados Baseada nos Padrões MOF e XML"

Por

Hélio Lopes dos Santos

Dissertação de Mestrado

Universidade Federal de Pernambuco

posgraduacao@cin.ufpe.br

www.cin.ufpe.br/~posgraduacao

Recife, Março/2003

UNIVERSIDADE FEDERAL DE PERNAMBUCO

CENTRO DE INFORMÁTICA

PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

HÉLIO LOPES DOS SANTOS

"Uma Solução de Metadados Baseada nos Padrões MOF e XML"

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

ORIENTADOR: DÉCIO FONSECA

CO-ORIENTADOR: ROBERTO SOUTO MAIOR DE BARROS

Agradecimentos

A minha Família, meus pais Lourival e Raquel e meus irmãos Hérmeson, Elizeu e Tatiane, pelas orações, pelo apoio moral e financeiro;

Aos irmãos da igreja em Rondonópolis, pelas orações, especialmente à congregação de Boa Vista;

A minha avó Luisa e minha tia Virgínia por me acolher durante os cinco anos de graduação em Cuiabá;

Aos meus amigos da graduação em Cuiabá Eric, Rodrigo, Wilton, Mario e Adirson;

Aos professores de graduação Cláudia, Edna e Einstein pelas cartas de recomendação e pelo incentivo a fazer este mestrado;

Ao primeiro amigo em Recife: Adimilson que muito me ajudou na escolha do tema do mestrado;

Ao amigo Roberto, por discutir assuntos relacionados à dissertação, tornando este trabalho menos solitário;

Ao professor e orientador Décio, pelo apoio acadêmico e por ter me ajudado

financeiramente nos primeiros meses aqui em Recife, conseguindo alguns "bicos", enquanto a bolsa de pesquisa não chegava;

Ao professor e co-orientador Roberto pelas suas contribuições indispensáveis;

A todos os amigos do INOCOP, especialmente a galera que conviveu e que convive no apartamento 301, Adjamir, Bruno, Thiago e Trinta;

Aos professores e amigos do CIn;

À Maísa, minha namorada pela ajuda indispensável na correção e formatação deste trabalho;

A Deus, pelas muitas providências, entre elas, a de me fazer conhecer todas essas pessoas que ajudaram durante esses dois anos de mestrado.

Resumo

Nos últimos anos, com o crescimento dos sistemas de informação, metadados tornaram-se peças chave no gerenciamento de todo o ciclo de vida desses sistemas. Muitos esforços recentes, tanto das áreas acadêmicas quanto da indústria, estão sendo concentrados em pesquisas relacionadas a metadados. Esses estudos tentam definir metodologias e padrões para construção e interoperabilidade de sistemas de informação baseados em metadados.

Atualmente, metadados são utilizados em diversas áreas, como *Data Warehouse*, bibliotecas eletrônicas, engenharia de software e integração de aplicações heterogêneas. Com o advento da internet, novas aplicações surgem, juntamente com novos padrões para representação. Esses padrões incluem XML (*Extensible Markup Language*), DTD (*Document Type Definition*), XML Schema, RDF (*Resource Description Framework*), RDF Schema, XSLT (*Extensible Stylesheet Language Transformation*), entre outros. Cada padrão foi desenvolvido para determinados tipos de aplicação. Por exemplo, DTD e XML Schema são padrões para descrição de estruturas de dados, RDF é utilizado para descrição de recursos como páginas Web enquanto que XSLT é utilizado na transformação de documentos XML.

A especificação de metadados, aliada à sua gerência eficiente, são os grandes desafios de um ambiente de construção de um *Data warehouse*. Este trabalho tem como objetivo a construção de uma solução de metadados para o ambiente REDIRIS (*Research Environment on Data Integration, Reuse and Quality in Information Systems*). O REDIRIS propõe auxiliar os usuários nas tarefas de análise, modelagem, construção e reuso de solução de *Data warehouse*. O módulo de metadados será responsável por gerenciar toda a informação necessária para a funcionalidade e gerenciamento do ambiente. Para isto, foram projetados e implementados vários metamodelos baseados nos padrões, citados anteriormente. Foi utilizado o MOF (*Meta Object Facility*) para a construção dos metamodelos e JMI (*Java Metadata Interface*) para a implementação dos mesmos.

As vantagens deste tipo de abordagem são um conjunto de interfaces comuns para gerenciamento dos metadados representados em qualquer padrão (XML, DTD, RDF) e um formato único para intercâmbio desses metadados, o XMI (*XML Metadata Interchange*).

Palavras chave: metadados, metamodelos, metamodelagem, XML, XSLT, MOF, RDF.

Abstract

In the last few years, with the increasing importance of information systems, metadata has become a key issue in management of the life cycle of these systems. Many recent efforts, both in academic areas and in industry, have concentrated in researchs related to metadata. These studies try to define methodologies and standards for the construction and interoperability of information systems based on metadata.

Nowadays, metadata are used in many areas, like Data warehouse, electronic libraries, software engeneering and integration of heterogeneous applications. With the Internet, new aplications have surged, together with new standards for representation. These new standards include XML (Extensible Markup Language), DTD (Document Type Definition), XML Schema, RDF (Resource Description Framework), RDF Schema, XSLT (Extensible Stylesheet Language Transformation), and others. Each of these standards was developed for specific classes of applications. For example, DTD and XML Schema are standards for data structure description; RDF is used for describing resources as Web pages and XSLT is used in XML documents transformation.

The metadata especification, as well as its efficient management, are the great challenge of a Data warehouse construction environment. This work has as objective the construction of a metadata solution for the REDIRIS (Research Environment on Data Integration, Reuse and Quality in Information Systems) enviroment. The REDIRIS proposes to assist the users in the analysis tasks, modeling, construction and reuse of Data warehouse solutions. The metadata module will be responsable to manage all the necessary information for the functionality and management of the environment. To achieve this, many metamodels based on the aforementioned standards were projected and implemented. The MOF (Meta Object Facility) was used for the metamodels construction and JMI (Java Metadata Interface) for their implementation.

The advantages of this approach are a set of common interfaces for the management of the metadata represented in any standard (XML, DTD, RDF) and a unique format for the interchange of these metadata, the XMI (XML Metadata Interchange).

Key Words: metadata, metamodels, metadamodeling, XML, XSLT, MOF, RDF.

Sumário

Índice de Figuras

Índice de Listagens

Índice de Tabelas

Lista de Siglas

API	Application Programming Interface
CORBA	Common Object Request Broker Architecture
DTD	Document Type Definition
ETL	Extraction, Transformation and Loading
IDL	Interface Definition Language
JMI	Java Metadata Interface
MDR	Metadata Repository
MOF	Meta Object Facility
OCL	Object Constraint Language
OLAP	On Line Analytical Processing
OMG	Object Management Group
RDF	Resource Description Framework
REDIRIS	Reuse Environment on Data Integration, Reuse and Quality in Information Systems
SAD	Sistemas de Apoio à Decisão
SGBD	Sistemas de Gerenciamento de Banco de Dados
UML	Unified Modeling Language
XML	eXtensible Markup Language
XMI	XML Metadata Interchange
XSLT	Extensible Stylesheet Language Transformation
W3C	World Wide Web Consortium

1 Introdução

1.1 Motivação

O tema metadados não é novo, porém, o escopo de sua atuação está em constante mudança conforme a evolução dos sistemas de informação. Originalmente, metadados se referiam aos modelos de dados, por exemplo, os esquemas de banco de dados, as informações que descreviam os formatos dos registros, como nomes dos campos, tipos, tamanhos dos campos, entre outros.

Atualmente, metadado é utilizado em diversas áreas, como *Data warehouse*, bibliotecas eletrônicas, Sistemas de Informações Geográficas, engenharia de software e integração de aplicações heterogêneas. Essas novas aplicações trouxeram novos desafios para os metadados, que geraram novas pesquisas e definições de novos padrões [HJE2001, PCTM2001, XMI2000]. São exemplos desses novos desafios: acomodar múltiplos formatos de dados; gerenciar a evolução dos formatos atuais e os novos formatos que surgem; ser interpretado pelo computador e por pessoas ao mesmo tempo, e dar suporte à criação, administração e acesso aos dados por comunidades de usuários específicas.

O uso de metadados envolve muitos tipos diferentes de sistemas trabalhando juntos [AHAY2001], por exemplo, Sistemas de Gerenciamento de Banco de Dados, Ferramentas de *Data warehouse*, Sistemas CAD (*Computer Aided Design*), entre outras. Cada uma dessas tecnologias tem as suas próprias características e padrões para a representação de metadados.

Os padrões do W3C (*World Wide Web Consortium*) como XML [ABK2000], XML Schema [FALL2002], RDF [LASW1999, HJE2001, BRAY1998] são padrões para dados semi-estruturados e flexíveis, voltados para o contexto Web, suportados por uma variedade de

ferramentas, e que podem ser utilizados como formatos de descrição, armazenamento e intercâmbio de metadados.

A OMG (*Object Management Group*) possui um outro conjunto de padrões como o MOF [MOF1999] e XMI [XMI2000], que foram construídos especificamente para dar suporte à modelagem, gerenciamento e intercâmbio de metadados e estão voltados ao contexto de sistemas de informação. Estes padrões possuem como uma de suas características principais permitir interoperabilidade entre ferramentas de software que produzem e compartilham metadados.

Uma abordagem que integre estes padrões poderia aproveitar a flexibilidade dos padrões do W3C e a interoperabilidade dos padrões da OMG para o gerenciamento de metadados.

1.2 Objetivos do Trabalho

Este trabalho tem como objetivo construir uma solução adequada de metadados, voltada inicialmente para o contexto de SAD (Sistema de Apoio à Decisão), dentro do ambiente REDIRIS (*Research Environment on Data Integration, Reuse and Quality in Information Systems*). O REDIRIS é um ambiente proposto para auxiliar os usuários nas tarefas de análise, modelagem, construção e reuso de solução de *Data warehouse*. Reutilizar soluções pré-existentes, ao invés de criar conceitos e arquiteturas muitas vezes similares é uma alternativa econômica, mas não tem sido tema de pesquisas em SAD [VASS2000]. As técnicas de reuso possibilitam exatamente esse aproveitamento de experiências. O processo envolve a identificação de objetos, operadores e relações que compõem o domínio de uma aplicação, ou família de aplicações. Este tema "Reuso para SAD" foge ao escopo deste trabalho e deverá ser tratado em uma outra dissertação.

Os objetivos específicos deste trabalho são os seguintes:

- Realizar um levantamento do estado-da-arte e aprofundar conhecimentos acerca das soluções de metadados e dos padrões utilizados como MOF, CWM [CWM2000, TOLB2000], XMI, XML, DTD [ABK2000], XSLT [CLA1999, KAY2002], RDF e RDF Schema [BRGU2000, HJE2001];
- Projetar um metamodelo para o padrão XML utilizando o MOF;
- Projetar um metamodelo para o padrão DTD utilizando o MOF;

- Projetar um metamodelo para o padrão XSLT utilizando o MOF;
- Projetar um metamodelo para o padrão RDF utilizando o MOF;
- Projetar um metamodelo para o padrão RDF Schema utilizando o MOF;
- Implementar um protótipo de uma solução de metadados utilizando os metamodelos, citados anteriormente, para o ambiente REDIRIS;
- Avaliar a implementação e o grau de generalidade da solução proposta.

1.3 Visão Geral do Trabalho

Esta dissertação está distribuída em 8 capítulos, incluindo a introdução. Cada capítulo aborda os seguintes assuntos:

Capítulo 2 – Aborda o estado da arte sobre metadados, trabalhos relacionados e uma perspectiva histórica sobre o tema. Logo após, são discutidos tópicos sobre metadados para o gerenciamento de *Data warehouse*. Este capítulo aborda também o conceito de solução de metadados e as suas categorias. E por último, apresenta o ambiente REDIRIS;

Capítulo 3 – Aborda alguns padrões do W3C e OMG que podem ser utilizados para representar, armazenar e descrever metadados. A abordagem começa pelos padrões do W3C que são XML, DTD, XSLT, RDF e RDF Schema, e termina discutindo alguns padrões da OMG para metadados, como o MOF, CWM e XMI. Todos esses padrões podem ser utilizados para representar metadados.

Capítulo 4 – Aborda o padrão para metamodelagem MOF. O MOF foi utilizado na modelagem e implementação dos metamodelos da solução de metadados. É realizada uma descrição do que é o MOF, a sua arquitetura e quais os tipos de aplicações. Logo após é realizada uma breve descrição do modelo MOF e dos seus construtores. São descritos também, o padrão de mapeamento entre os metamodelos MOF e API (*Application Programming Interface*) [MOF1999] e as interfaces reflexivas do MOF.

Capítulo 5 – Apresenta uma parte da solução, a abordagem utilizada e os passos utilizados na construção e implementação dos metamodelos. É apresentada uma descrição detalhada da construção do metamodelo XML até a sua implementação em um repositório MOF.

Capítulo 6 – Apresenta os outros metamodelos que fazem parte da solução. São eles, o metamodelo DTD, utilizados para gerenciar DTD; o metamodelo XSLT, utilizado para

gerenciar documentos XSLT; o metamodelo RDF, utilizado para gerenciar documentos RDF e o metamodelo RDFS, utilizado para gerenciar documentos RDF Schema. Todos esses metamodelos foram modelados segundo as suas respectivas especificações.

Capítulo 7 – Apresenta um estudo de caso com dois exemplos, ilustrando como o repositório poderá ser utilizado por outras ferramentas para o gerenciamento de metadados descritos pelos padrões modelados. O primeiro exemplo apresentado se refere ao gerenciamento dos metadados gerados pela metodologia *Fast Cube* do ambiente REDIRIS. Neste exemplo, são utilizados os metamodelos XML, DTD e XSLT. O segundo exemplo utiliza o próprio metamodelo RDFS para armazenar o documento que descreve o padrão RDF Schema.

Capítulo 8 – Apresenta as conclusões, as contribuições e os trabalhos futuros.

2 Estado da Arte

Este capítulo discute alguns trabalhos relacionados. Aborda o tema metadados, abrangendo a sua evolução e o uso do mesmo para gerenciamento de *Data warehouse*. Discute o que é uma solução de metadados e as suas categorias e, no final, apresenta o ambiente REDIRIS. Este ambiente é o principal cliente da solução de metadados proposta por esta dissertação.

2.1 Trabalhos Relacionados

Em linhas gerais, os tópicos de pesquisas abordam metadados em relação a aplicações específicas. Por exemplo, Marino [MAR2001] propõe um metamodelo baseado no padrão RDF para descrever documentos científicos na Web. Gonçalves [GON1999] apresenta uma proposta de um metamodelo conceitual para a representação de metadados para Sistemas de Informações Geográficas, que utiliza a arquitetura de metadados da OMG (veja seção 4.1); porém, não utiliza o MOF como *framework* [BCS1999] para metamodelagem e gerenciamento dos metadados. Barreto [BAR1999] propõe um metamodelo para descrição de documentos eletrônicos, que denominou de MODDE (Modelo de Objetos Digitais para Documentos Eletrônicos). Este metamodelo é baseado na arquitetura *Warwick*. O padrão RDF foi baseado na arquitetura *Warwick* [HJE2001]. Silva [SIL2000] propõe um metamodelo que é uma extensão do MODDE para bibliotecas digitais.

Esta dissertação propõe um conjunto de metamodelos que darão suporte à representação e gerenciamento de metadados XML, DTD, XSLT, RDF e RDF Schema, em repositórios MOF. Desta forma, qualquer aplicação que deseje representar metadados nestes padrões poderá utilizar os metamodelos.

Outros metamodelos baseados no padrão MOF são o UML (*Unified Modeling Language*)

[UML2001] para modelagem de sistemas, o CWM (*Common Warehouse Metamodel*) [CWM2001], que propõe um conjunto de metamodelos para representação de metadados em ambientes de *Data warehouse*. O CWM será mostrado no Capítulo 3. Uma outra iniciativa importante da empresa *Sun Microsystem* é um metamodelo para a linguagem Java [DEMA2002], que propõe uma maneira de gerenciar código de programas Java em repositórios MOF.

2.2 Metadado, modelo e metamodelo

Estes termos serão muito utilizados nesta dissertação e é importante saber qual a relação entre cada termo. O termo "meta" é o papel que um determinado dado ou modelo desempenha em relação a outro dado ou modelo.

Metadado é definido como sendo um dado que descreve outro dado, incorporando a este, significado. Sem metadado, a informação se restringe a um conjunto de dados sem significado. Metadados podem ser formais ou informais. Os formais podem ser interpretados por máquinas e os informais servem apenas para leitura por humanos. Eles atuam também nos diversos níveis de abstração, do físico ao conceitual [SMVV2000]. São utilizados nas diversas áreas de sistemas de informação como bibliotecas digitais, integração de aplicações, *Data warehouse*, descoberta de recursos na Web, engenhos de busca, entre outras [DMOF2001]. Eles podem descrever:

- Qualquer aspecto do dado alvo que é importante para a aplicação;
- O dado alvo em qualquer linguagem abstrata. Esta linguagem pode ser formal ou informal, precisa ou imprecisa, estruturada ou não estruturada;
- O dado alvo no grau de detalhe requerido pela aplicação cliente.

O termo modelo pode ser definido como um conjunto de metadados inter-relacionados que descrevem um conjunto de dados alvo. Um modelo é uma descrição abstrata de um sistema ou de um processo, uma representação simplificada que possibilita o entendimento e simulação.

Um metamodelo define um modelo para construir modelos. Ele é um conjunto de metadados inter-relacionados utilizados para definir modelos. O metamodelo descreve formalmente os elementos do modelo, a sintaxe e a semântica das notações que permitem sua manipulação. Alguns conceitos que podem ser encontrados em um metamodelo são por exemplo: "Classe",

"Processo" e "Método".

Um Meta-metamodelo é uma linguagem abstrata para definição de metamodelos. Ele está relacionado para um metamodelo da mesma forma que um metamodelo para um modelo.

Este capítulo concentra-se no estudo de metadados na área de *Data warehouse,* pois é o tema que está diretamente relacionado com o ambiente REDIRIS.

2.3 A evolução do termo metadado

Como mostra a Figura 2.1, o conceito de repositório de metadados não é novo. A sua origem data de meados da década de 1970 [MAIN2000]. Os primeiros repositórios de metadados eram chamados de dicionários de dados. Esses dicionários de dados proviam informações sobre os dados, tais como as suas definições, relacionamentos, origem, domínio, uso e formato. Sua proposta era auxiliar os administradores de bancos de dados (*Database Administrators* - DBAs) no planejamento, controle e avaliação das coleções, armazenamento e uso dos dados. Por exemplo, os dicionários de dados eram usados principalmente para registrar os requisitos da aplicação, modelagem dos dados da corporação, geração das definições dos dados e suporte ao banco de dados.

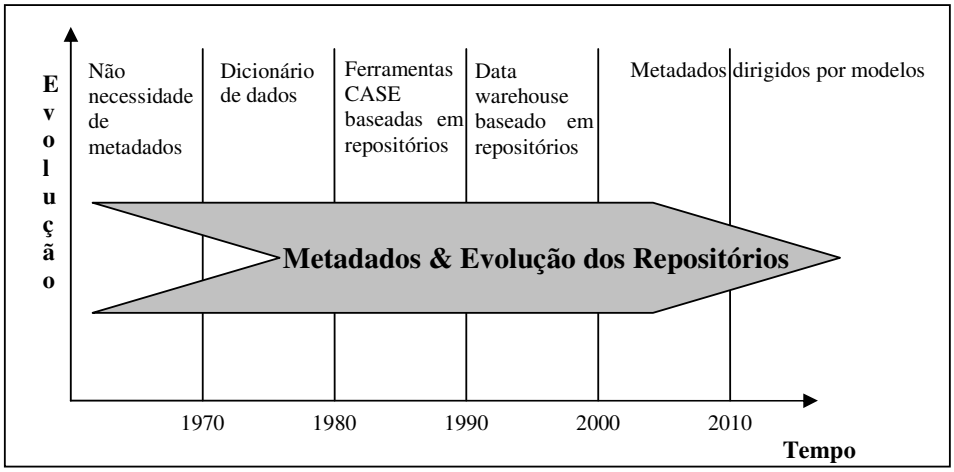

Figura 2.1 – A evolução dos repositórios de metadados [MAIN2000]

As ferramentas CASE (*Computer Aided Software Engineering*) surgiram em 1970 e foram as primeiras ferramentas a oferecer serviços de metadados [MAIN2000]. Elas auxiliavam no processo de desenvolvimento de aplicações de banco de dados e armazenavam informações sobre os dados que elas gerenciavam. Mas até então essas ferramentas não possuíam uma

interface padrão para comunicação entre as mesmas.

Em 1987, com a necessidade de integração entre ferramentas CASE, a EIA (*Electronic Industries Alliance*) iniciou um trabalho para a definição de um formato de intercâmbio de metadados entre ferramentas CASE que chamou de CDIF (*Case Data Interchange Format*). Esse padrão tornou-se muito importante para a indústria de ferramentas CASE.

Durante os anos 80 várias companhias, inclusive a IBM, desenvolveram ferramentas de repositórios de metadados para equipamentos mainframe. Esses esforços eram as primeiras iniciativas na área de ferramentas de repositório. Porém, o escopo era ainda limitado ao gerenciamento de metadados técnicos e a maioria delas ignorava os metadados de negócio (veja a seção 2.4.5 - metadados técnicos e de negócio).

Muitos desses primeiros repositórios eram utilizados apenas como dicionários de dados com o propósito, como os primeiros dicionários de dados, de serem usados pelos DBAs e projetistas de dados. E no mais as companhias que criaram esses repositórios educaram pouco seus usuários sobre os benefícios dessas ferramentas. Como resultado, poucas companhias viram o valor desses primeiros repositórios de metadados.

No início dos anos 90, as corporações começaram a reconhecer o valor dos repositórios de metadados. Neste momento essas aplicações já funcionavam em ambiente cliente-servidor. As ferramentas de sistema de suporte a decisão requeriam acesso a um repositório de metadados. A partir desse momento várias empresas começaram a desenvolver essas ferramentas.

Atualmente, os metadados dirigidos por modelos, também conhecido como abordagem de integração baseada em modelos, consiste na integração e compartilhamento de metadados entre ferramentas de software em formas de modelos independentes de plataforma [PCTM2001]. Todas as ferramentas, aplicações e bancos de dados que participam da integração concordam com um metamodelo comum, definido para um domínio específico. Assim, produtos de software podem ler e disponibilizar metadados, ou seja, instâncias de metamodelos, no padrão convencionado. Cada produto mapeia este padrão para a sua própria representação interna, que é independente. Estes padrões especificam interfaces entre a forma de representação interna dos metadados das ferramentas e o ambiente externo.

A OMG adota esta estratégia de integração, através do MDA (*Model Driven Architecture).* O MDA é uma abordagem de especificação de sistemas e interoperabilidade baseados em modelos formais [POOL2001, MDA2002, MDA2001]. O MDA tem como base alguns

padrões de integração e interoperabilidade como o MOF, XMI e UML também da OMG, baseados no gerenciamento de metadados independente de plataforma. Esses padrões são descritos nos Capítulos 3 e 4.

A seção seguinte discute o uso e a importância de metadados para o gerenciamento de *Data warehouse*.

2.4 Metadados para o gerenciamento de Data warehouse

Um ambiente de *Data warehouse* depende efetivamente de uma boa infra-estrutura de metadados [HARA2000, IWG1999]. Esses metadados descrevem os aspectos relevantes dos dados e processos do *Data warehouse.* Os dados que são obtidos de fontes heterogêneas são tratados e integrados em um repositório e disponibilizados para consultas através das ferramentas OLAP (*On Line Analytical Processing*) [CHDA1997] ou ferrramentas de *Data mining*. Todo o processo de tratamento e integração dos dados, também chamado de ETL (*Extraction, Transformation and Loading*), requer uma grande quantidade de metadados que descrevem os dados nas suas fontes de origem e o processo de tratamento e integração do mesmo ao *Data warehouse*. Os dados tratados e integrados no *Data warehouse* também são descritos por metadados para serem disponibilizados para consultas ao usuário final, através das ferramentas *OLAP* e *Data mining* [KIM1998].

Os metadados de um *Data warehouse* servem a dois propósitos principais [IWG1999]:

- Minimizar o esforço de desenvolvimento e administração – isto envolve a integração das fontes de dados de origem, suporte à análise e projeto de novas aplicações, prover a flexibilidade e reuso dos modelos de software existentes, automatizar os vários processos administrativos do *Data warehouse*, entre outros;
- Oferecer as informações aos usuários finais – isto envolve garantir a qualidade dos dados; facilitar a interação com o sistema de *Data warehouse*, prover a análise dos dados, entre outros.

Um ambiente típico de *Data warehouse* é apresentado na Figura 2.2. Ele consiste de várias fontes de dados em diferentes formatos, várias ferramentas, provavelmente desenvolvidas por diferentes fornecedores, entre outros. Estes componentes são citados a seguir:

- *Fontes operacionais* – são as fontes de origem dos dados. Elas provavelmente estarão representadas em diversos formatos e armazenadas em diversas plataformas.

Normalmente são sistemas de arquivos, Sistemas de Gerenciamento de Banco de Dados.

- *Data warehouse e Data marts* [KIM1998] – São os locais onde estarão armazenados os dados que darão suporte às decisões da empresa. Os *Data marts* também podem ser categorizados como *Data warehouses* departamentais, ou seja, dão suporte apenas a uma parte específica da empresa, por exemplo, vendas, estoque, entre outros.
- *Ferramentas de ETL* – realizam as tarefas de ETL, ou seja, são as ferramentas que extraem os dados das fontes, fazem as transformações e tratamento adequados e carregam esses dados para o *Data warehouse*.
- *Ferramentas de modelagem* – são as ferramentas que modelam os dados do *Data warehouse* e dos *Data marts.*
- *Ferramentas de acesso* – são as ferramentas que disponibilizam os dados do *Data warehouse* e do *Data mart* para os usuários consultarem. Podem ser ferramentas de consulta e relatório, *OLAP* ou de *Data mining*.

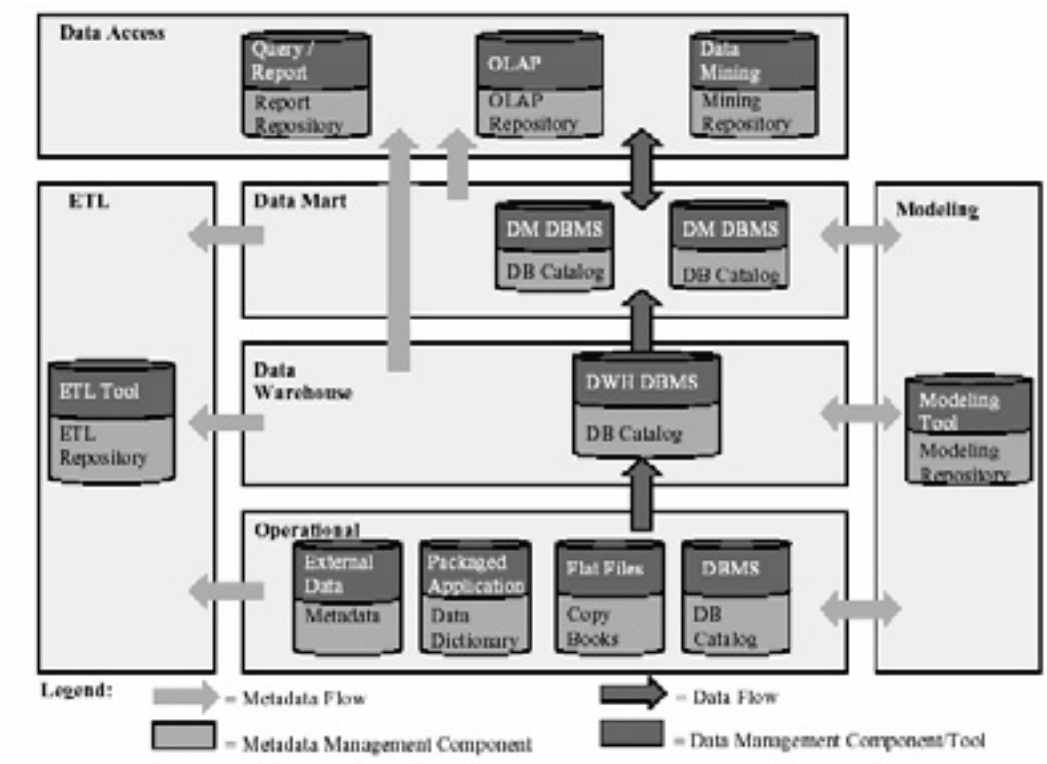

Figura 2.2 - Um típico ambiente de Data warehouse [HARA2000]

Todos os componentes da Figura 2.2 criam e mantêm metadados, por exemplo, catálogo de banco de dados, dicionário de dados, metadados gerados por cada ferramenta específica. Os metadados são um dos aspectos mais importantes em um ambiente de *Data warehouse,* conforme descrito nas subseções seguintes.

2.4.1 O mapeamento das Fontes Operacionais

Uma das tarefas dos metadados em um ambiente de *Data warehouse* é o mapeamento dos dados entre as fontes operacionais e o *Data warehouse*, conforme apresentado na Figura 2.3 [KIM1998, INM2000].

O mapeamento inclui uma extensa variedade de tipos, tais como:

- *O mapeamento de um atributo para outro* – mapeia os atributos das fontes de origem para os atributos do esquema do *Data warehouse*;
- *Conversões* – por exemplo, se o valor distância é armazenado na fonte de origem na unidade polegada e no *Data warehouse* precisa ser armazenado em centímetros;
- *Mudanças de conversões de nomes*;
- *Mudanças de características físicas dos dados* – por exemplo, mudanças de atributos do tipo inteiro para reais ou vice-versa.

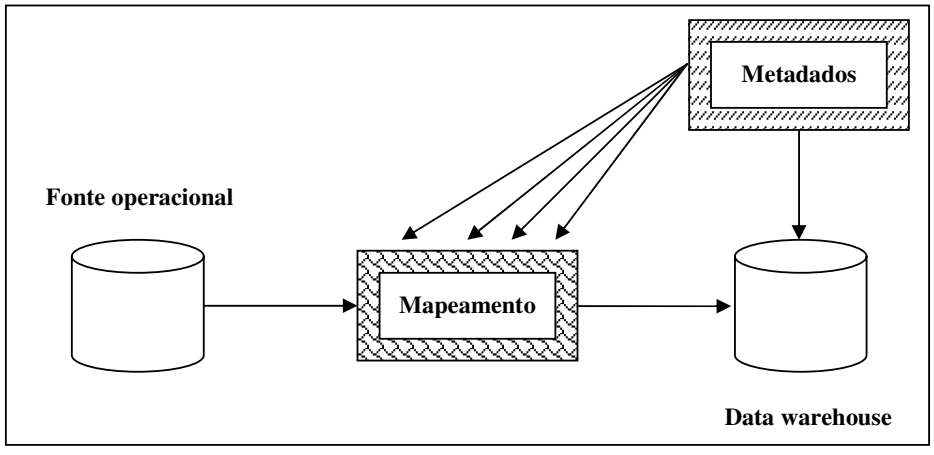

Figura 2.3 – Metadados para o mapeamento dos dados das fontes de origem [INM2000]

2.4.2 Gerenciamento de dados históricos e versionamento

Uma das características de um *Data warehouse* é que ele contém dados que foram produzidos durante anos, ou seja, são dados históricos [HARA2000, KIM1998]. Portanto, com o passar

do tempo, provavelmente, as estruturas das fontes operacionais mudarão. São exemplos de mudanças: uma tabela que mudou a chave primária, um atributo que mudou de nome, um banco de dados que foi reestruturado para suportar as novas regras da organização, entre outras.

O registro das estruturas originais das fontes de dados e as mudanças ocorridas com o passar do tempo devem ser armazenados como metadados no *Data warehouse*. A Figura 2.4 apresenta a importância dos metadados para o gerenciamento de dados históricos.

As múltiplas estruturas de dados para um *Data warehouse* contrastam com uma estrutura de dados apenas das fontes operacionais. Isto é uma das diferenças fundamentais entre um *Data warehouse* e um ambiente operacional [INM2000].

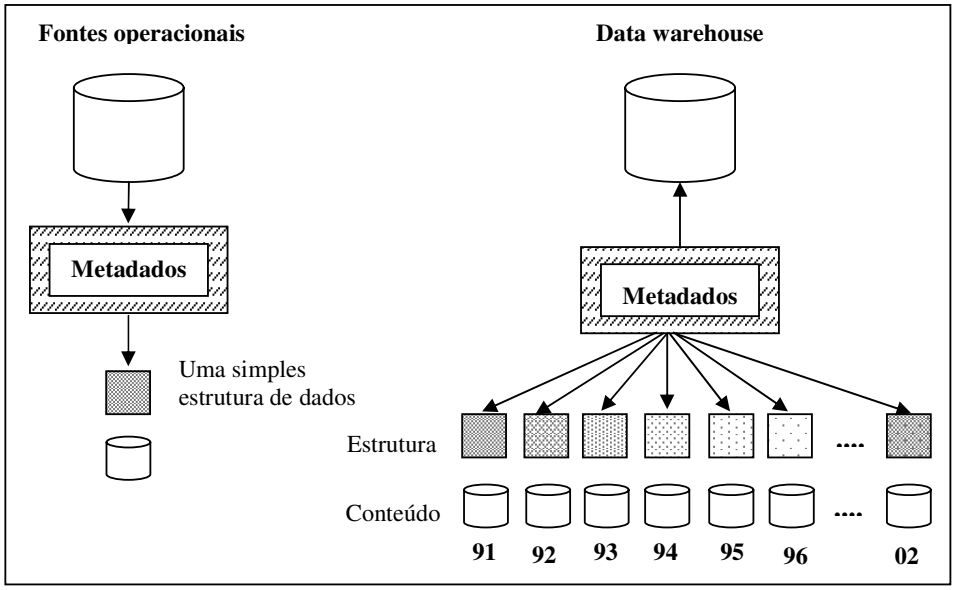

Figura 2.4 – Metadados para gerenciamento dos dados históricos [INM2000]

2.4.3 Gerenciamento das rotinas de ETL (*Extraction, Transformation and Loading*)

As rotinas ETL realizam o processo de extração dos dados das fontes, transformam os mesmos e os carrega para o *Data warehouse*. Este processo tem impacto direto na qualidade dos dados do *Data warehouse*. A qualidade dos dados é fundamental para o sucesso do *Data warehouse*. Quando a qualidade dos dados está comprometida, o uso ou interpretação incorretos da informação do *Data warehouse* podem acabar com o nível de confiança dos usuários e comprometer a própria existência do sistema.

Os metadados gerados pelas rotinas de ETL são importantes para o gerenciamento do processo e garantir a qualidade dos dados. A Figura 2.5 apresenta este processo.

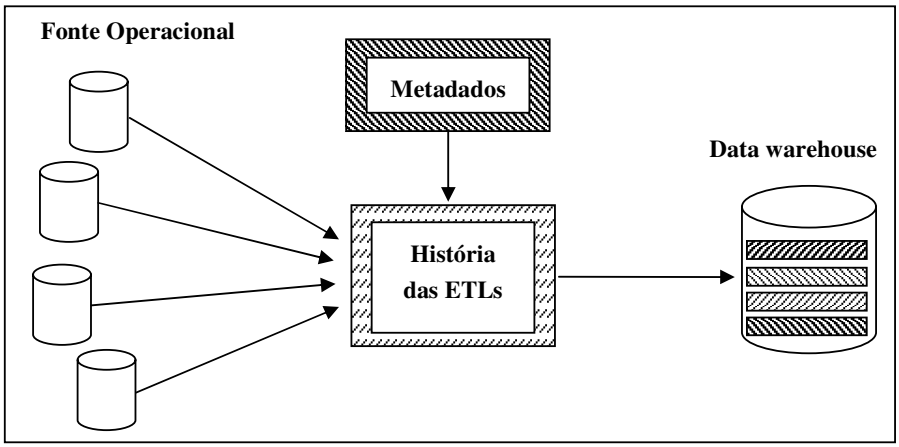

Figura 2.5 – Metadados gerados pelas rotinas ETLs [INM2000]

Os metadados gerados pelas rotinas de ETL dizem qual foi a última carga de dados realizada no *Data warehouse*, quando um determinado dado entrou, quais foram as transformações sofridas pelo mesmo, entre outras.

2.4.4 Outros componentes de metadados

Existe uma variedade de metadados que podem auxiliar o administrador de *Data warehouse* nas suas tarefas rotineiras. Alguns deles estão apresentados na Figura 2.6.

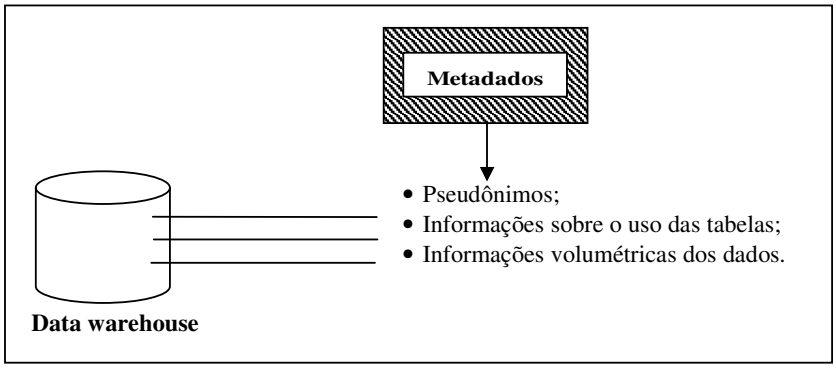

Figura 2.6 - Outros componentes de metadados[INM2000]

Os pseudônimos (*aliases*) oferecem nomes alternativos para os atributos e tabelas do *Data warehouse*. Muitas vezes os nomes definidos pelas equipes técnicas não são tão compreensíveis para outros usuários. Um *alias* é uma alternativa para tornar o ambiente mais

amigável.

Os metadados sobre o uso das tabelas podem oferecer informações ao administrador do *Data warehouse* sobre quais tabelas estão sendo mais utilizadas que outras. Isto pode auxiliá-lo na tarefa de manutenção, como a criação de determinados índices que diminuiria o tempo das consultas.

As informações sobre volume, como por exemplo, o número de linhas, a taxa de crescimento, e os índices mais utilizados nas consultas às tabelas do *Data warehouse,* podem auxiliar o administrador nas suas tarefas diárias.

2.4.5 Classificação dos metadados

Inmon [MAIN2000] classifica os metadados em dois grupos: os técnicos e os de negócio.

Os metadados técnicos são importantes para todos os processos e dados do ambiente de *Data warehouse*. Os metadados técnicos são formados por:

- Esquema dos dados das fontes operacionais e do próprio *Data warehouse*;
- Descrição técnica sobre as fontes operacionais como, por exemplo, nome do servidor, endereço da rede, nome dos arquivos, entre outros;
- Metadados para suporte administrativo, como as informações sobre o uso dos índices, das tabelas, informações volumétricas dos dados, entre outros.

Os metadados de negócio dão suporte ao usuário final no entendimento dos significados dos dados. Estes metadados são formados por:

- Modelos conceituais dos dados – são modelos independentes de tecnologia, normalmente são modelos UML. Estes modelos podem ser tanto das fontes operacionais quanto do *Data warehouse*;
- Definição dos termos do negócio – Estes metadados descrevem em linguagem natural as entidades que são relevantes para o negócio. Os usuários que tomam as decisões devem estar familiarizados com estes termos;
- Mapeamento entre as definições dos termos do negócio e os elementos do *Data warehouse* como tabelas e atributos;
- Informações sobre o estado e a qualidade dos dados do *Data warehouse*. Estas

informações são obtidas a partir do metadados técnicos;

Existem outras formas de classificação. Cada autor define a sua forma, isto se deve também ao foco dado ao estudo em questão. A Tabela 2.1 apresenta algumas formas de classificação [SMVV2000].

Tabela 2.1 - Classificação dos metadados

Classificação	Categorias
Tipos de aplicação dos metadados	• Para dados primários – são os metadados referentes às estruturas das fontes operacionais e do *Data warehouse*; • Para processos – São metadados provenientes das rotinas de ETL, rotinas de *backup*, entre outras;
Nível de abstração	• Conceitual – são os metadados que descrevem os termos do negócio; • Lógico – são os metadados que fazem o mapeamento entre os conceituais e os físicos; • Físico – são os metadados do nível de implementação.
Origem – Quem produziu	• Ferramentas – São os metadados gerados de forma automática por ferramentas ETL, CASE, entre outras; • Usuários – São os metadados que o usuário constrói, como os modelos de dados.
Propósito – refere-se a atividade fim do metadado.	• Extração – são os metadados utilizados para realizar o processo de extração dos dados das fontes operacionais; • Carga – são os metadados utilizados para realizar o processo de carga dos dados do *Data warehouse*; • Administração – são os metadados utilizados na administração do *Data warehouse*; • entre outros.
Tempo de Produzir/Consumir	• Tempo de Projeto - São os metadados produzidos ou consumidos nos projetos das aplicações, por exemplo os esquemas das fontes, regras de transformação, direitos de acesso, entre outros; • Tempo de Construção – São os metadados produzidos ou consumidos durante a construção das aplicações. No contexto de *Data warehouse* isto se refere às rotinas ETL, onde os dados são extraídos das fontes, transformados e carregados para o *Data warehouse*. São exemplos, arquivos de *log*, atributos sobre a qualidade dos dados, entre outros; • Tempo de Uso – São os metadados produzidos ou consumidos durante o uso das aplicações. Por exemplo, estatísticas das consultas realizadas pelos usuários.

2.5 Soluções de Metadados

Uma solução de metadados é um conjunto de metadados organizados e integrados, conectados logicamente por pontos de acesso comuns. As bases de metadados podem ser mais de uma, distintas ou com um metamodelo comum, e devem estar disponíveis tanto para os elementos internos da solução quanto para os externos. Uma solução de metadados é composta por vários componentes: os metadados; um ou mais metamodelos que definirão como estes serão armazenados e os seus relacionamentos; o banco de dados para o armazenamento; e o software que gerencia. O principal cliente de uma solução de metadados são outras

ferramentas, diferente da maioria dos sistemas de informação cujo principal cliente são os usuários [TANN2002].

As implementações de sistemas de metadados servem a outras aplicações e variam com a necessidade de cada organização. A literatura [MAIN2000, TANN2002] apresenta vários exemplos de implementações, cada uma adequada para um certo tipo de necessidade. Esta seção tem como objetivo mostrar algumas categorias de soluções de metadados e as suas características.

Repositório Centralizado - Em um repositório centralizado, todos os metadados são capturados, organizados e armazenados em um único local. Os usuários acessam e recuperam metadados diretamente através das interfaces definidas pelo repositório. Um típico repositório centralizado deve ser flexível o bastante para permitir armazenar os metadados em uma variedade de formatos, produzidos por uma variedade de ferramentas, como documentos em formato texto, banco de dados, ferramentas CASE (*Computer Aided Software Engineering*) e aplicações legadas.

Repositório Integrado - Nesta categoria, toda a solução representa um conjunto de ferramentas que acessam uma ou mais bases de metadados. Cada base pode ter o seu próprio metamodelo, mas todas estão integradas através de informações que relacionam todos os metamodelos.

Diretório de Informações - Neste tipo de solução, o objetivo principal não é a integração, mas apenas identificar e armazenar os metadados sem se preocupar com redundâncias. Os metadados são categorizados por assunto ou área, o que facilita a sua recuperação.

Repositório Standalone - É similar a uma solução centralizada simples. Ela captura, armazena e distribui os metadados, com uma diferença na distribuição. Esta última é realizada através de consultas que os usuários fazem aos metadados, e estes são emitidos em formas de relatórios.

Enterprise Portal - É uma abordagem que utiliza meta-metadata, ou seja, informações sobre os metadados para encontrar os metadados. Um *Enterprise Portal* integra logicamente diversos repositórios de metadados e disponibiliza um ponto de acesso comum a todos os metadados.

2.6 Uma Solução para o Ambiente REDIRIS

Após a apresentação dos diversos tipos de solução, esta seção apresenta o ambiente REDIRIS *(Research Environment on Data Integration, Reuse and Quality in Information Systems)*, o principal ambiente da solução de metadados proposta nesta dissertação. Serão apresentadas as suas principais características.

2.6.1 O Ambiente REDIRIS

O ambiente REDIRIS foi concebido para ajudar os usuários e os projetistas nas tarefas de análise, modelagem e construção de solução de *Data warehouse*, também conhecido como SAD (Sistema de Apoio à Decisão). Ele combina reuso, metadados e técnicas de integração e análise em uma abordagem original para tratar o problema de construção e uso de sistemas de suporte à decisão eficientes.

Os projetistas de *Data Warehouse* precisam utilizar-se de técnicas apropriadas para desenvolvimento e implantação de Sistemas de Informação, porque as organizações demandam cada vez mais informações de qualidade e cada vez mais rápido. Esta demanda só pode ser suprida por um ambiente voltado para esse fim, construído e/ou adaptado em tempo hábil, utilizando sempre o conhecimento adquirido em implementações anteriores.

2.6.1.1 – Princípios Básicos

Alguns princípios básicos regem a concepção e a construção do ambiente REDIRIS. A seguir são apresentados estes princípios.

Primeiro Princípio. Qualidade como o maior fator de sucesso, desde os processos de captura e análise dos dados até a entrega de uma nova solução de suporte à decisão.

Segundo Princípio. A dinâmica de um SAD requer velocidade, respostas confiáveis, que pedem a implementação de um esquema de reuso eficiente.

Terceiro Princípio. O ciclo de desenvolvimento de um SAD deve perseguir a interatividade, características dinâmicas e incrementais, e requer avaliação durante todos os estágios do processo, com uma atenção especial para as mudanças estratégicas que podem ocorrer dentro da organização.

Quarto Princípio. A especificação de metadados e domínios de aplicação, aliados com a sua gerência eficiente, são os grandes desafios de um ambiente de construção de um SAD.

Acoplado a estes princípios, o REDIRIS introduz técnicas de inteligência artificial

[CHEN2001, PYL1999] ao ciclo de desenvolvimento de um SAD como um caminho para conseguir objetivos na qualidade de dados. Entre outros aspectos, essas técnicas relacionadas a aplicações de indução de regras para descobrimento e avaliação do conhecimento de negócio, casamento e correlação de algoritmos (*matching*) para tratar anomalias de dados, algoritmos de associação para identificação de hierarquias dimensionais, e métodos estáticos para desenvolvimento de soluções de suporte à decisão de alta qualidade.

Considerando o problema da integração de dados, não se pode ignorar a *World Wide Web* como um principal recurso de dados prontos e disponíveis, mas de forma muito heterogênea. A semântica da integração dos metadados [VDO2001], em conjunto com semânticas ontológicas [HJE2001], formarão a base dessa futura arquitetura. O REDIRIS foca no desenvolvimento do processo semi-automático para semântica de integração do esquema, baseado em padrões XML e RDF [LASW1999].

2.6.1.2 – A Arquitetura do REDIRIS

A arquitetura REDIRIS é apresentada na Figura 2.7. O componente de captura e análise responde aos dois processos sobre a captura de aspectos herdados de uma aplicação de domínio, e a análise de sistemas legados para avaliar seus potenciais para reuso e qualidade de dados operacionais.

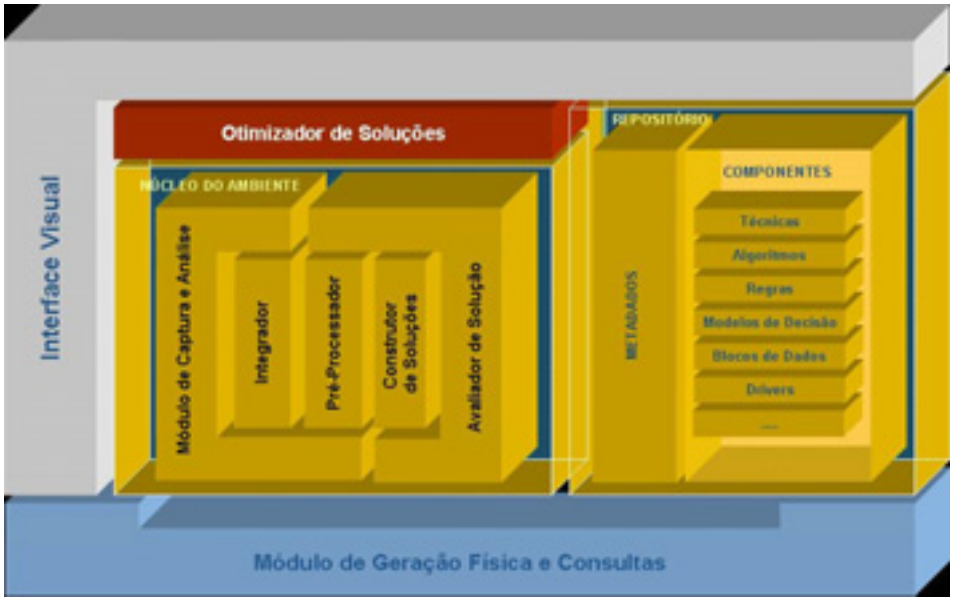

Figura 2.7 – Arquitetura do ambiente REDIRIS

Acoplado com as funcionalidades anteriores, o Pré-Processador executa a tarefa importante de

promover técnicas de análise de dados previamente descritas para ajudar na avaliação de seu potencial de suporte à decisão interno. A integração de Dados é um dos mais importantes requerimentos do Pré-Processador. O módulo fornece auxílio na integração dos dados que estão distribuídos em diversos formatos, integrando estas informações em um formato do ambiente REDIRIS. Além disso, ele ajuda no trabalho de análise de dados gerando tabelas de exemplos concordando com uma granularidade pré-definida de dados. O Pré-Processador também pode ser utilizado para gerar protótipos de SAD a partir dos sistemas legados (baseados nos resultados de análise de dados), então um usuário pode explorar capacidades analíticas desencadeadas já descobertas. Além disso, protótipos criados, assim como soluções pré-existentes podem também ser carregadas dentro do ambiente e, mais ainda, enriquecer reusando componentes do domínio disponíveis de outras soluções já integradas ao repositório REDIRIS.

Os mesmos componentes de domínio podem ser reusados para montar novas soluções, uma funcionalidade a que o componente Construtor deverá alcançar. Ainda, o conhecimento da aprendizagem adquirido durante este processo é retroalimentado ao metamodelo do ambiente. O Construtor também interage com o componente Otimizador para ajudar usuários na seleção do melhor conjunto de configuração para um determinado modelo multidimensional.

Outra parte significativa desse ambiente é o Avaliador, um componente responsável pela coleta de *feedback* de usuários a respeito de soluções REDIRIS geradas. O *feedback* adquirido subsidia a execução de ações que visam a melhoria do processo de construção e avaliação pós-implantação, registrando sempre todo o conhecimento adquirido no repositório da solução. Finalmente, todos os componentes REDIRIS acima mencionados são controlados por uma interface interativa.

2.6.1.3 – O Repositório do Ambiente

Como foi apresentado anteriormente, o objetivo do ambiente REDIRIS é auxiliar na construção de Sistemas de Apoio a Decisão, mais especificamente soluções de *Data warehouse* com ênfase no reuso dessas soluções. Isto requer uma infra-estrutura de metadados genérica que dê suporte ao gerenciamento de *Data warehouse* e ao reuso de esquemas multidimensionais.

O repositório é composto basicamente por dados e metadados. Ele é responsável por armazenar toda a informação necessária para a funcionalidade e gerenciamento do ambiente.

Isto inclui as próprias soluções de SAD, independente de se resultaram ou não em sucesso. Algumas características requeridas para o repositório são:

- Extensibilidade - Ele deve ser extensível para permitir que outros módulos do ambiente construam seus modelos de metadados e armazenem no repositório;
- Flexibilidade - Ele deve ser flexível para armazenar dados, metadados e meta-metadados;
- Interoperabilidade – Ele deve possuir interfaces padrões de comunicação com outros módulos do ambiente REDIRIS e com outras ferramentas externas. Por exemplo, ele deve permitir a exportação das soluções de SAD, ou seja, os esquemas multidimensionais, para que estes sejam implementados em uma ferramenta de *Data warehouse*.

2.7 Considerações Finais

Este capítulo discutiu o tema metadados voltado para o gerenciamento de *Data warehouse*.

Foi mostrado o que é uma solução de metadados e as suas arquiteturas. Em seguida, foi apresentado o ambiente REDIRIS que é o principal ambiente cliente da solução de metadados proposta por esta dissertação. O próximo capítulo aborda os padrões para a representação de metadados do W3C e OMG que foram utilizados na solução de metadados proposta por esta dissertação.

3 Alguns Padrões do W3C e OMG para representação de metadados

A proposta deste capítulo é realizar uma breve revisão dos principais padrões desenvolvidos pelo W3C (*World Wide Web Consortium*) e OMG (*Object Management Group*) que são importantes para a área de metadados. Algumas especificações não foram finalizadas, portanto não são recomendações[1] e podem sofrer alterações. As correntes versões e futuras versões dessas especificações podem ser acessadas pelos endereços www.w3c.org/TR/ e www.omg.org.

Estes padrões são abertos, independentes de plataforma, suportados por uma grande quantidade de ferramentas, são flexíveis para suportar os metadados mais estruturados e até os sem estrutura [ABS2000]. As próximas seções apresentam estes padrões.

3.1 Os Padrões do W3C

O W3C desenvolve padrões e tecnologias para interoperabilidade entre sistemas no contexto WEB e tem como o principal padrão a XML [W3C2002]. Estes padrões serão mostrados nas seções seguintes.

3.1.1 XML – Extensible Markup Language

XML é uma metalinguagem – linguagem para definir outras linguagens, de marcação extensível, derivada de SGML (*Standard Generalized Markup Language*) com recursos

[1] Uma recomendação é o nível onde o trabalho ou especificação representa um consenso dentro do W3C, ou seja, já foi aprovado.

voltados para a Web [ABK2000]. Trata-se de uma metalinguagem extensível e independente de plataforma que veio para facilitar a troca de informação na Web. XML tornou-se um padrão muito difundido e utilizado nas mais diversas áreas, suportada por uma quantidade variada de aplicações, ferramentas, *parsers*[2], *browsers,* SGBD, entre outras. A Figura 3.1 apresenta um exemplo de um documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<Table>
  <Metadata>
    <TableName>Produto</TableName>
    <Fields>
      <Field>
          <name>Codigo</name>
        <type>Integer</type>
      </Field>
     <Field>
          <name>Descricao</name>
        <type>Varchar(60)</type>
    </Field>
   </Fields>
 </Metadata>
</Table>
```

Figura 3.1 - Exemplo de um documento XML

Em um documento XML, o usuário define o seu próprio formato, que dependerá de cada aplicação. No exemplo da Figura 3.1 foi apresentado um formato para armazenar metadados sobre as estruturas das tabelas relacionais. Este formato ou esquema pode ser publicado para que outras aplicações o conheçam e possam trocar dados XML entre si. Para descrição deste formato, existem dois padrões, são eles DTD e XML Schema. Esses padrões serão abordados nas próximas seções.

3.1.2 DTD – *Document Type Definition*

O processo de verificação da estrutura de um documento XML que obedece a um certo conjunto de regras é chamado de validação. As DTD [ABK2000] são instrumentos de descrição da estrutura e validação de documentos XML herdados da linguagem SGML. O papel das DTD é fornecer meios de validar os dados XML e funcionar como uma gramática ou esquema para os mesmos. Ao construir um documento DTD, o usuário está construindo

[2] Parsers são ferramentas que lêem os documentos XSLT, XML e disponibilizam os mesmos para as aplicações

um conjunto de regras (gramática) que define a estrutura dos documentos XML validados por tal DTD.

As informações que um documento DTD pode conter são: a seqüência e aninhamento de marcadores permitidos; valores *default* e tipos de atributos; nomes de arquivos externos que podem ser referenciados; formato de algum dado externo que não seja XML e que deverá ser incluído no documento XML; entre outros. Uma DTD é especialmente necessária se os documentos XML serão processados por software que precisam receber esses documentos em uma estrutura determinada. A Figura 3.2 apresenta uma DTD para o documento da Figura 3.1.

O exemplo da Figura 3.2 diz que um elemento chamado *Table* possui dois subelementos, na seguinte ordem: *Metadata* e *Data*, sendo este último opcional (por ter o símbolo "?"). O elemento *Fields* possui como subelementos vários elementos *Field* (o símbolo "+" diz que o subelemento pode repetir *n* vezes com $n > 0$) . O elemento *TableName* não possui subelementos, ele possui apenas texto (#PCDATA) e o elemento *Data* pode possuir qualquer conteúdo, desde que este esteja declarado também na DTD.

```
<!ELEMENT Table (Metadata, Data?)>
<!ELEMENT Metadata (TableName, Fields)>
<!ELEMENT Fields (Field+)>
<!ELEMENT Field (name, type)>
<!ELEMENT TableName (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT Data ANY>
```

Figura 3.2 - Exemplo de um documento DTD

Um dos aspectos positivos do padrão DTD é que ele é simples e fácil de aprender [ABK2000], mas por outro lado possui alguns aspectos negativos citados a seguir:

- Apresenta aspectos de tipificação bastante pobre, pois apenas operam com tipos de dados String (#PCDATA e #CDATA)
- As declarações de elementos implicam na ordem dos subelementos dentro do elemento. Por exemplo, a declaração <!ELEMENT *Metadata (TableName, Fields)*> obriga os subelementos de *Metadata* a estarem dispostos dentro do documento nessa ordem definida na DTD - *TableName* e depois *Fields*. Caso seja necessário inserir um subelemento *PrimaryKey* antes de *Fields*, o documento não atende mais aos requisitos de validação dessa DTD;
- Não possui mecanismo de extensibilidade e reuso, como herança;

- Não possui a sintaxe XML.

3.1.3 XML SCHEMA

XML Schema é um outro padrão de validação assim como DTD. É uma proposta do W3C para compensar algumas deficiências das DTD. A Figura 3.3 apresenta um documento XML Schema equivalente ao documento DTD da Figura 3.2 e que também valida o documento XML da Figura 3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xs:complexType name="FieldType">
        <xs:sequence>
          <xs:element name="name" type="xs:string"/>
          <xs:element name="type" type="xs:string"/>
        </xs:sequence>
     </xs:complexType>
     <xs:complexType name="MetadataType">
        <xs:sequence>
          <xs:element name="TableName" type="xs:string"/>
             <xs:element name="Fields">
             <xs:complexType name="FieldsType">
                <xs:sequence>
                    <xs:element name="Field" type="FieldType" maxOccurs="unbounded"/>
                </xs:sequence>
             </xs:complexType>
          </xs:element>
       </xs:sequence>
      </xs:complexType>
      <xs:element name="Table">
         <xs:complexType>
             <xs:sequence>
                <xs:element name="Metadata" type="MetadataType"/>
                                    <xs:element ref="Data" minOccurs="0"/>
             </xs:sequence>
          </xs:complexType>
      </xs:element>
      <xs:element name="Data">
         <xs:complexType mixed="true"/>
      </xs:element>
</xs:schema>
```

Figura 3.3 - Exemplo de um documento XML Schema

As principais características de XML Schema são [FALL2002]:

- É escrito com a mesma sintaxe XML. Um documento XML Schema também é um documento XML;
- Possui recursos poderosos de tipificação de dados: são 44 tipos de dados fixos;
- Suporta *namespace* [BTH1999] – é o padrão proposto pelo W3C que permite a um

documento XML utilizar vários vocabulários diferentes para fazer a validação de seu conteúdo. Entende-se por vocabulário, o conjunto de termos definidos em um documento XML Schema;

- Além de permitir a declaração de elementos e atributos, também permite a definição de novos tipos, baseados ou não nos tipos pré-definidos.
- Suporte a herança entre tipos. O usuário pode construir seus próprios tipos herdando características de outros tipos previamente definidos;
- Permite uma maior precisão na cardinalidade das relações entre os elementos. No XML Schema o usuário define tanto a cardinalidade máxima quanto a mínima, ambas através de números inteiros. Ou seja, em XML Schema o usuário pode dizer que um elemento <ANO> possui um subelemento <MES>, e este tem cardinalidade mínima 1 e máxima 12.

Um documento XML Schema é um documento XML, mas que possui um conjunto de *tags* já predefinidas pelo XML Schema como, por exemplo, *complextype* que define um tipo complexo, *element* que define um elemento, entre outros [ABK2000].

3.1.4 XSLT - Extensible Stylesheet Language Transformation

XSLT é uma linguagem para extração e transformação de conteúdo XML. As transformações expressas em XSLT definem regras para converter um documento XML fonte em um outro documento resultante. XSLT é importante para extrair o conteúdo e efetuar transformações em documentos XML. A atual versão recomendada pelo W3C é a 1.0 [CLA1999], mas a versão 2.0 [KAY2002] já está em processo de revisão.

Para a transformação de um documento XML em um outro documento, seja este XML, HTML ou outro tipo, o projetista deve construir um documento contendo as regras XSLT para a transformação. Este documento é composto de unidades chamadas *templates*. Cada *template* possui duas partes: a) um "*pattern*" ou "*matching part*" que identifica o *node* XML no documento para que a ação "*action*" seja realizada; b) um "*action*" que contém o formato ou estilo a ser aplicado. A Figura 3.4 apresenta um exemplo de documento com regras XSLT.

O exemplo da Figura 3.4 apresenta um *template* cujo "*matching part*" é *"/Table/Metadata/TableName"* e a parte "*action*" é o conteúdo que está dentro do quadro mais interior. Este exemplo mostra regras para transformação de um documento XML em

instruções SQL para a criação de tabelas em um banco de dados. Os documentos XML contêm os dados referentes ao esquema e foi apresentado na Figura 3.1.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                    xmlns:fo="http://www.w3.org/1999/XSL/Format">
          <xsl:output  method="text" encoding="ISO-8859-1"/>
          <xsl:template match="/Table/Metadata/TableName">

                    <xsl:text>create table </xsl:text>
                    <xsl:value-of select="text()"/>
                    <xsl:text> (</xsl:text>
        <xsl:for-each select="/Table/Metadata/Fields/Field">
          <xsl:if test="position() != 1">
             <xsl:text>, </xsl:text>
          </xsl:if>
           <xsl:value-of select="name/text()"/>
           <xsl:text>  </xsl:text>
           <xsl:value-of select="type/text()"/>
              </xsl:for-each>
              <xsl:text>);</xsl:text>

          </xsl:template>
</xsl:stylesheet>
```

Figura 3.4 - Exemplo de um documento XSLT

Cada *template* de um documento XSLT é executado quando o *parser* encontra o elemento XML associado a ele. O corpo de um *template* contém comandos de transformação para XML, HTML, PDF, TXT, entre outros formatos. No exemplo da Figura 3.4, o *template* é executado quando o *parser* acha o elemento que possui o seguinte caminho */Table/Metadata/TableName*.

Ao submeter o documento XML da Figura 3.1, o resultado da transformação será o apresentado na Figura 3.5.

```
create table Produto (Código Integer, Descricao Varchar(60));
```

Figura 3.5 - Resultado da Transformação XSLT

3.1.4.1 Estrutura de um documento XSLT

Um documento XSLT sempre começa com uma das tags *xsl:stylesheet* ou *xsl:transform*. Um elemento *xsl:stylesheet* contém um conjunto de *templates*, representado pelo elemento *xsl:template* e por um conjunto de outros elementos que contém informações sobre a transformação. Por exemplo, o elemento *xsl:output* diz qual o tipo de formato de saída

produzida pela transformação XSLT. Todos os elementos que são filhos diretos de *xsl:stylesheet* são chamados de elementos *top level*. Dentro dos elementos *xsl:template* pode existir elementos que pertencem ao padrão XSLT ou não. Os subelementos de *xsl:template* compõem a parte *action* do *template*.

3.1.5 XQuery

Uma das grandes vantagens de XML é a flexibilidade para representar muitos tipos diferentes de informação a partir de diversas fontes. Para explorar esta flexibilidade, XML deve possuir uma linguagem de consulta que seja capaz de recuperar e interpretar informações de diversas fontes diferentes. XQuery [BCF2002] é a linguagem proposta pelo W3C para consultas a documentos XML. Ela possui alguns conceitos comuns a XSLT como, por exemplo, as expressões de caminhos. XQuery e XSLT utilizam expressões de caminhos definidos pela linguagem XPath [ABK2000]. A expressão */Table/Metadata/TableName* é uma expressão de caminho definido pelo padrão XPath e retorna o conteúdo do elemento *TableName,* que possui como pai um elemento chamado *Metadata,* cujo pai é o *node* raiz chamado *Table.*

A Figura 3.6 apresenta um exemplo de documento *XQuery* que consulta o documento XML da Figura 3.1 e gera instruções SQL para criar as tabelas relacionais.

```
FOR $doc IN document("metadata.xml") /Table/Metadata
WHERE $doc/TableName = "Produto"
RETURN
  <sql>
create table $doc/TableName (
      FOR $c IN $doc/Fields/Field
       RETURN
          $c/name $c/type ,
 </sql>
```

Figura 3.6 - Exemplo de um documento Xquery

XQuery ainda está em desenvolvimento e a última versão da especificação pode ser encontrada em http://www.w3.org/XML/Query.

3.1.6 RDF - Resource Description Framework

RDF é um padrão do W3C para descrição, intercâmbio e reuso de metadados. É baseado em tecnologias Web e pode servir para descrever qualquer domínio de conhecimento incluindo: descrição de recursos eletrônicos, mapas de *Web sites*, classificação de conteúdo, comércio eletrônico, *groupware*, entre outros [LASW1999, HJE2001, BRAY1998].

RDF reúne dois conceitos importantes que estão intimamente ligados: Metadados e Representação do Conhecimento. Ele serve para descrever as relações entre os objetos, também chamados de recursos. Um recurso é qualquer entidade que possa ser identificada por uma URI (*Uniform Resource Identifier*) [WURI2000]. São exemplos de recursos: uma página Web, um documento XML, um elemento dentro de um documento XML, uma tabela em um banco de dados relacional, entre outros.

A descrição dos relacionamentos entre os recursos segue um modelo formal que foi criado baseado em padrões de representação do conhecimento como redes semânticas e na teoria dos grafos. As descrições e relacionamentos são realizados em forma de declarações ou sentenças e codificados em XML [HJE2001]. RDF é composto basicamente por três partes: o modelo de dados, o esquema RDF e a codificação em XML.

3.1.6.1 O Modelo de Dados RDF

É representado por um grafo, onde cada objeto é um *node*, também chamado de recurso. Os arcos que vão de um objeto (recurso) a outro são as propriedades.

O modelo de dados permite que os metadados sejam interpretados por máquina. Ele é a base do padrão RDF. Os dados RDF são asserções (*statements*) codificados em XML. As asserções conferem ao RDF o poder de ser, ao mesmo tempo, lidos por humanos e interpretado por máquinas [HJE2001].

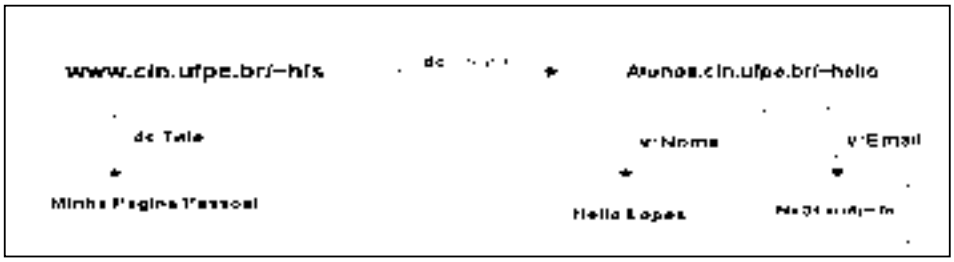

Figura 3.7 – Metadados RDF para descrição de páginas Web

A Figura 3.7 apresenta um exemplo de uma página Web que é descrita em RDF. Neste exemplo, o recurso, graficamente representado como *node*, é a página Web que está no endereço *www.cin.ufpe.br/~hls*. Os arcos *dc:creator* e *dc:title* são as propriedades, assim como *v:Nome* e *v:Email*. Essas propriedades ligam os recursos. A propriedade *dc:creator* interliga o recursos *www.cin.ufpe.br/~hls* e *Alunos.cin.ufpe.br/~helio*. Os recursos que não possuem arcos provenientes deles são chamados também de literais. Os literais normalmente representam os tipos de dados que uma propriedade pode assumir, por exemplo, inteiros,

strings, entre outros. No exemplo da Figura 3.7 existem três recursos literais, representados pelas *strings* "Minha Pagina Pessoal", "Hélio Lopes" e "hls@cin.ufpe.br". Existem ainda quatros asserções (*statments*) e são apresentados na Figura 3.8.

As asserções conferem, aos metadados, a propriedade de serem lidos por pessoas e processados por máquinas ao mesmo tempo, pois uma pessoa pode entender que a página no endereço *www.cin.ufpe.br/~hls* tem como criador *"Hélio Lopes"* e possui o título *"Minha Página Pessoal"*, e uma máquina também pode entender o modelo formal apresentado na Figura 3.8.

1 - (dc:Creator,[www.cin.ufpe.br/~hls],[Alunos.cin.ufpe.br/~helio])
2 - (dc:Title,[www.cin.ufpe.br/~hls],["Minha Pagina Pessoal"])
3 - (v:Nome,[Alunos.cin.ufpe.br/~helio],["Helio Lopes"])
4 - (v:Email,[Alunos.cin.ufpe.br/~helio],["hls@cin.ufpe.br"])

Figura 3.8 - Asserções referentes ao modelo apresentado na Figura 3.7

3.1.6.2 Esquema RDF – RDF Schema

RDF provê um modelo geral para descrever os recursos que podem ser quaisquer objetos identificados por uma URI. Estes recursos possuem várias propriedades que são outros recursos. Um conjunto de propriedades e tipos de recursos é chamado de vocabulário ou ontologia [BRGU2000, HJE2001]. RDF usa estes vocabulários, mas sozinho não pode construí-los ou defini-los. Para isto, o W3C criou o RDF Schema. Ele veio complementar o modelo RDF básico no sentido de permitir aos usuários construir seus próprios vocabulários. RDF Schema possui um conjunto de tipos básicos que torna possível a construção de outros tipos. Ele utiliza o modelo de dados RDF, visto anteriormente, para montar os relacionamentos entre os conceitos. Com RDF Schema, RDF torna-se extensível, muitos vocabulários de domínios específicos podem ser construídos e compartilhados por diversas aplicações. RDF Schema possui um vocabulário primitivo e, através dele, outros vocabulários são construídos. Os termos deste vocabulário estão agrupados em classes, propriedades e restrições. A Figura 3.9 apresenta o modelo RDF Schema [BRGU2000].

As classes têm por objetivo permitir a extensibilidade e reuso através do mecanismo de herança múltipla. RDF Schema traz um conjunto de classes primitivas, dentre as quais se destacam:

- *rdfs:Resource* – é a classe mais genérica de modelo RDF Schema. Qualquer objeto

descrito em RDF é um recurso.

- *rdfs:Class* – É subclasse de *rdfs:Resource* e representa os objetos (recursos) que possuem o mesmo tipo ou categoria. Similar ao conceito de classe em orientação a objetos.
- *rdfs:Property* – É subclasse de *rdfs:Resource* e representa um aspecto específico de um objeto (recurso). Similar ao conceito de atributo em orientação a objetos.

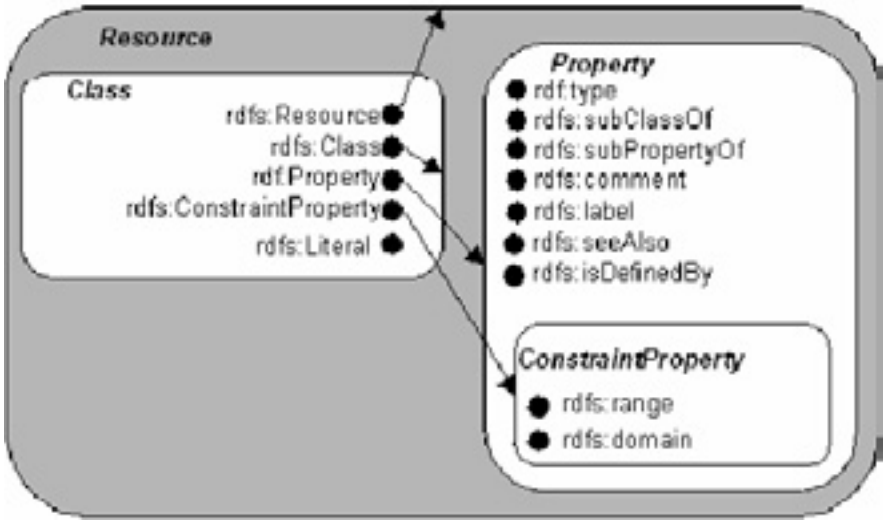

Figura 3.9 - Hierarquia de classes do modelo RDF Schema [BRGU2000]

As propriedades em RDF Schema têm como objetivo expressar relacionamentos entre os recursos. Este recurso pode ser uma classe ou uma propriedade. RDF Schema permite relacionamentos entre propriedades. As propriedades primitivas que mais se destacam são:

- *rdf:type* – interliga um recurso a uma classe. Quando isto acontece, o recurso passa a ser instância da classe, e isto o permite ter todas as suas características. Um recurso pode ser instância de mais de uma classe.
- *rdfs:subClassOf* - interliga duas classes, fazendo com que a primeira seja subclasse da segunda, herdando todas as suas características.
- *rdfs:subPropertyOf* –interliga duas propriedades, definindo uma hierarquia entre elas, fazendo com a primeira seja subpropriedade da segunda, herdando todas as suas características.

As restrições estão associadas às propriedades de um recurso e são duas:

- *rdfs:domain* – é instância da classe *rdfs:ConstraintProperty* e restringe à qual classe

uma propriedade se aplica. Este conceito mostra uma diferença entre o modelo orientado a objeto e RDF Schema. Em orientação a objeto, um atributo (propriedade) está sempre associado a uma classe de forma implícita, enquanto que em RDF Schema esta associação entre classe e propriedade é realizada de forma explícita, através da restrição *rdfs:domain*.

- *rdfs:range* – é instância da classe *rdfs:ConstraintProperty* e restringe os valores de uma propriedade.

3.1.6.3 RDF Codificado em XML

Tanto o modelo básico quanto o RDF Schema são codificados (serializados) em XML. Codificar dados RDF em XML traz muitas vantagens:

- É um padrão aberto e suporta qualquer sistema de representação de caracteres, o que permite representar os metadados em qualquer língua;
- É largamente utilizada e uma grande quantidade de aplicações, *parsers*, *browsers* suportam XML;
- Os vocabulários construídos em RDF Schema são publicados através do mecanismo de *namespace* de XML [BTH1999].

A Figura 3.10 apresenta a descrição em XML do modelo apresentado na Figura 3.7.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <rdf:RDF xmlns:rdf="http://www.w3.org/TR/WD-rdf-syntax#"
    xmlns:dc="http://purl.org/metadata/dublin_core#"
    xmlns:v="http://www.cin.ufpe.br/vocabularioPessoa#">
  - <rdf:Description about="www.cin.ufpe.br/~hls">
      <dc:Creator rdf:resource="Alunos.cin.ufpe.br/~helio" />
      <dc:Title>Minha Pagina Pessoal</dc:Title>
    </rdf:Description>
  - <rdf:Description about="Alunos.cin.ufpe.br/~helio">
      <v:Nome>Helio Lopes</v:Nome>
      <v:Email>hls@cin.ufpe.br</v:Email>
    </rdf:Description>
</rdf:RDF>
```

Figura 3.10 – Metadados RDF da Figura 3.7 codificado em XML

3.1.7 Outros padrões

Além dos padrões apresentados anteriormente que possuem uma representação própria dos dados ou metadados, existem outros padrões que são auxiliares, cujo objetivo é oferecer melhores recursos aos padrões apresentados anteriormente.

3.1.7.1 Namespace

Namespace [BTH1999] é o padrão criado pelo W3C para permitir que uma coleção de termos, também chamado de vocabulário, seja identificado por uma URI [WURI2000]. Este recurso permite utilizar um qualificador que está associado à URI onde o termo foi definido. *Namespace* facilita a reuso de esquemas XML.

3.1.7.2 XLink - XML Linking Language

Este padrão permite a criação de ligações entre os dados XML, também chamados de recursos. Ele utiliza a sintaxe XML para criar estruturas que descrevem ligações entre dados XML que podem estar no mesmo documento ou não [XLP2000].

3.1.7.3 XPointer - XML Pointer Language

O padrão *XPointer* define um mecanismo para a identificação de fragmentos (recursos) dentro de um documento XML. A união de *XLink* com *XPointer* permite a ligação (*link*) entre fragmentos específicos de documentos XML diferentes [XLP2000].

3.1.7.4 XPath - XML Path Language

Este padrão é uma linguagem para definição de expressões de caminhos. As expressões de caminhos endereçam partes de um documento XML. *XPath* é utilizado pelos padrões XSLT e XQuery [XPA1999].

3.2 Os Padrões OMG – Object Management Group

A OMG, assim como o W3C, é uma organização responsável por fornecer padrões e metodologias, tendo como foco principal o reuso, portabilidade e interoperabilidade de sistemas orientados a objetos distribuídos em ambientes heterogêneos [MOF1999]. Ela foi criada em 1989 e hoje reúne mais de 800 membros, dentre eles, desenvolvedores, vendedores e usuários de software. Dentre os principais padrões de metadados desenvolvidos pela OMG está a UML (*Unified Modeling Language*), a linguagem padrão da indústria para a especificação, visualização, construção e documentação de sistemas de software, o MOF (*Meta Object Facility*), a linguagem padrão para metamodelagem, e XMI (*XML Metadata Interchange*), o padrão para intercâmbio dos metadados que são as instâncias dos metamodelos MOF.

A especificação MOF define uma linguagem abstrata e um *framework* para especificação, construção e gerenciamento de metamodelos independentes de tecnologia de

implementação. Alguns exemplos incluem o metamodelo UML, CWM e o próprio MOF. O Capítulo 4 aborda o MOF em mais detalhes.

3.2.1 CWM – Common Warehouse Metamodel

CWM é o padrão da OMG para integração de ferramentas de *data warehousing*. Essa integração se dá pelo compartilhamento de metadados. CWM define um conjunto de metamodelos para as diversas subáreas de *data warehousing* [CWM2001, TOLB2000, PCTM2001]. A Figura 3.11 apresenta as diversas subáreas.

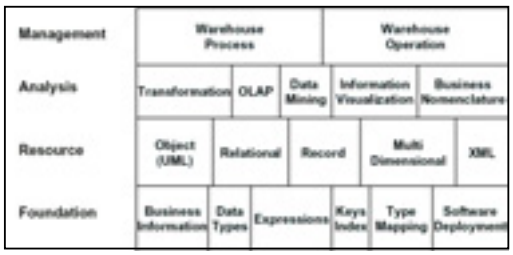

Figura 3.11 – Os pacotes do metamodelo CWM

CWM utiliza UML (*Unified Modeling Language*) para definir os diversos metamodelos, chamados também de pacotes. Cada pacote possui um metamodelo sobre um domínio específico, mas existem dependências entre os mesmos. Esses pacotes estão divididos em quatro categorias como apresentado na Figura 3.11:

- *Foundation* - Estes pacotes compreendem um conjunto de elementos de metadados utilizados por outros pacotes da especificação;
- *Resource* - Estes pacotes organizam os metadados relacionados às estruturas das fontes de dados. São exemplos: relacional, registro, multidimensional e XML;
- *Analysis* - Estes pacotes modelam os metadados relacionados ao processo de análise dos dados, são os pacotes referentes ao processo transformação, OLAP, *Data Mining*, entre outros;
- *Management* – Reúne os pacotes relacionados aos metadados que dão suporte aos dados do *Data warehouse*, como os processos e operações.

O padrão CWM poderia ser utilizado para representar metadados do ambiente REDIRIS, porém este padrão não possui suporte ao reuso. Foi apresentado no Capítulo 2 que uma das características do ambiente REDIRIS é o suporte ao reuso de soluções de *Data warehouse*.

3.2.2 XMI – XML Metadata Interchange

A principal proposta de XMI é facilitar o intercâmbio de metadados entre ferramentas de modelagem UML e repositórios de metadados baseados em MOF, em ambientes heterogêneos.

XMI é uma aplicação de XML na área de metadados. Qualquer modelo ou metamodelo baseado na especificação MOF pode ser codificado em XMI. Ela define como as *tags* de XML são utilizadas para representar metamodelos MOF descritos em XML. Assim, cada metamodelo MOF é transformado numa DTD e as instâncias dos metamodelos, ou seja, os modelos, são transformados em documentos XML validados pelas DTD. A especificação XMI [XMI2000] consiste de:

- Um conjunto de regras para produzir DTD a partir de metamodelos MOF;
- Um conjunto de regras para produzir documentos XML a partir das instâncias dos metamodelos;
- Princípios para o projeto de documentos XMI;
- Definição das DTD para os metamodelos MOF e UML;

O padrão XMI facilita a interoperabilidade entre as ferramentas por prover um formato de intercâmbio flexível e de fácil processamento, qualquer *parser* que processe documentos XML pode processar documentos XMI.

Figura 3.12 - Cabeçalho de um documento XMI

Um documento XMI é composto por duas partes: 1) um cabeçalho, representado pelo elemento *XMI.Header;* 2) um conteúdo, representado pelo elemento *XMI.Content*. O cabeçalho armazena informações sobre o conteúdo do documento, como por exemplo, o nome da ferramenta que gerou tal documento e a versão da mesma. O elemento *XMI.Header* possui ainda qual modelo, metamodelo ou metametamodelo que pertence os metadados

do documento. A Figura 3.12 apresenta um exemplo de cabeçalho XMI. O elemento *XMI.Content* contém os metadados do documento.

A Figura 3.13 apresenta um exemplo de modelo MOF e a sua representação em XMI. Neste exemplo, o conteúdo do documento é o metamodelo MOF que é formado por um pacote *Relacional* que contém uma classe *Tabela*. A classe possui um atributo *nome* referente ao nome da tabela. Além de metamodelos, XMI também pode intercambiar os metadados que são instâncias desses metamodelos. A Figura 3.14 apresenta um documento XMI que armazena as instâncias do metamodelo apresentado na Figura 3.13.

Figura 3.13 - Conteúdo XMI : metamodelo relacional

O conteúdo do documento XMI apresentado na Figura 3.14 transporta três objetos, instâncias da classe *Tabela* do metamodelo *Relacional*.

Figura 3.14 - Conteúdo XMI : metadados que são instâncias do metamodelo relacional

XMI traz muitos benefícios como um padrão de intercâmbio aberto para transporte de metadados e metamodelos MOF. Por ser XML, ele utiliza a infra-estrutura Web, permitindo a

publicação de metadados na internet, facilitando o trabalho de desenvolvedores que trabalham em ambientes remotos.

3.3 Comparação entre os padrões

Os padrões revisados neste capítulo possuem características e aplicações diferentes, mas todos podem ser utilizados para aplicações de metadados. Nesta seção serão discutidas algumas características desses padrões, quanto às vantagens e desvantagens em relação a três aspectos ligados a metadados. São eles: modelagem, intercâmbio e reuso.

Tabela 3.1 – Comparações entre os padrões do W3C e OMG

Padrão	Modelagem	Intercâmbio	Reuso	Vantagens	Desvantagens
XML		X		Suportada por uma grande quantidade de ferramentas.	Sozinha não oferece muita facilidade de intercâmbio. Precisa estar aliada a um padrão de validação como DTD ou XML Schema.
DTD	X			Fácil de aprender;	Não possui sintaxe XML; Possui poucos tipos de dados; Não possui mecanismo para reuso dos seus elementos.
XML Schema	X		X	Possui a sintaxe XML; Os esquemas podem ser reutilizados através de namespace; Permite a criação de tipos de dados;	É pouco utilizado, por ser uma especificação recente.
RDF		X		Flexível e evolutiva; Utiliza outros padrões como namespace e XML. É uma solução genérica para codificação de metadados. Serve para modelos como para outros metadados.	Por tentar ser genérica, torna-se incompleta.
RDF Schema	X		X	Possui as mesmas vantagens de RDF.	Possui as mesmas desvantagens de RDF.
XMI		X		Utiliza XML como formato. Um amplo suporte na Indústria de Software	Poucos modelos estão prontos atualmente para que possam ser intercambiados. São eles: CWM e UML.
MOF	X		X	Orientado a objeto; Flexível e evolutivo;	Poucas ferramentas dão suporte.

O aspecto modelagem significa dar suporte à construção de novas aplicações. Os padrões DTD, XML Schema e RDF Schema do W3C dão suporte à criação de esquemas XML. Os esquemas DTD e XML Schema validam a estrutura dos documentos XML. O esquema RDF

(RDF Schema) possui regras de como as sentenças de RDF devem ser interpretadas. O padrão MOF da OMG dá suporte à construção de metamodelos. A Seção 4.5 apresenta uma comparação mais detalhada entre o MOF e RDF.

O aspecto intercâmbio significa facilitar a transferência de metadados entre diversas ferramentas. Alguns padrões que oferecem as facilidades de intercâmbio são XML e RDF do W3C e XMI da OMG. XML facilita o intercâmbio, pois os dados XML são rotulados através de *tags* que descrevem os mesmos. RDF facilita o intercâmbio por possuir um padrão de codificação do seu modelo de dados para o padrão XML. O XMI é um padrão para intercâmbio de metamodelos MOF que utiliza a sintaxe XML.

O aspecto reuso significa permitir que componentes criados para uma determinada aplicação possam ser reutilizados em outras. Os padrões do W3C XML Schema e RDF Schema facilitam o reuso. XML Schema permite que novos elementos e tipos de dados possam ser construídos a partir de outros elementos e tipos definidos por outros esquemas. Estes elementos e tipos são importados de um esquema para outro através do mecanismo de *namespace* e são reutilizados através de herança. RDF Schema também utiliza *namespace* e herança para reutilizar as suas classes, propriedades e recursos. O MOF é um padrão estritamente orientado a objetos e utiliza herança como mecanismo de reuso.

Como apresentado na Tabela 3.1, cada padrão possui certas características que são adequadas a certos tipos de aplicações. Uma solução de metadados que implementa todos estes padrões tem a vantagem de suportar uma gama maior de aplicações. Pois cada aplicação poderá utilizar o repositório para gerenciar os metadados no padrão desejado.

3.4 Considerações Finais

Este capítulo realizou uma breve descrição de alguns padrões do W3C e OMG que podem ser utilizados para a representação de metadados. O padrão XML é semi-estruturado [ABS2000], o que facilita a representação de metadados de diversos formatos, tanto os mais estruturados quanto os menos estruturados. Além do XML, o W3C possui um conjunto de outros padrões que auxiliam no processo de validação dos documentos, na transformação, consulta, relacionamentos (*links)*, entre outros. A OMG possui um outro conjunto de padrões que auxilia na modelagem, armazenamento e intercâmbio de metadados. Foi realizada apenas uma breve descrição de cada padrão, pois não era o foco principal deste trabalho. Uma descrição mais detalhada pode ser encontrada nas suas respectivas especificações.

4 MOF – Meta Object Facility

Este capítulo discute a especificação MOF. O MOF foi utilizado na modelagem e implementação dos metamodelos da solução de metadados, proposta neste trabalho. É realizada uma descrição do que é o MOF, a sua arquitetura e quais os tipos de aplicações. Em seguida, é realizada uma breve descrição do modelo MOF e dos seus construtores. É abordado também o padrão de mapeamento entre os metamodelos MOF e API (*Application Programming Interface*) e é realizada uma breve descrição sobre as interfaces reflexivas do MOF.

A especificação MOF define uma linguagem abstrata e um *framework* para especificação, construção e gerenciamento de metamodelos independentes de tecnologia de implementação. Alguns exemplos incluem o metamodelo UML, CWM, discutido no Capítulo 3, e o próprio MOF. O MOF possui ainda um conjunto de regras para implementação de repositórios, que manipulam metadados descritos pelos metamodelos. Essas regras definem um padrão de mapeamento entre os metamodelos MOF e um conjunto de API para gerenciamento de metadados, instâncias do metamodelo. Por exemplo, o mapeamento *MOF -> IDL* (*Interface Definition Language*) é aplicado aos metamodelos MOF (UML, CWM) para produzir API CORBA que gerenciem os metadados, instâncias desses metamodelos. O mapeamento *MOF -> Java*, através do padrão JMI (*Java Metadata Interface*) define as mesmas regras de mapeamento para API Java. A especificação MOF compreende o seguinte:

- Uma definição formal para o meta-metamodelo MOF, ou seja, uma linguagem abstrata para a definição dos metamodelos.
- As regras para o mapeamento dos metamodelos MOF para uma tecnologia de implementação, por exemplo, CORBA ou Java.

- O padrão XMI para intercâmbio, em XML, dos metadados e metamodelos entre as ferramentas. XMI define um conjunto de regras que mapeiam os metamodelos MOF e os metadados em documentos XML.

4.1 A Arquitetura de Metadados da OMG

MOF é um *framework* extensível, ou seja, novos padrões de metadados podem ser adicionados ao mesmo. Para isto, conta com uma arquitetura em quatro camadas, também chamada de *OMG Meta Data Architecture* [MOF1999, MDA2002, MDA2001], mostradas na Tabela 4.1:

Tabela 4.1 – A arquitetura de metadados da OMG

Nível MOF	Termos Utilizados	Exemplos
M3	Meta-Metamodelo	Modelo MOF
M2	Metamodelo, Meta-Metadados	Metamodelo UML e CWM
M1	Modelos, Metadados	Modelos UMLs – diagramas de classes, Esquemas Relacionais instâncias do metamodelo CWM da camada M2
M0	Dados, Objetos	Dados do Datawarehouse

A instância de uma camada é sempre modelada por uma instância de uma camada imediatamente superior. Desta forma, a camada M0 onde estão os dados são modelados por modelos UML, como diagramas de classes que estão na camada M1. A camada M1 por sua vez é modelada pelo metamodelo UML, camada M2, que utiliza os construtores básicos como classes, relacionamentos, entre outros. Este metamodelo é uma instância do modelo MOF, que é chamado também de meta-metamodelo. Um outro exemplo, um modelo na camada M1 que esteja no padrão CWM, é uma instância do metamodelo CWM na camada M2. A extensibilidade se dá pelo fato de, em cada camada, podermos adicionar classes que são instâncias de outra classe de uma camada imediatamente superior.

4.2 Cenários de uso

O MOF define um modelo que é implementável, provendo um conjunto de interfaces para manipular os metadados. Ele pode ser utilizado em muitas áreas e aplicações que necessitem ou utilizem metadados. A seguir são apresentadas algumas destas áreas.

4.2.1 Desenvolvimento de Software

O suporte ao desenvolvimento de aplicações orientadas a objetos e distribuídas foi uma das primeiras aplicações do MOF. Um ambiente de desenvolvimento poderia consistir de um repositório de serviços para armazenamento e gerenciamento de modelos de software independentes de plataforma, por exemplo, modelos UML. Os projetistas poderiam criar estes modelos, carregá-los para o repositório. Os programadores poderiam consultar tais modelos e transformá-los em implementações de software. Neste exemplo, o repositório poderia se comunicar com ferramentas, editores gráficos que permitiriam ao projetista criar os modelos de software e carregá-los para o repositório. O repositório poderia possuir ainda um gerador de interfaces (CORBA e Java) que faria o mapeamento entre os modelos armazenados no repositório e suas implementações, através do mapeamento MOF -> CORBA IDL, MOF -> Java, ou outro mapeamento para uma outra plataforma que vier a ser construído. O uso do repositório MOF num ambiente de desenvolvimento de software seguiria os seguintes passos:

1. Os projetistas construiriam os modelos, utilizando ferramentas que se comunicariam com o repositório utilizando uma determinada notação como XMI.
2. Quando os modelos estivessem prontos, os programadores executariam os geradores de interfaces para uma determinada plataforma, que transformariam estes modelos em interfaces para o gerenciamento dos objetos, instâncias dos tais modelos.
3. Os programadores examinariam as interfaces geradas, repetiriam os passos 1 e 2 e refinariam os modelos.
4. Após o processo de refinamento dos modelos, os programadores poderiam então implementar tais interfaces geradas, construindo o servidor de objetos e construir as aplicações clientes que utilizariam tais objetos.

O repositório MOF implementaria o *MOF Model*, apresentado nas seções seguintes. Outras ferramentas poderiam implementar as interfaces reflexivas que permitiriam realizar operações sobre os objetos do modelo, sem um prévio conhecimento do modelo, ou seja, o usuário consultaria o repositório para verificar quais modelos estariam armazenados e, a partir de tais modelos, poderiam consultar as suas instâncias, que são os objetos. Funcionalidades como persistência, controle de versão e controle de acesso deveriam ser suportadas pelo *framework* repositório ou delegadas a um SGBD.

4.2.2 Gerenciamento de Tipos

Este segundo caso refere-se ao gerenciamento dos vários "tipos de informação" providos por serviços CORBA [MOF1999]. O *CORBA Interface Repository* (IR) é um servidor central que provê serviços de persistência de objetos que representam os tipos das IDLs CORBA e disponibiliza estas informações (metadados sobre as IDLs) aos clientes CORBA em tempo de execução das operações disponíveis por tais IDLs. O IR da especificação atual possui suporte apenas à leitura ao repositório; não existe um padrão para as alterações dos tipos das interfaces e não há uma maneira de aumentar as informações sobre as definições das interfaces. Alguns dos serviços providos pelo IR CORBA são: acesso dinâmico e verificação dos tipos envolvidos nas assinaturas das interfaces; interoperabilidade entre diferentes implementações de ORBs (*Object Request Broker*) e gerenciamento da instalação e distribuição das definições das interfaces.

Através de um simples ambiente de desenvolvimento baseado em MOF, os desenvolvedores poderiam escrever os modelos de informações para CORBA IDL usando o modelo MOF. O resultado poderia então ser transformado para IDL CORBA através do mapeamento MOF -> CORBA IDL e essas interfaces poderiam ser utilizadas no lugar do IR. As vantagens de utilizar as interfaces mapeadas seriam:

- Suporte às alterações das interfaces;
- Ser extensível, no sentido de facilitar a extensão do metamodelo CORBA IDL, através dos construtores da especificação MOF;
- Tornar fácil a federação de múltiplos IRs e representar associações entre tipos das interfaces CORBA e outros tipos de informações.

4.2.3 Gerenciamento de Informação

Este cenário diz respeito ao gerenciamento da informação de domínio geral, isto é, o projeto, implementação e gerenciamento de grande quantidade de informações estruturadas. Neste cenário, o repositório MOF serviria para armazenar modelos de informações de um determinado domínio. O usuário construiria o seu modelo, similar a um esquema de banco de dados, e poderia criar e gerenciar os objetos, instância deste modelo. O MOF oferece vários benefícios para o gerenciamento de grandes quantidades de informações estruturadas como, por exemplo, acesso à meta-informação em tempo de execução, consulta aos objetos do

repositório sem um prévio conhecimento do modelo que descreve tais objetos, e um padrão único, o XMI, para o intercâmbio dos objetos.

4.2.4 Gerenciamento de Data Warehouse

Como foi visto no Capítulo 2, um ambiente de *Data warehouse* depende efetivamente de uma boa infra-estrutura de metadados. Esses metadados descrevem os aspectos relevantes dos dados e processos do *Data warehouse.*

O MOF pode ser utilizado neste cenário na modelagem e implementação dos modelos específicos para os esquemas das fontes de dados, os esquemas do próprio *Data warehouse*, os modelos de transformação, entre outros. A vantagem disto seria a integração desses modelos em tempo de execução do *Data warehouse* com o ambiente de desenvolvimento, como os modelos UML das aplicações operacionais.

Atualmente, o CWM é uma especificação que define um conjunto de metamodelos MOF para um ambiente de *data warehousing* [CWM2001].

4.3 O Modelo MOF

Como foi visto em seções anteriores, o metametamodelo MOF, também chamado de modelo MOF está situado na terceira camada (M3) da arquitetura de metadados da OMG. Esta seção descreve os seus principais aspectos.

O MOF é descrito em termos dos seus próprios conceitos, ou seja, ele é auto descritivo. Isto significa que o modelo MOF é o seu próprio metamodelo. A especificação MOF o descreve de forma narrativa e através do uso de notação UML, tabelas e expressões OCL (*Object Constraint Language*). UML é utilizada apenas por possuir uma representação gráfica dos modelos, o que facilita a leitura por parte dos leitores. Ela não define a semântica do modelo MOF, que esta completamente definida na especificação MOF e não depende da semântica de qualquer outro modelo.

A linguagem OCL, que é definida na especificação UML [UML2001], provê um conjunto de elementos para definir expressões. OCL não muda o estado dos objetos, entretanto, ela apenas expressa restrições sobre os mesmos. O MOF não especifica qual linguagem deve ser utilizada para definir as restrições dos seus metamodelos, porém ele utiliza como linguagem padrão a OCL.

4.3.1 Os Construtores MOF

O MOF é um modelo orientado a objetos e possui um conjunto de elementos de modelagem que são utilizados na construção dos metamodelos, incluindo regras para o seu uso. As subseções seguintes apresentam estes construtores.

4.3.1.1 Classes

A classe é o elemento fundamental do MOF. Uma classe pode possuir três tipos de características: atributos, referências e operações. Ela pode herdar essas características a partir de outras classes (o MOF permite herança múltipla) e pode se relacionar com outras classes através das associações. O termo classe é similar ao termo utilizado em UML. Uma classe MOF é uma especificação abstrata ou classificação dos meta objetos[3] que incluem seus estados, suas interfaces, e seu comportamento, este último de maneira informal. A especificação de uma classe é suficiente para permitir a geração de interfaces concretas, com semânticas bem definidas para o gerenciamento do estado dos meta objetos.

As classes definidas no nível M2, conforme a Tabela 4.1, possuem instâncias no nível M1. Estas instâncias possuem identidade de objeto, estado e comportamento. O estado e o comportamento das instâncias do nível M1 são definidos pelas classes do nível M2.

4.3.1.2 Associações

As associações descrevem relacionamentos entre instâncias das classes dos níveis M1 e M2. Uma associação relaciona duas classes e define um conjunto de *links.* Cada *link* possui apenas duas propriedades chamadas *associationEnd*: a especificação MOF permite apenas associações binárias. Cada *associationEnd* é uma referência ao objeto envolvido na associação. As associações MOF não podem possuir outras características como atributos e métodos, apenas os dois *associationEnds*.

4.3.1.3 Tipos de Dados – DataTypes

As definições dos metamodelos às vezes necessitam utilizar atributos e valores de parâmetros de operações que possuem tipos definidos e cujos valores não precisam ter identidade de objeto. O MOF possui o conceito de *DataType* para preencher esta necessidade. Os tipos de dados se dividem em duas categorias:

[3] Meta objetos em MOF são objetos que representam metadados

- Tipos primitivos – São os tipos básicos como *boolean*, *string* e *integer*. O MOF define seis tipos primitivos.
- Tipos não primitivos – são os definidos pelo próprio projetista do metamodelo. Eles podem ser das seguintes categorias: tipos enumerados, tipo estruturado, coleção ou *alias*.

4.3.1.4 Pacotes – Packages.

Este construtor serve para agrupar grupos de elementos dentro do metamodelo. Os pacotes servem a dois propósitos:

- No nível M2, o pacote facilita o particionamento e modularização dos metamodelos. Os pacotes podem conter a maioria dos outros elementos do MOF como, por exemplo: outros pacotes, classes, associações, tipos de dados, exceções, constantes, entre outros.
- No nível M1, o pacote age como *container* de metadados. Através do objeto pacote é possível acessar todos os outros objetos, instâncias do metamodelo.

A especificação MOF define quatro mecanismos para composição e reuso de pacotes:

- *Generalization* – Os pacotes podem herdar os elementos de um ou mais pacotes, de maneira similar às classes.
- *Nesting* – Este mecanismo permite a composição de pacotes. Um pacote pode conter outros pacotes. Os pacotes aninhados possuem algumas restrições como, por exemplo, não podem ser diretamente instanciados, eles são instanciados pelo pacote que os contém. Eles não podem ser utilizados pelos mecanismos *Generalization*, *Importing* ou *Clustering*. Este mecanismo serve apenas para modularização, não serve para reuso.
- *Importing* – Algumas vezes os mecanismos *Nesting* e *Generalization* não são as melhores opções para composição ou reuso dos pacotes. Por exemplo, quando se deseja utilizar apenas alguns elementos do pacote e não outros. O MOF provê o mecanismo de *Importing* para suportar isto. Quando um pacote A importa elementos de um outro pacote, ele pode declarar atributos, operações ou exceções usando as classes ou tipos de dados do pacote importado; as operações podem levantar exceções definidas no pacote importado; ele pode definir subclasses das classes importadas, pode definir associações cujos tipos dos *associationEnds* são classes do pacote

importado.

- *Clustering* – Este mecanismo é um tanto semelhante ao Importing, porém ele faz uma ligação entre o pacote importador e o pacote importado dentro de um pacote chamado de *cluster*. Um pacote pode agrupar um ou mais pacotes ou pode estar agrupado em um ou mais pacotes. Os pacotes que foram agrupados no *cluster* se comportam da mesma forma dos pacotes aninhados. Quando o usuário cria uma instância de um pacote *cluster*, uma instância de cada pacote agrupado é criada automaticamente. Todos os pacotes agrupados pertencem ao mesmo pacote *cluster*. Os pacotes agrupados só podem ser apagados se o pacote *cluster* for apagado. Porém, ao contrário do mecanismo *Nesting*, é possível criar uma instância independente de um pacote agrupado.

4.3.1.5 Constraints

As *constraints* MOF são utilizadas para expressar regras de consistência a outros componentes como classes e pacotes. Uma *constraint* compreende:

- Um nome;
- Uma linguagem usada para especificar a *constraint*. A especificação MOF não determina uma linguagem para a definição das *constraints*. A linguagem padrão utilizada é a OCL (*Object Constraint Language*), mas outra linguagem poderia ser utilizada.
- A própria expressão descrita na linguagem OCL ou em outra linguagem;
- O tipo de avaliação que determina quando a *constraint* deve ser verificada, que pode ser uma verificação imediata (toda vez que houver mudança nos elementos onde tal *constraint* é aplicada) ou retardada.
- Os elementos do metamodelo onde tal *constraint* é aplicada.

4.3.1.6 Outros Construtores

Constantes - Uma constante MOF é utilizada pelo projetista para associar um determinado nome a um valor constante. Esta possui conceito similar às constantes das linguagens de programação.

Exceções - Possuem conceito similar aos das linguagens de programação orientadas a objetos como Java. Uma exceção MOF permite ao projetista do metamodelo definir a assinatura de

uma exceção que pode vir a ser levantada por uma determinada operação.

Tags - Uma *Tag* é o mecanismo básico que permite um metamodelo MOF ser estendido ou modificado. Uma *Tag* consiste de:

- Um nome que é utilizado para identificar a *Tag* dentro do seu *container*;
- Um identificador que identifica o tipo da *Tag*. O MOF possui algumas categorias de identificadores;
- Uma coleção de zero ou mais valores associado com a *Tag*;
- Uma coleção de outros elementos do metamodelo que estão associados com tal *Tag*. Estes elementos podem ser classes, pacotes, associaçõe, entre outros.

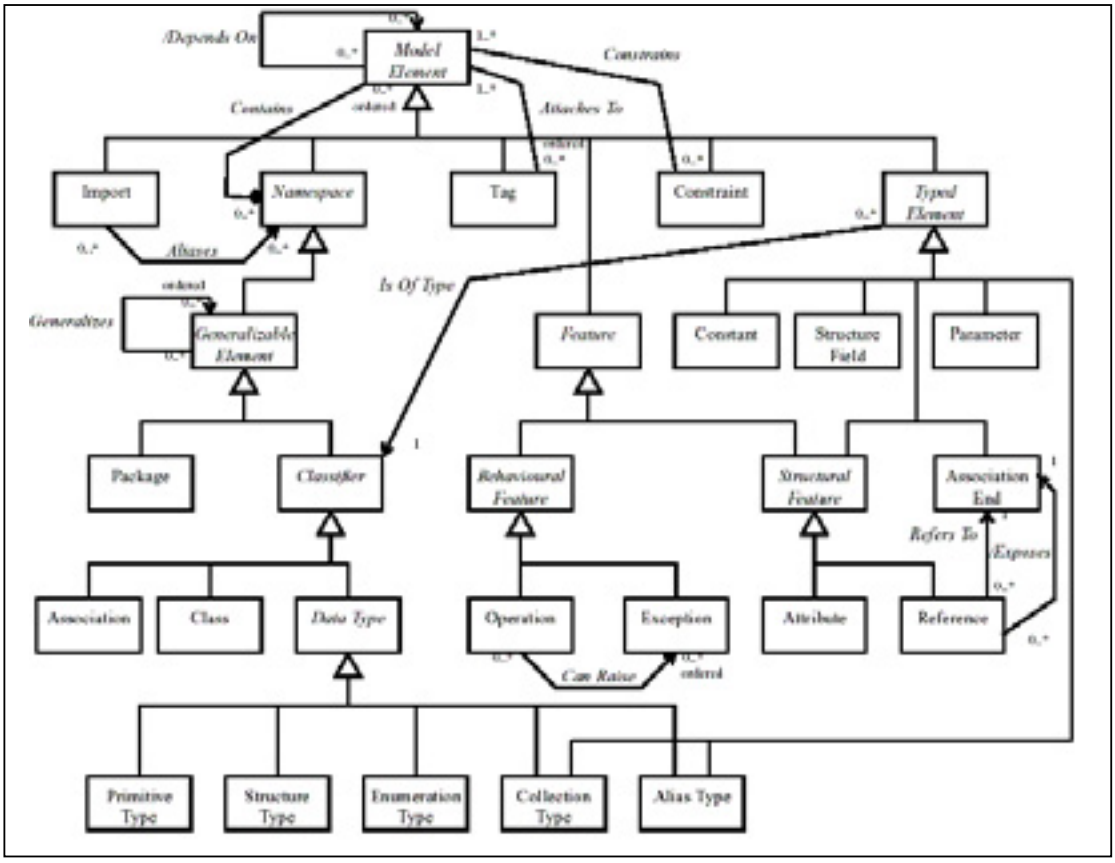

Figura 4.1 - O modelo MOF [MOF1999]

4.3.2 A Estrutura do modelo MOF

A especificação MOF utilizada neste trabalho é a versão 1.3. O modelo MOF desta versão está definido em um pacote chamado *Model*. Este pacote importa um outro pacote de tipo de

dados primitivos chamado *PrimitiveTypes* que possui instâncias dos tipos primitivos *Boolean*, *Integer*, *Long*, *Float*, *Double* e *String*. A Figura 4.1 apresenta, apenas as classes e associações do modelo MOF. Maiores detalhes podem ser encontrados na especificação do MOF 1.3 [MOF1999].

O pacote *Model* é utilizado para gerar as interfaces CORBA através do mapeamento MOF -> CORBA IDL ou as interfaces JMI através do mapeamento MOF -> Java. Através dessas interfaces o usuário poderá criar os metamodelos, instâncias do modelo MOF.

A próxima seção realiza uma breve descrição das principais classes e associações do modelo. Uma descrição mais detalhada pode ser encontrada na especificação MOF.

4.3.2.1 As principais Classes do Modelo

As principais classes do modelo MOF são:

- *ModelElement* – Esta é a classe raiz do modelo MOF. Todas as classes são subclasses de *ModelElement*. Quando mapeada para interfaces, esta classe herda da interface reflexiva *Reflective::RefObject*; porém, esta não é uma generalização dentro do modelo MOF, é apenas um artefato dos mapeamentos MOF ->CORBA IDL e MOF-> Java. Todo *ModelElement* possui um atributo *name,*que provê um nome único dentro do *namespace* que contém o elemento. Ao escolher o nome para um elemento num modelo, o projetista deve ter atenção especial aos problemas de portabilidade relacionados ao mapeamento para plataformas específicas. Por exemplo: o projetista cria uma classe cujo nome é *class.* Ao fazer o mapeamento para alguma linguagem, como Java, onde *class* é uma palavra reservada, ele terá problemas, pois o compilador Java acusará erro de uso de palavra reservada como nome de classe. O *ModelElement* possui ainda referências para o objetos *namespace* que o contém, para os outros elementos que dependem dele, e para as suas *contraints*.
- *Namespace* – Esta classe classifica e caracteriza os elementos que contém outros elementos. As subclasses desta classe oferecem duas características importantes: podem ser *containers* de outros elementos, e funcionam como um *namespace*, oferecendo um conjunto de nomes únicos para os elementos que fazem parte do *namespace*. Ela possui uma referência *contents* que identifica o conjunto de elementos que o objeto *Namespace* contém.
- *GeneralizableElement* – É a base de todas as classes do MOF que suportam herança.

Um objeto *GeneralizableElement* herda as características (atributos, referências, operações) de cada um dos seus supertipos e as características dos supertipos de cada um dos seus supertipos imediatos, e assim por diante. Ou seja, ele é transitivo. Quando um *GeneralizableElement* herda as características dos seus supertipos, os nomes dessas características passam a fazer parte do *namespace* de *GeneralizableElement.* Entretanto, um *GeneralizableElement* não pode herdar características cujos nomes venham a colidir com o das suas características. O MOF possui um conjunto de *constraint*s que prevê isto.

- *TypedElement* – É a classe base de todas as classes do nível M3 cuja definição requer a especificação de um tipo. São exemplos, atributos, parâmetros e constantes. Esta classe sozinha não define um tipo, ela está associada com a classe *Classifier,* através da associação *isOfType* apresentada na Figura 4.2.
- *Classifier* – É a classe base de todas as classes que definem tipos. Exemplos de subclasses de *Classifier* são as classes *Class* e *DataType*.

As classes *Classifier* e *TypedElement* em conjunto com as suas respectivas subclasses formam os tipos de dados utilizados pelo MOF, a Figura 4.2 apresenta o modelo.

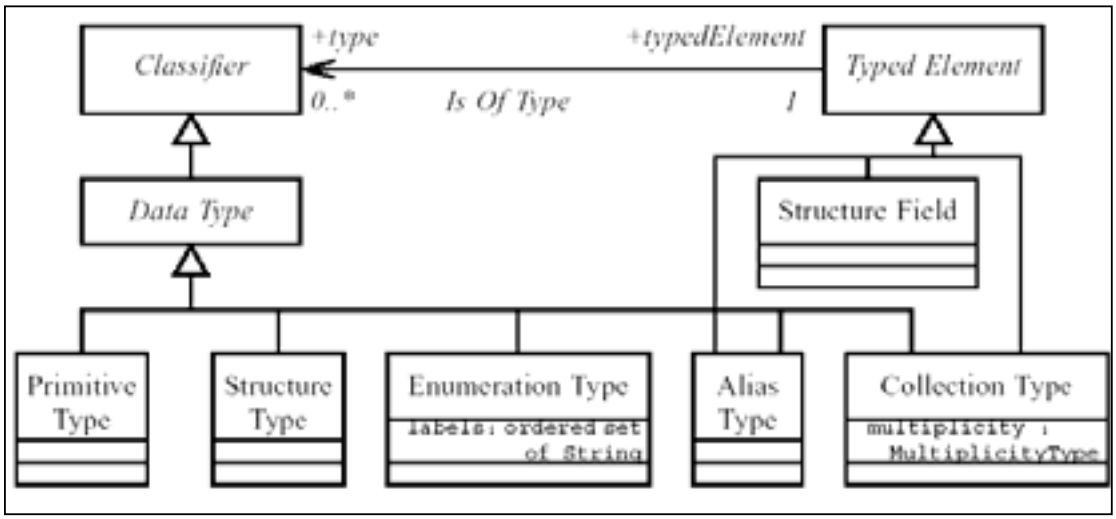

Figura 4.2 - Os tipos de dados do MOF

- *Feature* – Esta classe define as características dos objetos *ModelElement*. As características podem ser comportamentais ou estruturais. As comportamentais são as operações e exceções do modelo e as estruturais são os atributos e as referências.

- *Class* – Esta classe representa um objeto classe. Quando os metamodelos MOF são definidos e importados para um repositório MOF, as classes que pertencem a estes metamodelos são armazenadas como instâncias da classe *Class* de MOF.
- *Package* – Define as características e comportamentos dos objetos pacotes. Todos os pacotes dos metamodelos MOF são armazenados no repositório MOF como instâncias da classe *Package.*
- *Association* – Esta classe representa um objeto associação. Todas as associações dos metamodelos MOF são armazenadas no repositório MOF como instâncias da classe *Association*.
- *Attribute* – Todos os atributos das classes dos metamodelos MOF são representados por esta classe.
- *Reference* - Todos as referências que uma classe pode fazer a outra nos metamodelos MOF são representados por esta classe.

Todas essas classes possuem um conjunto de características que podem ser verificadas na especificação MOF [MOF1999].

4.3.2.2 As Principais Associações do Modelo

As principais associações do modelo MOF são:

- *Contains* – Representa um relacionamento entre as classes *Namespace* e *ModelElement*. Um metamodelo instância do MOF é definido através de uma composição de objetos do tipo *ModelElement*. Um *Namespace* define que um *ModelElement* contem outros *ModelElement*. Todas as subclasses de *Namespace* podem utilizar a associação *Contains* para compor outros elementos do metamodelo. Entretanto, existem algumas combinações que não podem acontecer, por exemplo, uma classe não pode conter um pacote. A especificação MOF trata isto na seção "*The MOF Model Containment Hierarchy*" [MOF1999].
- *Generalizes* – Esta associação é definida sobre a classe *GeneralizableElement*. Ela representa o relacionamento subclasse/superclasse entre os objetos, instâncias do modelo MOF.
- *RefersTo* – É um relacionamento entre as classes *Reference* e *AssociationEnd*. Quando um objeto do tipo *Associação* é criado, o MOF cria dois objetos do tipo

AssociationEnds, que representam as duas classes envolvidas na associação. Para cada objeto *Class* envolvido ele cria um objeto *Reference* que representa uma referência ao objeto *AssociationEnd* da *Associação*.

- *CanRaise* – Representa uma associação entre as classes *Operation* e *Exception* e diz quais exceções podem ser levantadas por uma determinada operação. Uma operação pode levantar zero ou mais exceções. Uma exceção pode ser levantada por zero ou mais operações.
- *Constrains* – É a associação entre as classes *ModelElement* e *Constraint* e diz quais *constraints* estão associadas a quais objetos *ModelElement*. Cada objeto *Constraint* pode restringir um ou mais objetos *ModelElement.*
- *DependsOn* – Esta associação é definida sobre a classe *ModelElement.* Ela permite identificar uma coleção de objetos *ModelElement* que depende estruturalmente de um outro objeto *ModelElement*.
- *AttachesTo* – É um relacionamento entre as classes *Tag* e *ModelElement* e diz quais tags estão associadas a um determinado objeto *ModelElement*.

4.3.3 Os tipos de Meta Objetos

As interfaces geradas a partir do mapeamento *MOF -> Plataforma_especifica* compartilham um conjunto comum de quatro tipos de meta objetos: *instance*, *class proxy, association* e *package*.

Package – Uma instância da classe MOF *Package*. Um meta objeto pacote representa um *container* de outros tipos de meta objetos. O pacote mais externo *(outer most package)*, representa o meta objeto raiz do modelo de meta objetos.

Class proxy – Cada classe do nível M2 possui uma classe proxy correspondente. Existe um objeto *proxy* para cada classe do nível M2. Este tipo de meta objeto serve a um número de propósitos:

- Produzir meta objetos do tipo *instance*. O meta objeto proxy é uma fábrica de meta objetos *instance*;
- Ele é um *container* para os meta objetos do tipo instance;
- Possui métodos para acessar e alterar o estado dos atributos cujo escopo é *classifier-*

scoped.

Instance – As instâncias das classes do nível M2, ou seja, dos metamodelos, são representados pelos meta objetos do tipo *instance*. Um meta objeto *instance* manipula os estados dos atributos das classes do nível M2, cujo escopo é *instance-scoped*. Podem existir muitos objetos *instance* para uma determinada classe do metamodelo. Como dito anteriormente, esses meta objetos possuem sempre um meta objeto *container* do tipo classe *proxy*. As classes *proxy* fornecem métodos para criar os meta objetos *instance*. Esses métodos recebem como parâmetros os valores dos atributos cujo escopo é *instance-scoped*. Quando um meta objeto *instance* é criado, ele é automaticamente adicionado ao seu meta objeto *proxy container*. Ele é removido do *container* quando ele é destruído. As interfaces dos meta objetos *instance* oferecem:

- Operações para acessar e alterar o estado dos atributos de escopo *instance-scoped*;
- Operações para invocar as operações de escopo *instance-scoped*;
- Operações para acessar e alterar os objetos do tipo *Association* através de referências;
- Operações que suportam identidade de meta objetos;
- Operações para deletar os meta objetos *instance.*

Association – Esses objetos manipulam as coleções de *links* correspondentes às associações do nível M2. São objetos estáticos e possuem como *containers* meta objetos do tipo *package*. As interfaces dos objetos do tipo *Association* disponibilizam:

- Operações para consultar um *link* no conjunto de *links;*
- Operações para adicionar, modificar e remover *links* a partir do conjunto;
- Operações que retornam todo o conjunto de *links*.

4.3.4 As Interfaces Reflexivas

Além do pacote *Model* que possui o modelo MOF, a especificação possui ainda o pacote *Reflective*. Este pacote possui um conjunto de interfaces que oferece um mecanismo para consultar, alterar e criar meta objetos sem um conhecimento prévio das interfaces desses meta objetos. A Figura 4.3 apresenta o diagrama que representa essas interfaces.

Todas as operações que são realizadas pelas interfaces geradas a partir do metamodelo, como

acessar atributos ou invocar operações, podem ser executadas utilizando os métodos das interfaces reflexivas. As interfaces reflexivas permitem ainda:

- Pesquisar o meta objeto de um determinado objeto;
- Pesquisar o container de um determinado meta objeto;
- Testar a identidade de um objeto;
- Deletar um objeto;

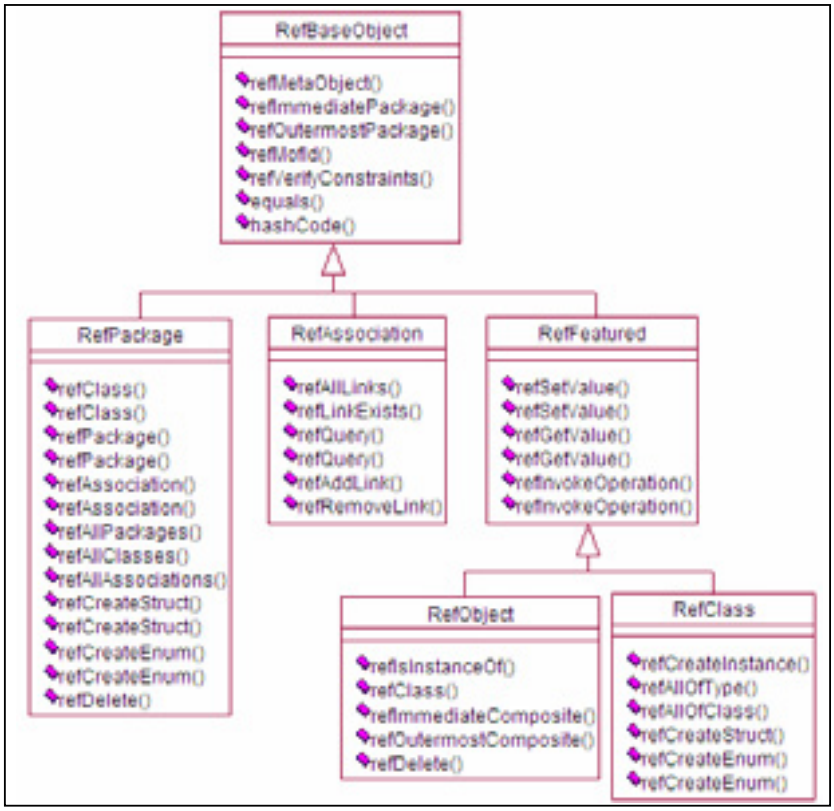

Figura 4.3 - O módulo *reflective* do MOF

A seguir é apresentada uma breve descrição das interfaces. Uma descrição mais detalhada das operações destas interfaces pode ser encontrada na especificação [MOF1999].

- *RefBaseObject* – Esta é a interface raiz do pacote, ela oferece operações comuns a todos os objetos MOF, exceto às exceções, tipos enumerados e tipos estruturados. Esta interface oferece operações para testar a identidade dos objetos, retornar um meta objeto de um objeto, entre outras.
- *RefFeatured* – Esta interface oferece métodos comuns aos meta objetos do tipo *instance* e *class proxy*. Ela oferece operações para acessar as características (atributos,

referências e métodos) dos meta objetos de maneira independente do modelo. Por exemplo, *getValue("nome_do_atributo")* desta interface retorna o valor do atributo para um determinado meta objeto que possui este atributo.

- *RefPackage* – Oferece métodos comuns aos objetos do tipo *Package,* ou seja oferece métodos para acessar os meta objetos de um determinado pacote. Por exemplo, o método *refClass("XMLDocument")* retornaria a classe *proxy* da classe *XMLDocument* do metamodelo. O *refAssociation("XMLContains")* retornaria o objeto associação *XMLContains* do metamodelo. Quando o nome da classe, associação ou pacote não existem, uma exceção é levantada.
- *RefAssociation* – Esta é a interface para manipular meta objetos do tipo *Association* do metamodelo. Ela provê operações genéricas para consultar e alterar os *links*, instâncias das associações.
- *RefObject* – Esta interface oferece métodos comuns aos meta objetos do tipo *instance*. Ela oferece métodos para acessar e alterar o estados dos atributos e invocar as operações cujos escopos forem *instance-scoped*.
- *RefClass* – Esta interface oferece métodos comuns aos meta objetos do tipo *class proxy*. Fazem parte desta interface, os métodos para acessar e alterar os estados atributos e invocar operações cujos escopos forem *classifier-scoped*, além dos métodos para criação dos meta objetos *instance*.

Todas as operações que o usuário faria com as interfaces geradas através do mapeamento MOF->*plataforma_específica*, ele pode fazer com as interfaces reflexivas. Estas interfaces são úteis para aplicações que manipulam diversos metamodelos e não têm um prévio conhecimento desses, como por exemplo, um *browser* de meta objetos.

4.3.5 O Mapeamento MOF -> CORBA IDL/Java

Esta seção descreve o padrão de mapeamento MOF para plataformas específicas. Atualmente, faz parte da própria especificação MOF o mapeamento MOF -> CORBA IDL. A comunidade Java definiu o mapeamento MOF->Java e chamou de JMI (*Java Metadata Interface*) [JMI2002]. As regras gerais de mapeamento são praticamente as mesmas para qualquer plataforma.

O resultado deste mapeamento é um conjunto de interfaces que permite ao usuário criar,

alterar e consultar metadados, instâncias dos metamodelos MOF. Por exemplo, se as interfaces forem geradas a partir do mapeamento MOF->CORBA IDL, o usuário pode utilizar clientes CORBA para acessar as interfaces. Se for JMI, o usuário poderá utilizar clientes Java.

4.3.5.1 A hierarquia das interfaces dos Meta Objetos

Esta seção descreve o padrão de herança para as interfaces mapeadas a partir dos metamodelos MOF. A Figura 4.4 apresenta um exemplo de metamodelo MOF expresso em UML que consiste de dois pacotes P1 e P2. O primeiro pacote P1 contém as classes C1 e C2, onde C2 é subclasse de C1 e uma associação A que conecta C1 e C2. O segundo pacote P2 é definido como subpacote de P1.

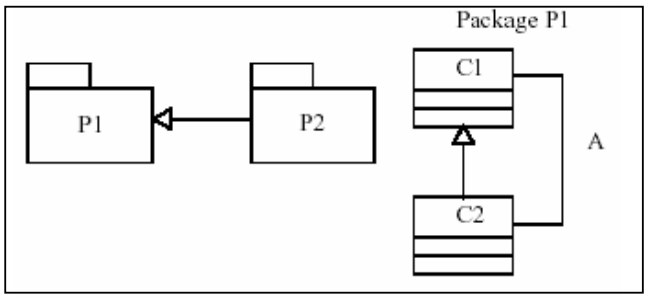

Figura 4.4 - Exemplo de um metamodelo MOF

A Figura 4.5 apresenta o diagrama UML que mostra gráfico de herança gerado a partir do mapeamento MOF para Java. A raiz do gráfico é um grupo de interfaces que fazem parte do pacote reflexivo, visto na seção anterior.

As regras de mapeamento dizem que para cada pacote e para cada associação do metamodelo são criadas uma interface *package* e uma *association,* respectivamente. Para cada classe do metamodelo são criadas uma interface *proxy* e uma interface *instance*. O padrão de herança segue as seguintes regras:

- Um meta objeto *instance,* que não possui supertipo, herda de *RefObject*; todos os outros meta objetos *instances* estendem seus supertipos.
- Um meta objeto *package,* que não possui supertipo, herda de *RefPackage*; todos os outros meta objetos *package* estendem seus supertipos.
- Todos os meta objetos *class proxy* herdam de *RefClass*.
- Todos os meta objetos *association* herdam de *RefAssociation*;

Para o exemplo da Figura 4.5, foram geradas duas interfaces *package* referentes aos pacotes P1 e P2. O pacote P1, que não possuía supertipo, herdou de *RefPackage*, enquanto P2, que possuía, herdou de P1. Foi gerada também a interface *A* para a associação entre as classes C1 e C2 do metamodelo. Para cada classe do metamodelo foram geradas duas classes: uma para os meta objetos *instances* e outra para as *class proxy*. As interfaces *C1Class* e *C2Class* representam as interfaces *proxy* geradas, respectivamente, a partir das classes C1 e C2 do metamodelo, e herdam diretamente de *RefClass*. As interfaces *C1* e *C2* representam as interfaces para os meta objetos instances. Apenas a interface *C1*, cuja classe do metamodelo não possuía supertipo, herdou de *RefObject*.

Figura 4.5 - Exemplo do metamodelo MOF mapeado para interfaces Java

4.4 JMI – *Java Metadata Interface*

A especificação JMI define o mapeamento entre metamodelos MOF e a linguagem Java. A versão 1.0 desta especificação é baseada na versão 1.3 do MOF. Usando JMI, as aplicações e ferramentas que especificam seus metamodelos MOF podem gerar as interfaces Java

referentes aos metamodelos. Estas aplicações e ferramentas podem criar, alterar, apagar e recuperar informações através dos serviços oferecidos pelas interfaces JMI. As interfaces JMI são divididas em dois pacotes: *javax.jmi.model,* que representa o modelo MOF, e javax.*jmi.reflect*, que representa as interfaces reflexivas. Este trabalho utiliza JMI como padrão para mapeamento dos metamodelos MOF.

4.5 MOF e Outros Padrões

Esta seção apresenta a relação entre o MOF e alguns padrões de modelagem e descrição de metadados.

4.5.1 MOF x UML

A UML foi adotada pela OMG como o padrão para modelagem de sistemas discretos [UML2001]. Ela provê uma linguagem de modelagem orientada a objetos independente de plataforma, com conceitos bem definidos como classes, objetos, *uses cases*, associação, generalização, e diagramas gráficos. Alguns desses conceitos são também utilizados pelo MOF, o que causa alguma confusão. Por exemplo, a classe *Class* de UML é definida como instância da classe *Class* de MOF. As diferenças chave entre o MOF e UML estão nas suas aplicações. A UML é utilizada como linguagem de modelagem de propósito geral enquanto que o MOF é utilizado para metamodelagem. A especificação define a UML como uma instância do MOF. Existe uma equivalência estrutural entre os construtores de UML e MOF, com algumas diferenças listadas nos tópicos seguintes:

- O MOF suporta apenas associações binárias enquanto que UML suporta *n-árias*. Isto foi definido assim porque dificilmente se usa, em metamodelagem, uma associação com mais de duas classes envolvidas. No entanto, as próximas especificações MOF suportarão associações *n-árias*.
- As associações MOF não contêm características como atributos e métodos enquanto que em UML isto é possível.
- O MOF suporta o conceito de referência. Se duas classes A e B possuem uma associação, então o MOF gera uma referência em A de B e em B de A, respeitando as suas multiplicidades. As referências permitem ao cliente navegar de uma classe para outra sem precisar utilizar a associação. A UML utiliza um atributo com o nome do *associationEnd*, porém a navegação é realizada pela associação. Do

ponto de vista de modelagem, tanto a referência quanto o atributo *associationEnd* são representados da mesma maneira, o que muda é a forma como está implementado.

- Alguns conceitos como generalização, dependência e refinamento são implementados como classes em UML e como associações em MOF.

4.5.2 MOF x RDF

Esta seção apresenta uma comparação entre os padrões MOF e RDF, demonstrando as suas relações. Já foi visto que tanto MOF como RDF são *frameworks* para modelagem e descrição de metadados. O MOF foi especificado pela OMG e está mais focado no contexto de sistemas de informação enquanto que o RDF foi especificado pelo W3C e está relacionado ao contexto Web. A seguir são apresentados alguns tópicos que focam as diferenças e relações entre estes dois padrões:

- *Padrão para Intercâmbio de metadados* – Os metamodelos MOF são descritos em XML através do padrão XMI. Os metadados RDF também são descritos em XML.
- *Extensibilidade e reuso* - MOF é extensível, permite que novos modelos MOF possam ser criados (CWM,UML,XML,DTD). MOF utiliza herança como mecanismo de extensibilidade e pacotes como mecanismo de reuso. O RDF também é extensível, através do RDF Schema. Ele utiliza *namespace* como mecanismo de reuso e herança como mecanismo de extensibilidade.
- *Interoperabilidade* – O MOF tem interoperabilidade em duas formas: através de interfaces padrões de software, onde outras ferramentas podem gerenciar metadados do modelo, ou através do intercâmbio dos metadados em XMI. O RDF tem interoperabilidade apenas em uma maneira: através do intercâmbio dos modelos descritos em XML.
- *Paradigma* - O MOF é estritamente orientado a objetos, possui as características básicas da orientação a objetos como herança, classes e associações. O RDF, apesar de ser orientado a objetos, possui diferença da orientação a objetos empregada nas linguagens de programação. Em RDF os objetos não são definidos em termos de propriedades e métodos, mas as propriedades são definidas em termos dos objetos. O RDF tem fortes bases em sistema de representação do conhecimento como *Frames*

[HJE2001].

- *Arquitetura* – O MOF possui quatro camadas ou níveis de abstração sugeridos pela OMG, enumeradas de zero a três. Estes quatros níveis não precisam ser fixos, e cada nível inferior é uma instância de um nível superior até chegar ao nível mais alto, o número três, que é o Meta-metamodelo MOF. O padrão RDF não define níveis, isso é feito pelo próprio usuário. RDF possui apenas o RDFS como Metamodelo, através do qual o usuário pode construir quantos modelos e em quantos níveis ele quiser.
- *Objetos e Recursos* - O MOF possui o conceito *Objeto* para representar os metadados. O RDF possui o conceito *Recurso* para representar o objeto a ser descrito. Cada objeto MOF está associado a um único identificador (ID) para todo o repositório. Cada recurso RDF possui uma URI (*Uniform Resource Identifiers*) única para toda a Web.
- *Modelo de Constraints* – O MOF e RDF possuem uma forma de especificar restrições sobre os metadados. O MOF possui a classe *Constraint*, onde o usuário pode definir uma *constraint* usando uma linguagem para definição como OCL. As *constraints* MOF podem agir sobre qualquer componente do metamodelo. RDF utiliza as propriedades de RDF Schema *range* e *domain* para especificar *constraints*.

4.6 Considerações Finais

Este capítulo descreveu a especificação MOF e as suas aplicações. Foi realizada uma descrição do modelo MOF, a sua estrutura, o padrão de mapeamento para interfaces CORBA IDL e Java e o conjunto de interfaces reflexivas. Logo após foi mostrado JMI, o padrão de mapeamento MOF utilizado nesta dissertação. E por último foi realizada uma comparação do MOF com outros padrões como UML e RDF. O próximo capítulo apresentará os metamodelos MOF construídos a partir dos padrões do W3C. Esses metamodelos fazem parte da solução de metadados que é o objetivo central deste trabalho.

5 Uma Proposta de Solução de Metadados

Este capítulo propõe uma implementação de uma solução de metadados para o ambiente REDIRIS. São apresentados uma proposta de implementação, os passos para realização da mesma e a ferramenta utilizada. Finalmente, é apresentado o metamodelo XML que faz parte da solução.

5.1 Introdução

Esta seção apresenta o problema e a abordagem utilizada na implementação da solução de metadados.

5.1.1 O problema

Como foi apresentado no Capítulo 2, o REDIRIS se propõe a auxiliar os usuários nas tarefas de análise, modelagem, construção e reuso de solução de Data warehouse. O módulo de metadados será responsável por gerenciar toda a informação necessária para a funcionalidade e gerenciamento do ambiente. Isto inclui os blocos de dados, proveniente de sistemas legados, as técnicas para pré-processamento e transformação dos dados, as regras de negócio das aplicações de SAD (Sistema de Apoio à Decisão), os modelos multidimensionais e outros metadados, primordiais para o gerenciamento do ambiente, tanto no mapeamento entre os componentes das soluções de SAD, quanto no suporte ao reuso dessas soluções.

5.1.2 A proposta

Os padrões do W3C (XML, DTD, XSLT) possuem muitas vantagens como formato de armazenamento, descrição e intercâmbio de metadados que são apresentados a seguir

[MAIN2000, AHAY2001]:

- São padrões abertos, estabelecidos por organizações internacionais. São independentes de plataforma e de vendedor;
- Suportam o padrão internacional *ISO Unicode*[4] para representação de caracteres em qualquer língua e alfabeto;
- Possuem sintaxe e estrutura textuais, o que facilita a leitura;
- Suportados por uma grande quantidade de ferramentas de modelagem, de desenvolvimento, *Browsers*, SGBD, entre outras;
- São padrões semi-estruturados, flexíveis o suficiente para armazenar metadados estruturados e não estruturados.

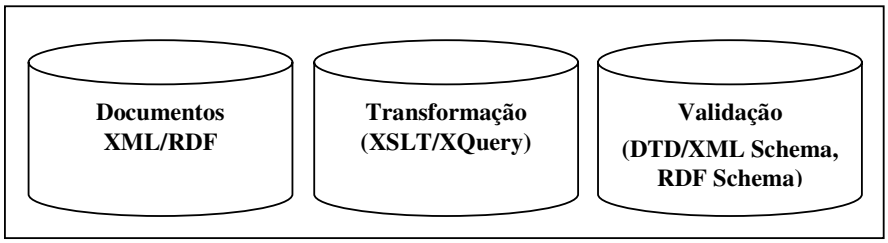

Figura 5.1 – Um típico repositório XML

Um típico repositório XML de metadados é apresentado na Figura 5.1. Ele armazena os metadados em forma de documentos XML. Esses documentos são validados pelas regras estabelecidas nos documentos do repositório de validação, que podem ser XML Schema, DTD (*Document Type Defination*) ou outro padrão de validação [ABK2000]. Os mesmos documentos XML podem ser transformados, ou integrados em outros documentos através das regras estabelecidas pelos documentos do repositório de transformação (XSLT ou XQuery) [CLA1999]. Por exemplo, um documento XML que armazena um modelo dimensional pode ser transformado e apresentado ao usuário em um *browser* através de uma interface HTML. Este mesmo documento pode ser transformado em um conjunto de scripts SQL para a criação

[4] http://www-cgrl.cs.mcgill.ca/~luc/standards.html

do banco de dados, ou ser gerado um outro documento XML no padrão XMI para exportação para outras ferramentas.

A abordagem da Figura 5.1 pode ter várias formas de implementação como [BOU2000]:

- **Banco de Dados Relacionais**: Os elementos dos documentos XML são mapeados diretamente para as colunas das tabelas relacionais. Os SGBD que implementam essa solução trazem ferramentas que auxiliam o usuário no mapeamento e recuperação dos dados XML. Um exemplo é o SQL Server 2000 [BOU2001];
- **Bancos de Dados Objeto-Relacionais com extensão XML**: Aqui existem basicamente duas opções: mapeamento direto dos elementos XML para as colunas das tabelas relacionais, ou utilizar tipos complexos para armazenamento de todo o documento XML. Esta última opção utiliza o poder de construir tipos complexos de dados dos bancos objeto-relacionais. Exemplos são as versões 8i e 9i da Oracle [HIG2001] e o DB2 6000 da IBM [DAN2002]. Esses SGBD fornecem tipos complexos para armazenamento de dados XML e operações próprias para manipulação do seu conteúdo; ou oferecem ferramentas para mapeamento direto dos elementos dos documentos XML em colunas de tabelas;
- **Bancos de Dados Orientado a Objetos**: Apesar de a tecnologia de orientação a objeto para SGBD não ter tido tanto sucesso quanto a relacional [BOU2001], algumas empresas que possuem SGBD Orientado a Objetos encontraram no padrão XML um novo foco de mercado. Nessa modalidade, os elementos dos documentos XML são transformados em objetos. Um exemplo é o *ObjectStore* da empresa ExcelonCorp [OBJ2001].
- **Bancos de Dados XML Nativo**: BDs desse tipo apresentam essa terminologia devido ao fato de apresentar documentos XML armazenados de forma intacta, ou seja, não existe nenhum tipo de mapeamento. São produtos construídos especialmente para armazenamento de dados XML. Dividem-se em duas categorias: aqueles que armazenam os dados em forma de texto e os que armazenam dados como objetos DOM. O uso de armazenamento em estruturas DOM possui a vantagem de não precisar construir tais estruturas na memória com fins de manipular dados XML. Estes são chamados de SGBD nativos baseados em objetos enquanto que os que armazenam os documentos XML em forma de texto são chamados de SGBD nativos baseados em

textos. Um exemplo de SGBD nativo é o Tamino da empresa SoftwareAG [TAM1999];

- **Arquivos Comuns**: Os dados XML são armazenados em arquivos comuns não requerendo nenhum SGBD para manipulá-los. Ao armazenar os dados em arquivos, o usuário fica sujeito apenas à segurança do sistema operacional. Um exemplo deste tipo de solução pode ser encontrada em [SBB2002].

Todas as implementações citadas anteriormente possuem vantagens e desvantagens quando se trata de armazenamento de documentos XML. Porém, elas não são focadas no gerenciamento de metadados. Elas são direcionadas ao gerenciamento de informações. Quando o foco é metadados, outras questões devem ser levantadas como, por exemplo, o uso de padrões para compartilhamento e interoperabilidade de metadados entre ferramentas [MAIN2000]. Foi visto nos Capítulos 3 e 4 que os padrões da OMG (MOF, XMI, CWM) foram construídos especificamente para dar suporte a modelagem, gerenciamento e intercâmbio de metadados. Como foi discutido no Capítulo 4, o MOF oferece um *framework* para modelagem e gerenciamento de metadados, dispondo de um conjunto de interfaces comuns para acesso aos metadados, instâncias dos diversos metamodelos da solução.

Este trabalho propõe a modelagem dos padrões XML em MOF. Estes padrões estão apresentados na Tabela 5.1.

Tabela 5.1 - Padrões suportados pelo repositório proposto

Nome do Padrão	Breve descrição
XML	Utilizado como meio de armazenamento dos metadados
DTD	Padrão utilizado na validação dos documentos XML
RDF	Padrão oficial do W3C para descrição, reuso e intercâmbio de metadados.
RDF Schema	Padrão oficial do W3C para construção de vocabulários de metadados.
XSLT	Padrão oficial de transformação de documentos.

Uma das vantagens desta abordagem é a possibilidade de compartilhamento dos metadados do ambiente entre diversas ferramentas de vendedores diferentes utilizando o padrão XMI [CHAN2000].

Uma outra vantagem é um conjunto de interfaces comuns para acessar e gerenciar os dados e metadados descritos por qualquer padrão modelado. Atualmente, para se acessar um documento XML é necessário um *parser* que implemente um dos dois tipos de interfaces para acesso a documentos XML: o DOM (*Document Object Model*) [CBNW1997] ou o SAX (*Simple API for XML*) [MEGG2001]. Para acessar um documento RDF é necessário um outro tipo de *parser* que possui um outro conjunto de interfaces, e que é definido por cada vendedor, pois não existe uma interface padrão. Para acessar um documento DTD é necessário um outro conjunto de interfaces, que também não possui um padrão definido e é fornecido por cada vendedor. E isto também é verdade para todos os outros padrões do W3C. Com a modelagem desses padrões em MOF seriam geradas interfaces de acesso aos dados e metadados descritos por esses padrões. Atualmente, o MOF define um mapeamento para IDL CORBA [MOF1999] e JMI (*Java Metadata Interface*) [JMI2002].

Cada padrão também possui uma forma diferente de representar os dados, apesar de a maioria deles possuir a sintaxe XML. Por exemplo, RDF possui uma forma de representar seus dados serializados em XML; XML Schema também possui um conjunto de elementos (*tags*) utilizado para definir um documento esquema; e DTD possui uma outra forma de representação que não é sintaxe XML. Com a modelagem desses padrões em MOF, todos os dados e metadados descritos por estes padrões poderão ser representados em apenas um padrão, o XMI (*XML Metadata Interchange*). XMI é um padrão suportado por várias ferramentas nas mais diversas áreas:

- Desenvolvimento de sistemas – Ferramenta *Netbeans*.[5]
- Especificação e modelagem de sistemas - Rational Rose[6], ARGO UML[7].
- Ferramentas de *Data warehouse* – DB2 Warehouse Manager[8], Oracle Warehouse Builder[9]
- Metadata Repository - Universal Repository (UREP)[10], MDR (*Metadata Repository*) [MDR2002], dMOF [DMOF2001]

Em resumo, esta abordagem traz as seguintes vantagens:

[5] http://www.netbeans.org/
[6] http://www.rational.com/
[7] http://argouml.tigris.org/
[8] http://www-3.ibm.com/software/data/db2/datawarehouse/
[9] http://otn.oracle.com/products/warehouse/content.html
[10] http://www.unisys.com/

- Possui a flexibilidade de representação de dados dos padrões XML;
- Suporte a um número de aplicações. Cada aplicação poderá utilizar o repositório para gerenciar os metadados no padrão desejado;
- Um conjunto de interfaces padrão para acessar qualquer padrão de metadados;
- O padrão XMI como formato único para intercâmbio de metadados dos padrões modelados.

Na seção seguinte, serão apresentados os passos propostos para a modelagem e implementação de metamodelos MOF.

5.2 Etapas para a Construção do Repositório

A modelagem e implementação dos metamodelos utilizaram um conjunto de ferramentas e uma seqüência de passos que são descritos a seguir e apresentados na Figura 5.2:

- Definição dos metamodelos em UML. Pelo fato de até a realização deste trabalho não existirem ferramentas para modelagem MOF, foi utilizado UML como padrão de representação dos metamodelos.
- Geração de documentos XMI relacionados aos metamodelos projetados anteriormente. Isto foi utilizado através de um *Plug-in*[11] da ferramenta *Rational Rose* que exporta um determinado modelo de classes para XMI MOF.
- Realização de eventuais mudanças nos documentos XMI relacionados aos metamodelos. Isto é necessário porque em UML não se consegue representar tudo que se representa em MOF. Desta forma pode-se adequar melhor o metamodelo ao que é proposto realizando-se mudanças nos próprios documentos XMI.
- Importação dos metamodelos XMI, definidos anteriormente, para o repositório MOF. Isto foi realizado utilizando a ferramenta MDR. Neste momento o metamodelo é uma instância do metamodelo MOF.
- Geração e compilação das interfaces *java* referentes ao metamodelo XMI, seguindo o padrão JMI de mapeamento de metamodelos MOF para interfaces *java*.

[11] http://www.rational.com/download/

- Instalação das interfaces geradas como um módulo da ferramenta *Netbeans*[12]. Este passo só é necessário para a ferramenta MDR, que é um módulo da ferramenta *Netbeans*, como será apresentado na Seção 5.3.
- Construção de classes que fazem a transformação dos metadados nos padrões específicos (XML, DTD, RDF, RDF Schema) para os seus respectivos metamodelos MOF.

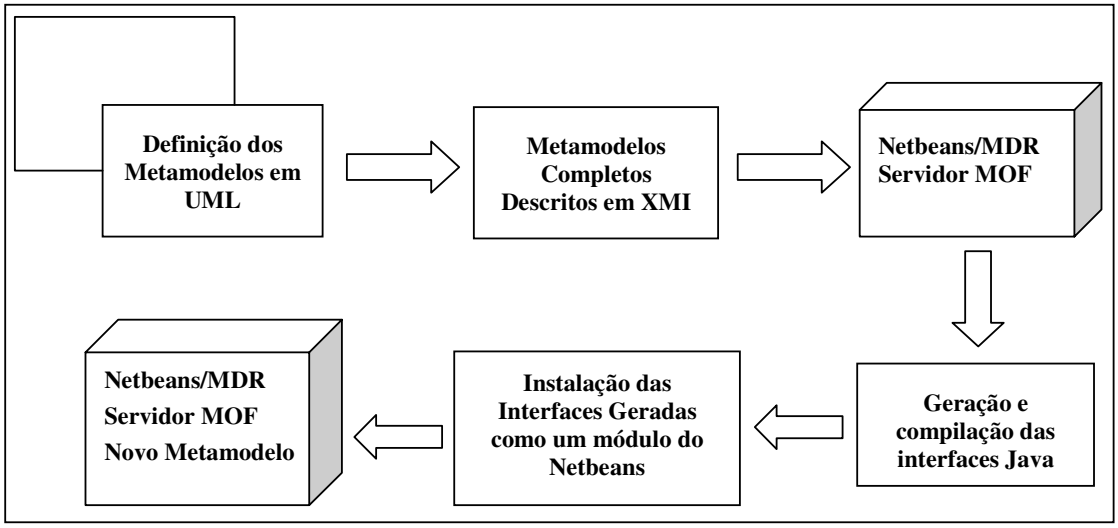

Figura 5.2 - Passos para a modelagem e implementação dos metamodelos MOF na ferramenta MDR

A próxima seção mostra a ferramenta utilizada na implementação dos metamodelos. Qualquer ferramenta que implementa a especificação MOF pode ser utilizada na implementação dos metamodelos. A ferramenta MDR foi escolhida por ser código aberto e gerar interfaces JMI, possuir um *browser* de objetos que apresenta os objetos do repositório MOF de uma forma gráfica, facilitando a compreensão. Além do fato da linguagem Java ser comum ao ambiente acadêmico.

5.3 MDR – Metadata Repository

O MDR foi a ferramenta utilizada na implementação dos metamodelos MOF. Ela é uma implementação *Open Source* da especificação MOF 1.3 e 1.4, através de JMI e está integrada à ferramenta *Netbeans* da Sun. Entre as suas características estão [MDR2002]:

[12] www.netbeans.org

- Persistência dos metamodelos e metadados. Esta persistência é feita em arquivos binários e de formato proprietário. A ferramenta não possui persistência em XML ou SQL, por exemplo. Porém, como ela é uma ferramenta de código aberto, qualquer usuário pode implementar a persistência em SQL, Banco de Dados Orientado a Objeto, XML ou qualquer outro tipo de armazenamento.
- Suporte a importação e exportação dos metamodelos para XMI, versões 1.1 e 1.2.
- Geração de interfaces *java* para qualquer metamodelo importado para o repositório.
- Instanciação do metamodelo. Este processo permite que os metadados possam ser importados para dentro do repositório como instâncias dos metamodelos específicos. Os metadados podem ser acessados através das interfaces reflexivas ou das interfaces específicas do metamodelo, geradas através do mapeamento MOF -> *Java*.
- Implementação automática das API geradas a partir dos metamodelos específicos. O usuário não precisa implementar estas API.
- Acesso via *java* ao repositório, ou seja, o usuário pode utilizar um programa escrito em *java* para realizar as diversas tarefas do repositório tais como: importar um metamodelo, gerar as suas interfaces e instanciá-lo.

Para instalar o MDR, o usuário deve primeiro instalar a ferramenta *Netbeans.* Após esta tarefa, ele deve instalar o módulo MDR selecionando o *menu tools -> update center* e escolher o módulo MDR e instalá-lo. Após a instalação, o menu *View* passará a apresentar a opção *MDR Repository*. Esta opção apresenta um *browser* contendo os metamodelos armazenados no repositório. A Figura 5.3 apresenta o *browser MDR Repository*.

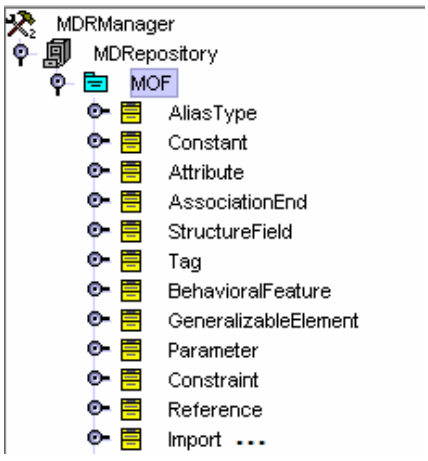

Figura 5.3 – O browser de objetos da ferramenta MDR

Inicialmente, apenas o metamodelo MOF está armazenado no repositório, ou seja, o repositório MOF armazena o seu próprio metamodelo. Para esclarecer melhor, a Figura 5.4 apresenta o metamodelo MOF e a suas instâncias.

Todas as classes do metamodelo MOF são armazenadas como instâncias da classe *Class*, assim como todos os atributos e associações são armazenados, respectivamente como instâncias das classes *Attribute* e *Association* do MOF.

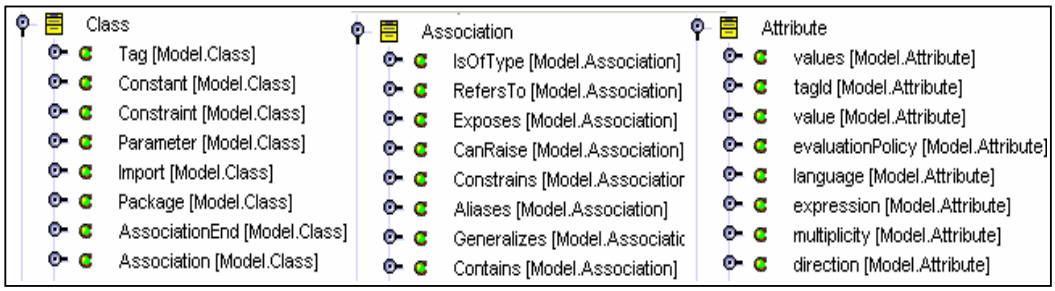

Figura 5.4 – O metamodelo MOF como instância do MOF

As próximas seções e o Capítulo 6 tratam da modelagem em MOF dos padrões suportados pelo repositório da Tabela 5.1. Os metamodelos são apresentados na notação UML. Este trabalho exigiu um estudo aprofundado das especificações dos padrões. As definições dos termos aqui modelados poderão ser encontradas nas especificações dos respectivos padrões.

5.4 O Metamodelo XML

Esta seção apresenta o metamodelo XML e a sua implementação como um repositório MOF. Ele tem como objetivo armazenar e manipular dados XML em repositórios MOF. Um documento XML é composto por *nodes*. Esses *nodes* podem ser elementos, atributos, textos, entre outros. Elementos podem conter atributos, textos ou outros elementos aninhados. Um elemento contém outros *nodes* através de uma agregação. Cada documento XML possui um *node* raiz do qual todos os outros *nodes* do documento XML são filhos.

A seguir será realizada uma descrição textual dos componentes do metamodelo.

- *XMLObject* é a classe mais genérica do metamodelo. Ela possui um atributo *name* e que todas as outras classes herdam.
- *XMLDocumentType* representa um *node* do tipo *DocType*. Um *node DocType* referencia uma DTD externa ou interna. O modelo apresentado aqui suporta apenas referências a DTD externas. Para o suporte às DTD internas teriam que ser

construídas classes para manipular DTD dentro do metamodelo XML, e essas classes já fazem parte do metamodelo DTD. Um *DocType* só pode vir no prólogo[13] do documento XML, desta forma, ele só poderá estar contido em um *node* do tipo *XMLDocument*. A seguir é apresentado um exemplo e a sua representação no metamodelo *XMLMetamodel*.

```
<!DOCTYPE DATAWAREHOUSES SYSTEM "D:\ServerREDIRIS\MDR\data\dw.dtd">
```

- *XMLEntityReference* representa uma referência a uma entidade declarada previamente no documento. A referência a uma entidade é realizada da seguinte forma: *&<nodedaentidade>;*. Um *node* do tipo *XMLEntityReference* poderá ser filho apenas de *XMLElement*, ele não poderá ser filho de *XMLDocument*, desta forma, o metamodelo não suporta referências a entidades no prólogo de documento XML.
- *XMLNodeText* representa um texto em um documento XML. Um *node* texto não poderá conter qualquer outro *node*.
- *XMLElement* representa um elemento em um documento XML. Os elementos podem conter outros elementos, atributos, texto, entre outros.
- *XMLNodeValue* é uma classe comum a alguns *nodes* que necessitam de um atributo chamado *value*. Esta classe é abstrata, ou seja, ela não possui instâncias diretas. Apenas as suas subclasses podem ser instanciadas.
- *XMLCdataSection* representa uma seção *Cdata* de um documento XML.
- *XMLComment* representa os comentários em um documento XML. Os comentários podem ser utilizados tanto dentro do documento, quanto no prólogo.
- *XMLAttribute* representa um atributo em um documento XML. A classe *XMLAttribute* possui dois atributos herdados: *name*, que herdou a partir de *XMLObject,* e *value,* a partir de *XMLNodeValue*.
- *XMLEntity* representa a declaração de uma determinada entidade no documento XML. Esta classe também possui dois atributos herdados chamados *name* e *value*.

[13] Parte incial de um documento XML, antes do elemento raiz.

- *XMLProcessingInstruction* representa uma instrução de processamento de um documento XML, e pode aparecer tanto no prólogo quanto dentro do documento.
- *XMLDocument* é a classe que representa o *node* raiz de um documento XML. Ela armazena dados como versão, data de criação, alteração e uma breve descrição do documento.
- Somente *XMLElement* e *XMLDocument* podem conter outros elementos, mas *XMLDocument* não deve estar contido em nenhum *node*.

A Figura 5.5 apresenta o metamodelo XML em notação UML. Todos os componentes de um metamodelo MOF (classes, atributos, associações) devem estar dentro de um ou mais pacotes. Para o metamodelo XML foi criado o pacote *XMLMetamodel*. As classes que estão em cinza são abstratas, ou seja, elas não podem instanciar objetos diretamente, apenas as suas subclasses.

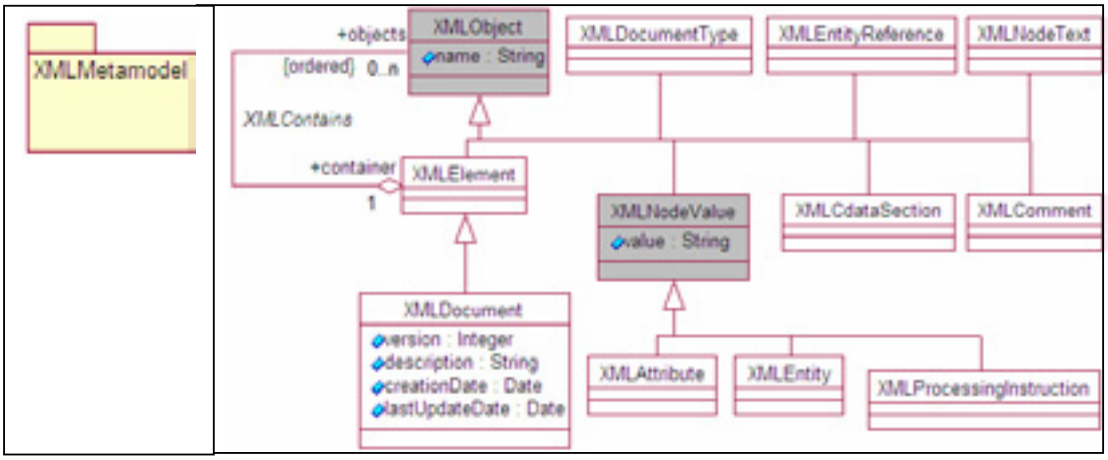

Figura 5.5 – O metamodelo XML proposto – XMLMetamodel

5.4.1 Transformação do Metamodelo XML para XMI

Após a construção do metamodelo XML, o próximo passo é gerar um documento XMI referente ao mesmo. Este processo foi realizado através de um *Plug In* da empresa Unisys para a ferramenta *Rational Rose* que foi visto na Seção 5.2.

As Figuras 5.6 e 5.7 apresentam uma parte da descrição do metamodelo XML em XMI. O pacote *XMLMetamodel* contém as várias classes, atributos e associações do metamodelo. Cada elemento do metamodelo MOF possui um identificador único, representado pelo atributo *xmi.id* de XMI. Cada classe do metamodelo é representado por um elemento

Model:Class e possui alguns atributos XMI que definem a função desta classe no metamodelo. Por exemplo: o atributo lógico *isAbstract* diz se uma determinada classe pode instanciar objetos ou não. O atributo *isSingleton,* quando é verdadeiro, permite que o metamodelo contenha apenas uma instância de objeto referente à classe. O atributo *supertypes* representa a lista de superclasses de uma classe, pois o MOF permite herança múltipla. Se a classe MOF possuir atributos ou referências, esses elementos estarão dentro do elemento XMI *Model.Namespace.contents*, que representa a associação *Contains* do modelo MOF. Todas as classes de um pacote estão contidas dentro do elemento *Model:Namespace.contents* do pacote. No exemplo, a classe *XMLDocument* possui o atributo *version* e é representado pelo elemento de XMI *Model:Attribute*.

Figura 5.6 – Algumas classes e atributos do metamodelo XML descritos em XMI

Dentre as características mais importantes dos atributos estão a visibilidade e o escopo. A visibilidade é representada pelo atributo de XMI, *visibility,* que pode possuir os valores *public_vis*, *private_vis* ou *protected_vis*, mesmo conceito utilizado em linguagens de programação orientada a objetos como Java. O escopo é representado pelo atributo de XMI *scope* e pode assumir os valores *instance-scoped* e *classifier_scoped*. Estes atributos têm impactos diretos na geração das interfaces para gerenciamento dos metadados, instâncias do metamodelo. Por exemplo, apenas quando a visibilidade for pública existirá um método para alterar e consultar o atributo. Se o escopo for do tipo *instance_scoped*, o método para acesso a

tal atributo aparecerá na interface referente à classe do metamodelo, senão, o método aparecerá na interface *proxy* de tal classe.

Cada atributo ou referência possui também a multiplicidade, que define a quantidade de instâncias ou valores que tal atributo ou referência pode conter. A multiplicidade é representada em XMI pelo elemento *Model:StructuralFeature.multiplicity* que possui alguns atributos importantes como: *is_ordered* que se for verdadeiro, a coleção de valores deve ser ordenada; *lower* e *upper* definem respectivamente os números mínimo e máximo de instâncias ou valores; e *is_unique* que diz se os valores ou instâncias da coleção podem ser repetidos ou não. A multiplicidade também impacta diretamente na geração das interfaces do metamodelo. Por exemplo, se *is_ordered* é igual a verdadeiro e *upper* for maior que um, o método da interface para consultar tal atributo ou referência retornará uma coleção ordenada, que em Java é representada pela classe *java.util.List.* Se não for ordenada, retornará um *java.util.Collection,* ou seja, uma coleção não ordenada. E se *upper* for menor ou igual a um, o método retornará apenas um objeto ou valor representado pelo tipo do atributo ou referência.

Figura 5.7 – A associação XMLContains do metamodelo XML descrita em XMI

As associações do metamodelo XML são representadas em XMI pelo elemento *Model:Association.* Como explicado no Capítulo 4, toda associação no modelo MOF é binária e não tem atributos, possuindo dois elementos *AssociationEnd* referentes às respectivas classes envolvidas na associação. Uma associação não possui uma referência direta aos dois objetos do tipo *AssociationEnd*, isto é feito através da associação *Contains* do modelo MOF.

Por isso, eles aparecem dentro do elemento *Model:Namespace.contents*. No modelo MOF, todas as classes são subclasses, direta ou indiretamente, da classe *ModelElement,* inclusive a classe *AssociationEnd*. A classe *Association* é uma subclasse da classe *NameSpace*, ou seja, ela pode conter outros objetos do tipo *ModelElement*. Cada objeto do tipo *AssociationEnd* possui um objeto do tipo *Multiplicity* que define algumas regras sobre quantidade, ordenação e valores repetidos do conjunto de instâncias da associação. Ao utilizar as interfaces geradas a partir do metamodelo, o usuário tem duas opções para acessar os objetos envolvidos em uma associação: a) através do próprio objeto associação; b) através dos objetos referências correspondentes às duas classes envolvidas na associação. A interface da classe A terá uma referência aos objetos da classe B da associação e vice-versa.

Após gerar o documento XMI referente ao metamodelo, algumas correções ou adaptações podem ser realizadas mudando o próprio documento XMI. Esta tarefa é necessária porque, como foi explicado na Seção 5.2, as ferramentas que modelam UML não possuem todos os construtores da especificação MOF.

No metamodelo XML foram acrescentadas algumas *constraints* que limitam alguns tipos de agregações entre os *nodes* XML. Como explicado no Capítulo 4, o MOF utiliza OCL como linguagem para a definição de *constraints*. As *constraints* que foram adicionadas ao metamodelo XML estão na Tabela 5.2.

Essas *constraints* foram descritas em OCL e adicionadas ao metamodelo XML descrito em XMI. A Figura 5.8 apresenta um exemplo de *constraint* descrita em XMI. A ferramenta MDR ainda não suporta *constraints*, porém qualquer outra ferramenta que implemente a especificação MOF e suporte *constraints* poderia utilizá-las.

Todas as *constraints* da tabela são invariantes, ou seja, elas são verificadas sempre que o estado do objeto alvo muda, independente de qualquer pré-condição. Elas são aplicadas aos itens da coleção da associação *XMLContains.* Esses itens são acessados pela referência *objects*.

Em OCL é possível selecionar apenas alguns itens da coleção através do construtor *collection ->select(v | boolean-expression-with-v)*, onde *v* é chamado de *iterator* e *boolean-expression-with-v* é uma expressão lógica sobre *v*. Quando a *constraint* é avaliada, o conjunto retornado é formado pelos itens onde a expressão lógica *boolean-expression-with-v* é verdadeira . Para saber quais itens podem estar na coleção, é necessário verificar a que classes pertencem tais itens. No caso da *constraint C0* da Tabela 5.2*,* apenas um objeto instância de *XMLElement*

poderá fazer parte da coleção e este objeto não pode ser instância de *XMLDocument*, visto que esta classe é subclasse de *XMLElement*. A propriedade lógica *oclIsTypeOf(type : OclType):boolean* pré-definida em OCL permite verificar, para qualquer objeto do metamodelo, se *type*, passado como parâmetro, é igual a algum de seus tipos.

Tabela 5.2 - As *constraints* OCL do metamodelo XML

Contexto	Descrição Textual	OCL
XMLDocument	Um objeto do tipo XMLDocument **pode conter apenas um** objeto do tipo XMLElement.	context document : XMLDocument inv C0:document.objects->select(obj:XMLObject | (obj.oclIsTypeOf(XMLElement) and not obj.oclIsTypeOf(XMLDocument)))->size=1
XMLDocument	Um objeto do tipo XMLDocument **não pode conter** um outro objeto do tipo XMLDocument.	context document : XMLDocument inv C1:document.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLDocument)) ->isEmpty
XMLDocument	Um objeto do tipo XMLDocument **não pode conter** um objeto do tipo XMLAttribute.	context document : XMLDocument inv C2:document.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLAttribute)) ->isEmpty
XMLDocument	Um objeto do tipo XMLDocument **não pode conter** um objeto do tipo XMLCdataSection.	context document : XMLDocument inv C3:document.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLCdataSection))->isEmpty
XMLDocument	Um objeto do tipo XMLDocument **não pode conter** um outro objeto do tipo XMLEntityReference.	context document : XMLDocument inv C4:document.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLEntityReference))->isEmpty
XMLDocument	Um objeto do tipo XMLDocument **não pode conter** um outro objeto do tipo XMLNodeText.	context document : XMLDocument inv C5:document.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLNodeText))->isEmpty
XMLDocument	Um objeto do tipo XMLDocument **não pode conter mais de um** objeto do tipo XMLDocumentType.	context document : XMLDocument inv C6:document.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLDocumentType))->size<=1
XMLElement	Um objeto do tipo XMLElement **não pode conter** um objeto do tipo XMLDocumentType.	context element : XMLElement inv C7:element.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLDocumentType))->isEmpty
XMLElement	Um objeto do tipo XMLElement **não pode conter** um objeto do tipo XMLEntity.	context element : XMLElement inv C8:element.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLEntity))->isEmpty
XMLElement	Um objeto do tipo XMLElement **não pode conter** um objeto do tipo XMLDocument.	context element : XMLElement inv C9:element.objects->select(obj:XMLObject | obj.oclIsTypeOf (XMLDocument))->isEmpty

Uma *constraint* é representada em XMI pelo elemento *Model:Constraint* e possui alguns atributos como *evaluationPolicy* e *language.* O atributo *evaluationPolicy* pode possuir os valores *immediate* ou *deferred* e se refere ao momento para a verificação de violação da

constraint.

```
- <Model:Constraint annotation="" evaluationPolicy="deferred" language="OCL"
    xmi.id="C0">
    <Model.ModelElement.name
      xml:space="preserve">XMLDocumentContemUmXMLElement</Model.ModelElement.name>
    <Model.Constraint.expression xml:space="preserve">context document :
      XMLDocument inv C0:document.objects->select(obj:XMLObject |
      (obj.oclIsTypeOf(XMLElement) and not obj.oclIsTypeOf(XMLDocument) ))-
      >size=1</Model.Constraint.expression>
  - <Model:Constraint.constrainedElements>
      <Model:ModelElement xmi.idref="a3D2A1A46009D" />
    </Model:Constraint.constrainedElements>
  </Model:Constraint>
```

Figura 5.8 – Uma *constraint* do metamodelo XML descrita em XMI

O atributo *language* diz em qual linguagem está escrita a *constraint.* Por enquanto este atributo terá o valor *"OCL",* pois a especificação MOF utiliza OCL como linguagem de especificação de *constraints*.

Além dos atributos, existem também os subelementos *Model:ModelElement.name* e *Model:Constraint.expression* que armazenam, respectivamente, o nome e a própria *constraint,* e *Model:Constraint.contrainedElements* que diz a quais elementos do metamodelo tal *constraint* se aplica.

Após gerar o documento XMI do metamodelo e fazer as eventuais modificações, o próximo passo é importar o documento XMI para o repositório MOF. A seção seguinte apresenta este processo.

5.4.2 Importação do metamodelo XML em XMI para o repositório MOF

O processo de importação do documento XMI para o repositório MOF criará um conjunto de meta objetos que representarão o metamodelo dentro do repositório.

Antes de importar o metamodelo XML é necessário instanciar o metamodelo MOF: este processo criará um sub-repositório, inicialmente vazio. Após a importação do metamodelo XML para o novo sub-repositório criado, os componentes do metamodelo XML serão instâncias do metamodelo MOF. Por exemplo, a classe *XMLElement* do metamodelo XML será uma instância de *Class* do modelo MOF.

A Figura 5.9 apresenta o metamodelo XML importado para o repositório MOF, onde cada classe do metamodelo XML será instância da classe *Class* do MOF, cada associação do metamodelo XML será instância da classe *Association* do MOF, cada pacote do metamodelo

será instância da classe *Package* do MOF, e assim por diante.

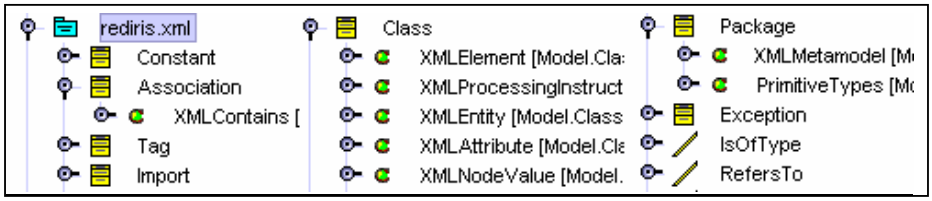

Figura 5.9 – O metamodelo XML como instância do MOF

Após ter importado o metamodelo para o repositório MOF, o usuário deve gerar as interfaces do repositório referentes ao metamodelo importado. A próxima seção descreve este processo.

5.4.3 Geração das interfaces Java para acesso ao metamodelo XML

Os usuários podem gravar metadados no repositório de duas maneiras. A primeira é importando os metadados descritos em documentos XMI. A segunda é através das interfaces geradas a partir do metamodelo. Essas interfaces permitem ao usuário criar, alterar e acessar as instâncias dos metamodelos, que são os metadados, utilizando programas Java. Como foi visto no Capítulo 4, a especificação MOF define um conjunto de regras para a geração de um conjunto de interfaces a partir de um metamodelo específico. Uma das regras diz que para cada classe do metamodelo, duas interfaces são geradas: a primeira representa as instâncias da classe, os meta objetos *instance*, e a segunda representa uma classe *proxy*, os meta objetos *class proxy*.

Listagem 5.1 - Interface XMLObject do metamodelo XML

```
public interface Xmlobject extends javax.jmi.reflect.RefObject {
    public java.lang.String getName();
    public void setName(java.lang.String newValue);
    public xmlmetamodel.Xmlelement getContainer();
    public void setContainer(xmlmetamodel.Xmlelement newValue);
}
```

A Listagem 5.1 apresenta a interface *XMLObject* gerada a partir do metamodelo XML. Essa interface oferece métodos para acessar e manipular o estado dos atributos e referências da classe do metamodelo. Por exemplo, as operações *getName* e *setName* recuperam e alteram, respectivamente, o valor do atributo *Name* da classe *XMLObject*. Esta interface herda da interface reflexiva *RefObject*.

A Listagem 5.2 apresenta duas interfaces *proxys*: uma para *XMLObject* e outra para *XMLAttribute*. Como foi visto no Capítulo 4, essas interfaces *proxys* possuem operações que

criam instâncias de suas classes correspondentes, isto se a classe não for abstrata. A classe *XMLObject* do metamodelo XML é abstrata, ela não instancia diretamente objetos, por isso a interface *XMLObjectClass* não possui operação para a criação de objetos do tipo *XMLObject*.

Além das operações para a criação dos objetos, as interfaces *proxys* possuem operações para acessar atributos e invocar as operações que possuam escopo do tipo "*classifier-scoped"*. O metamodelo XML não possui atributo ou operação deste tipo de escopo.

Listagem 5.2 – Interfaces XMLObjectClass e Xmlattribute do metamodelo XML

```
public interface XmlobjectClass extends javax.jmi.reflect.RefClass {
}

public interface XmlattributeClass extends javax.jmi.reflect.RefClass {
          public Xmlattribute createXmlattribute();
          public  Xmlattribute  createXmlattribute(java.lang.String  name,
java.lang.String value);
}
```

Para cada associação do metamodelo é gerada uma interface com um conjunto de métodos para acessar, alterar, inserir as instâncias das associações. Essas instâncias representam ligações entre os objetos, instâncias das classes do metamodelo. O metamodelo XML possui a associação *XMLContains* que diz que um *node* do tipo *XMLElement* poderá conter outros *nodes* do tipo *XMLObject*. A Listagem 5.3 apresenta a interface *XMLContains*.

Listagem 5.3 – Interface XMLContains do metamodelo XML

```
public interface Xmlcontains extends javax.jmi.reflect.RefAssociation {
       public boolean exists(xmlmetamodel.Xmlobject objects,
            xmlmetamodel.Xmlelement container);
       public java.util.List getObjects(xmlmetamodel.Xmlelement container);
       public xmlmetamodel.Xmlelement getContainer(xmlmetamodel.Xmlobject
            objects);
       public boolean add(xmlmetamodel.Xmlobject objects,
            xmlmetamodel.Xmlelement container);
      public boolean remove(xmlmetamodel.Xmlobject objects,
            xmlmetamodel.Xmlelement container);
      }
```

A interface *XMLContains* possui um conjunto de métodos que permite gerenciar as associações entre os objetos do metamodelo. Por exemplo, o método *exist* verifica se um objeto do tipo *XMLElement* contém um outro objeto do tipo *XMLObject*, enquanto que os métodos *add* e *remove,* respectivamente, adiciona e remove ligações entre os objetos do metamodelo. O método *getXMLObject* retorna uma lista de objetos do tipo *XMLObject* que estão associados a um determinado objeto do tipo *XMLElement*. O método *getContainer* retorna o objeto do tipo *XMLElement* que está associado a um determinado objeto do tipo

XMLObject, ou seja, ele verifica qual é o container do objeto do tipo *XMLObject*.

Para cada pacote do metamodelo é gerada uma interface que contém métodos para acessar todos os objetos *proxys* referentes a todas as classes e associações do metamodelo. A Listagem 5.4 apresenta a interface para o pacote *XMLMetamodelPackage*, único pacote do metamodelo XML.

Listagem 5.4 - Interface XMLMetamodelPackage do metamodelo XML

```
public interface XmlmetamodelPackage extends javax.jmi.reflect.RefPackage
 {
    public xmlmetamodel.XmlcdataSectionClass getXmlcdataSection();
    public xmlmetamodel.XmlcommentClass getXmlcomment();
    public xmlmetamodel.XmldocumentClass getXmldocument();
    public xmlmetamodel.XmldocumentTypeClass getXmldocumentType();
    public xmlmetamodel.XmlentityReferenceClass getXmlentityReference();
    public xmlmetamodel.XmlnodeTextClass getXmlnodeText();
    public xmlmetamodel.XmlobjectClass getXmlobject();
    public xmlmetamodel.XmlnodeValueClass getXmlnodeValue();
    public xmlmetamodel.XmlattributeClass getXmlattribute();
    public xmlmetamodel.XmlentityClass getXmlentity();
    public xmlmetamodel.XmlprocessingInstructionClass
                                        getXmlprocessingInstruction();
    public xmlmetamodel.XmlelementClass getXmlelement();
    //unica associação do metamodelo
    public xmlmetamodel.Xmlcontains getXmlcontains();
}
```

Após a geração das interfaces, o próximo passo será a compilação dessas interfaces e posterior instalação como um módulo da ferramenta *NetBeans.* Como foi explicado na Seção 5.3, a ferramenta MDR que implementa o MOF é um módulo da ferramenta *NetBeans*. Outros repositórios MOF exigem apenas a compilação das interfaces para que possam ser utilizadas por ferramentas cliente.

Para instalar o metamodelo XML como um módulo da ferramenta *NetBeans* é necessário criar um arquivo *jar*[14] das interfaces geradas. Depois, deve-se selecionar na ferramenta o menu *Tools->Options* e então selecionar o item *System->Modules* e utilizar o botão direito do mouse sobre o item *Modules* para escolher a opção *ADD->Module* do menu *popup*. Após, o usuário deverá escolher o arquivo *jar* a ser instalado. A próxima seção apresenta o uso do metamodelo XML através das interfaces geradas.

[14] Um arquivo compactado contendo as interfaces do metamodelo.

5.4.4 Uso do Metamodelo XML

Com o metamodelo XML instalado, o usuário poderá construir programas clientes que conectem ao repositório MOF, obtenham o pacote *XMLMetamodel* e utilizem-no na construção de novos documentos XML.

Listagem 5.5 – Código Java para criar documentos XML no repositório MOF

```
1 - MDRepository repository =
                              MDRManager.getDefault().getDefaultRepository();
2 - XmlmetamodelPackage extent =
                  (XmlmetamodelPackage)repository.getExtent("xmlmetamodel");
3 - if (extent !=null)   {
4 -         repository.beginTrans(true);
5 -         xmlmetamodel.Xmldocument docroot =
                               extent.getXmldocument().createXmldocument(
                                                    DATAWAREHOUSES.xml", 1,
             "Armazena dados sobre o esquema do data warehouse academico",
                                                "20/07/2002", "20/07/2002");
6 -         xmlmetamodel.XmlprocessingInstruction   pi   =
      extent.getXmlprocessingInstruction().createXmlprocessingInstruction(
                               "xml", "version=\"1.0\" encoding=\"UTF-8\"");
7 -         xmlmetamodel.XmldocumentType dt=
      extent.getXmldocumentType().createXmldocumentType(
                                    "\"D:\\ServerREDIRIS\\ data\\dw.dtd\"");
8 -         xmlmetamodel.Xmlelement elroot =
                    extent.getXmlelement().createXmlelement("DATAWAREHOUSES");
9 -         extent.getXmlcontains().add(pi,docroot);
10  -        extent.getXmlcontains().add(dt,docroot);
11  -    extent.getXmlcontains().add(elroot,docroot);
12  -    extent.getXmlcontains().add(
           extent.getXmlcomment().createXmlcomment("Este documento armazena
metadados sobre os esquemas dos datawarehouses"),
                                                    docroot);
13  -    repository.endTrans();
}
```

A Listagem 5.5 apresenta uma parte de um código em Java para acessar o repositório XML. A linha 1 do programa *MDRManager.getDefault().getDefaultRepository()* retorna o repositório padrão armazenado. Este repositório vai ser sempre o próprio metamodelo MOF. Após, é necessário pesquisar, no repositório padrão, o pacote *proxy* referente ao metamodelo XML. O método *repository.getExtent("xmlmetamodel"),* na linha 2, procura um determinado pacote *proxy* dentro do repositório MOF. Ele recebe o nome do pacote como parâmetro e retorna um objeto do tipo *RefPackage,* que faz parte do pacote reflexivo de JMI. Se a pesquisa ao pacote desejado for bem sucedida, o próximo passo será utilizar o pacote para gerenciar metadados descritos em XML.

Para criar qualquer objeto no metamodelo é necessário obter a referência à interface *proxy* do objeto a ser criado. Por exemplo, para criar um novo documento XML é necessário

XmldocumentClass, que é obtido pelo método *getXmldocument()* do pacote *XmlmetamodelPackage*. A interface *XmldocumentClass* possui o método para criar um novo documento XML, instância de *Xmldocument*. Da mesma maneira, para se criar um objeto, instância de qualquer classe do metamodelo, é necessário obter a referência à sua interface *proxy*. A execução do código da Listagem 5.5 cria um objeto instrução de processamento, um objeto *DocType* e um objeto elemento, que é a raiz do documento XML. Além disso, é necessário fazer as associações entre os objetos. Isto é feito através da associação *XMLContains* do metamodelo.

A Figura 5.10 apresenta o documento XML criado pela execução do código Java da Listagem 5.5.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DATAWAREHOUSES SYSTEM "D:\ServerREDIRIS\data\dw.dtd">
<!--Este documento armazena metadados sobre os esquemas dos datawarehouses-->
<DATAWAREHOUSES></DATAWAREHOUSES>
```

Figura 5.10 - Documento XML criado através da execução do código da Listagem 5.5

As linhas 1 e 2 da Listagem 5.5 variam segundo a implementação utilizada. A especificação MOF não diz como um programa cliente recupera a referência a um repositório MOF e nem como deve ser a pesquisa aos pacotes dentro do repositório. Isto é implementado segundo os critérios de quem projetou o repositório.

Além da importação dos metadados na forma de documentos XMI, e do uso das interfaces geradas a partir dos metamodelos, os usuários podem necessitar importar os metadados que estão em documentos XML, XSLT, DTD, entre outros. Isto é interessante, pois os usuários podem possuir ferramentas que já produzem metadados nestes padrões. Desta forma, basta importar os mesmos para o repositório MOF. Para isto, é necessário construir utilitários que façam o mapeamento entre os metadados descritos nos padrões originais (XML, XSLT, RDF) para os seus metamodelos correspondentes. Isto é uma alternativa a mais para intercâmbio dos metadados desses metamodelos. A próxima seção apresenta o mapeamento do padrão XML para o metamodelo XML.

5.4.5 Mapeamento XML para o Metamodelo XML

Para realizar o mapeamento entre o metamodelo XML e um documento XML foi construído

um utilitário que exporta e importa os metadados para documentos XML a partir do repositório MOF.

5.4.5.1 Importação de documentos XML

O processo de importação consiste em submeter o documento XML a um *parser* XML, que processa o documento e importa para o repositório MOF como instância do metamodelo XML. A Figura 5.11 representa este processo.

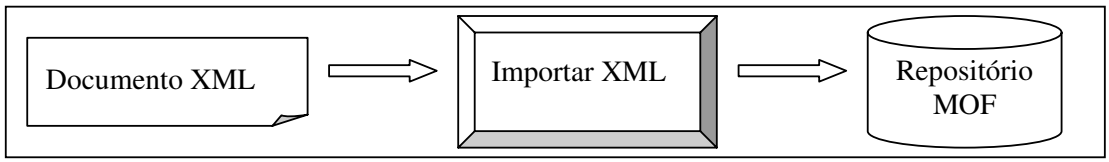

Figura 5.11 – Importação de um documento XML

O módulo Importar XML executa as seguintes tarefas:

1. Processar o documento XML, verificando se ele é bem formado, ou seja, se os dados do arquivo passado como parâmetro estão mesmo na sintaxe XML.
2. Iniciar o repositório MOF e pesquisar o repositório XML, representado pelo metamodelo XML. Esta tarefa consiste em achar uma instância da classe *MOFPackage* do metamodelo MOF cujo nome é *XMLMetamodel*.
3. Criar uma instância da classe *XMLDocument* que representa o novo documento a ser importado para o repositório.
4. Para cada elemento do documento XML, criar uma instância da classe *XMLElement*.
5. Para cada atributo do documento XML, criar uma instância da classe *XMLAttribute*.
6. Similar aos passos 4 e 5, fazer os outros mapeamentos, como comentários, entidades, instrução de processamento, entre outros.
7. Além de criar os objetos, é necessário criar as ligações entre os mesmos através da classe *XMLContains,* única associação do metamodelo XML.

A Listagem 5.6 apresenta uma parte do código Java que faz o mapeamento. O *parser* utilizado no processamento do documento XML foi o *Xerces*[15] utilizando a API SAX (*Simple API XML*). No exemplo, é apresentado o código referente ao evento *endElement* do *parser* SAX,

[15] http://xml.apache.org/

ou seja, quando o *parser* encontra o final de um elemento. Ao chegar ao final do processamento de um elemento do documento, o programa cria um objeto MOF que é do tipo *XMLElement*, que faz parte do metamodelo XML. Ele verifica também se tal elemento possui conteúdo texto, que será adicionado ao objeto MOF do tipo *XMLNodeText*.

Listagem 5.6 - Código Java para gerar instâncias do metamodelo XML a partir de documentos XML

```
public void endElement(String str, String str1, String str2)
      throws org.xml.sax.SAXException {
      String data = content.toString();
      content.setLength(0);
      Xmlelement me = (Xmlelement) parents.pop();
      if (data.trim().length() > 0) {
            XmlnodeText text = factorynodetext.createXmlnodeText(data);
            factorycontains.add(text, me); }
      Xmlelement parent = (Xmlelement) parents.peek();
      if (parent != null) {
            factorycontains.add(me, parent);
      }
}
```

5.4.5.2 Exportação de documentos XML

A exportação consiste em gerar documentos XML a partir dos metadados, instâncias do metamodelo XML. Este processo é apresentado na Figura 5.12.

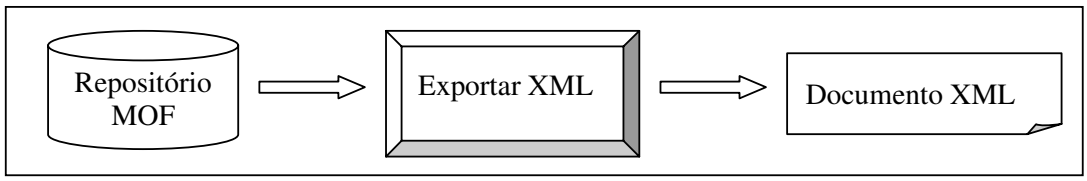

Figura 5.12 – Exportação de um documento XML

Para exportar um determinado documento é necessário seguir as seguintes etapas:

1. Inicializar o repositório MOF e pesquisar o sub-repositório XML, ou seja, o pacote *XMLMetamodel*.
2. Consultar a instância da classe *XMLDocument,* dentro do pacote, que representa o documento XML a ser exportado. A Listagem 5.7 apresenta o código Java que faz isto.
3. O objeto pesquisado no passo 2 é a raiz do documento XML. A partir dele é possível navegar para os outros objetos, gravando-os em um arquivo, através da associação *XMLContains*.

Listagem 5.7 - Código Java para pesquisar um documento XML dentro do repositório

```
Public ModelGenerate(XmlmetamodelPackage pkg, String filename) {
1     this.pkg = pkg;
2     javax.jmi.reflect.RefClass refclass =  pkg.refClass("XMLDocument");
3     java.util.Collection c = refclass.refAllOfClass();
4     java.util.Iterator iter = c.iterator();
      while(iter.hasNext()){ //procura o documento XML Correto
6         xmlmetamodel.Xmldocument docroot =
                      (xmlmetamodel.Xmldocument)iter.next();
7      if (docroot.getName().equals(filename)){
8           generateXML(docroot);
       }
}
```

O método *refClass* (linha 2) da interface reflexiva *Refpackage,* retorna a interface *proxy* de uma determinada classe do metamodelo que é passada como parâmetro. Neste caso, a classe é *XMLDocument* e a interface retornada é *XMLDocumentClass*. O método *refAllOfClass* retorna uma lista de todos os objetos instância de uma determinada classe do metamodelo, no exemplo *XMLDocument*. A próxima tarefa é pesquisar, através do atributo *name*, o objeto na lista de objetos e passar para o método *genarateXML* (linha 8) que navega para os outros objetos através da associação *XMLContains,* imprimindo-os para um arquivo. Por exemplo, a Listagem 5.8 apresenta uma parte do método *genarateXML* que é executada quando o programa encontra um atributo.

Listagem 5.8 - Código para gerar atributos XML a partir do metamodelo XML

```
If (obj instanceof Xmlattribute){
      Xmlattribute att  = (Xmlattribute)obj;
      Out.write(" " + att.getName() + "=\"" + att.getValue()+"\"");
}
```

5.5 Considerações Finais

Este capítulo apresentou a primeira parte da solução de metadados. Inicialmente foram apresentadas as vantagens de utilizar os padrões do W3C para descrição, armazenamento e representação de metadados modelados em MOF. Logo após, foi apresentado o metamodelo XML, utilizado para representar documentos XML em repositórios MOF. Foram apresentados e discutidos os passos necessários na modelagem e implementação de um metamodelo, como a criação do metamodelo em UML, geração do documento XMI referente a tal metamodelo, realização de eventuais alterações no documento, importação do documento para o repositório, geração, compilação, instalação e uso das interfaces do metamodelo. O próximo capítulo apresenta os outros metamodelos referentes aos padrões DTD, XSLT, RDF e RDF Schema.

6 Os metamodelos DTD, XSLT, RDF e RDF Schema

Este capítulo apresenta os metamodelos construídos a partir dos padrões DTD, XSLT, RDF e RDF Schema, que auxiliam o padrão XML no processo de validação, transformação ou interpretação dos metadados. Este capítulo apresenta os seguintes metamodelos:

- Metamodelo DTD – Armazena os documentos DTD no repositório MOF que dão suporte à validação dos metadados descritos pelo metamodelo XML, descrito no Capítulo 5;
- O metamodelo XSLT – Armazena os documentos XSLT que dão suporte ao processo de transformação dos metadados descritos pelo metamodelo XML;
- RDF e RDF Schema – Estes metamodelos armazenam os documentos RDF e RDF Schema.

6.1 Metamodelo DTD

O metamodelo DTD representa o padrão DTD. Este padrão é importante na validação das estruturas de um documento XML. Como foi visto no Capítulo 3, uma DTD contém regras de como os elementos de um documento XML devem estar organizados.

Os seguintes componentes fazem parte do metamodelo DTD :

- A classe mais geral é *DTDObject* e todas as outras classes do metamodelo herdam direta ou indiretamente desta classe. Ela possui um atributo *name*, que representa o nome do objeto. Cada objeto deve possuir um nome diferente, pois em DTD não é possível definir dois elementos com o mesmo nome. Esta regra foi escrita em OCL e está apresentada na Tabela 6.1.

Tabela 6.1 – As *constraints* OCL do metamodelo DTD

Contexto	Descrição Textual	OCL
DTDObject	**Todo objeto** instância de DTDObject **deve ter** um nome único.	context DTDObject inv C0: DTDObject.allInstances->forAll(o1, o2 | o1 <> o2 implies o1.name <> o2.name)
DTDNode	Um objeto do tipo DTDNode **não pode conter** um objeto do tipo DTDDocument.	context node : DTDNode inv C1: node.objects->select(obj : DTDObject | obj.oclIsTypeOf(DTDDocument))->isEmpty
DTDElement	**Todo objeto** instância de DTDElement **deve ter** pelo menos um filho do tipo DTDNode.	context element : DTDElement inv C2: element.objects->select(obj : DTDObject | obj.oclIsTypeOf(DTDNode))->notEmpty
DTDElement	Um objeto do tipo DTDElement **cujo atributo *operatortype* = *EMPTY*** deve possuir apenas um objeto filho, instância de DTDNode.	context element : DTDElement inv C3: (element.operatorType="EMPTY") implies (element.objects->select(obj : DTDObject | obj.oclIsTypeOf(DTDNode))->size = 1)
DTDVirtualElement	Um objeto do tipo DTDVirtualElement **não pode conter** um objeto do tipo DTDAttribute.	context elementv : DTDVirtualElement inv C4: elementv.objects->select(obj : DTDObject | obj.oclIsTypeOf(DTDAttribute))->isEmpty

- A classe *DTDAttribute* representa a definição de um atributo na DTD. Esta classe possui dois atributos, além de *name*, que é herdado de *DTDObject*. O atributos *attributeType* e *useType* são tipos enumerados e são representados pelo *EnumerationType* do MOF. *AttributeType* representa o tipo do atributo definido pela DTD. Os tipos para atributos suportados por DTD são: CDATA, que representa uma cadeia de caracteres; ID, que representa um identificador único no documento XML utilizado e possui algumas regras para a formação dos seus valores, por exemplo não pode começar por números e não pode possuir caracteres brancos; IDREF, que representa uma referência a um atributo ID e IDREFS que representa referências a um conjunto de atributos IDs; ENTITY, que representa uma entidade e ENTITIES que representa um conjunto de entidades; NMTOKEN, que representa uma cadeia de caracteres especiais, que não pode possuir espaços em branco e NMTOKENS, que representa um conjunto de NMTOKENS; e NOTATION, que representa uma notação[16] DTD. O atributo *useType* indica qual o uso do atributo: REQUIRED, o atributo é obrigatório; IMPLIED, o atributo não é obrigatório; DEFAULT, o atributo possui um valor padrão e FIXED, o atributo possui um valor fixo.

[16] uma declaração de notação identifica tipos específicos de dados binários externos a DTD, por exemplo: <!NOTATION GIF87A SYSTEM "GIF">

- A classe *DTDNode* representa uma classe abstrata, ou seja, ela não pode ser instanciada diretamente. Suas subclasses diretas são *DTDPCData, DTDAny* e *DTDElement*. Um objeto do tipo *DTDNode* pode conter outros objetos do tipo *DTDObject* através da associação *DTDContains.* Por esta associação, qualquer objeto do tipo *DTDNode* poderá conter outros objetos do tipo *DTDObject*.

- A classe *DTDComment* representa comentários dentro de uma DTD. Comentários podem estar associados ao próprio documento DTD ou a uma parte específica dentro da DTD como um *DTDPCData, DTDAny* ou *DTDElement*.

- A classe *DTDPCData* representa um elemento que é do tipo *PCData*. Um elemento deste tipo é utilizado no documento XML para armazenar apenas texto, não pode possuir subelementos.

- *DTDElement* representa um elemento que não é um *PCData*, ou seja, ele é definido em termos de outros elementos. Esta classe possui dois atributos que são tipos enumerados. O atributo *operatortype* pode possuir os valores SEQUENCE, CHOICE ou EMPTY. Este atributo diz se os subelementos do elemento atual serão uma seqüência, uma escolha ou não existe operador. O valor EMPTY será aplicado sempre que o número de subelementos for igual a um, se for maior que um, o valor deste atributo deverá ser obrigatoriamente uma seqüência (SEQUENCE) ou uma escolha (CHOICE). Esta regra também está descrita em OCL e é apresentada na Tabela 6.1. O atributo *multiplicity* determina a quantidade de vezes que se pode repetir os subelementos do elemento atual. Ele pode possuir os seguintes valores: ZERO_OR_ONE, os subelementos podem repetir no máximo uma vez e é representado pelo símbolo "?" no documento DTD; ZERO_OR_MORE, os subelementos podem repetir zero ou mais vezes e é representado pelo símbolo "*" na DTD; ONE, os subelementos devem ocorrer exatamente uma vez, não existe símbolo para a representação; e ONE_OR_MORE, os subelementos podem repetir uma ou mais vezes e é representado pelo símbolo "+" no documento DTD.

- *DTDAny* representa um elemento que pode conter qualquer coisa, seja texto ou outros elementos, desde que estes elementos sejam definidos também pela DTD. Quando se define um elemento do tipo *DTDAny* não há a necessidade de dizer quais elementos serão subelementos do mesmo.

- A classe *DTDDocument* é subclasse de *DTDElement* e representa o elemento raiz da DTD. Ela possui alguns atributos como a versão do documento, descrição, data de criação e última alteração.
- A classe *DTDVirtualElement* não faz parte do padrão DTD. Os objetos instâncias destas classes não são mapeados diretamente para um documento DTD. Esta classe foi adicionada ao metamodelo para auxiliar na construção das expressões que fazem a composição dos elementos e será explicada posteriormente.

A Figura 6.1 apresenta o metamodelo DTD na notação UML. As classes cinza são classes abstratas. Alguns tipos enumerados foram modelados inicialmente como classes (*DTDAttributeType*, *DTDAttributeUseType*, *DTDElementOperatorType* e *DTDElementMultiplicityType*) mas, depois, após gerar o documento XMI referente ao metamodelo, estas classes foram mudadas para tipos enumerados de MOF.

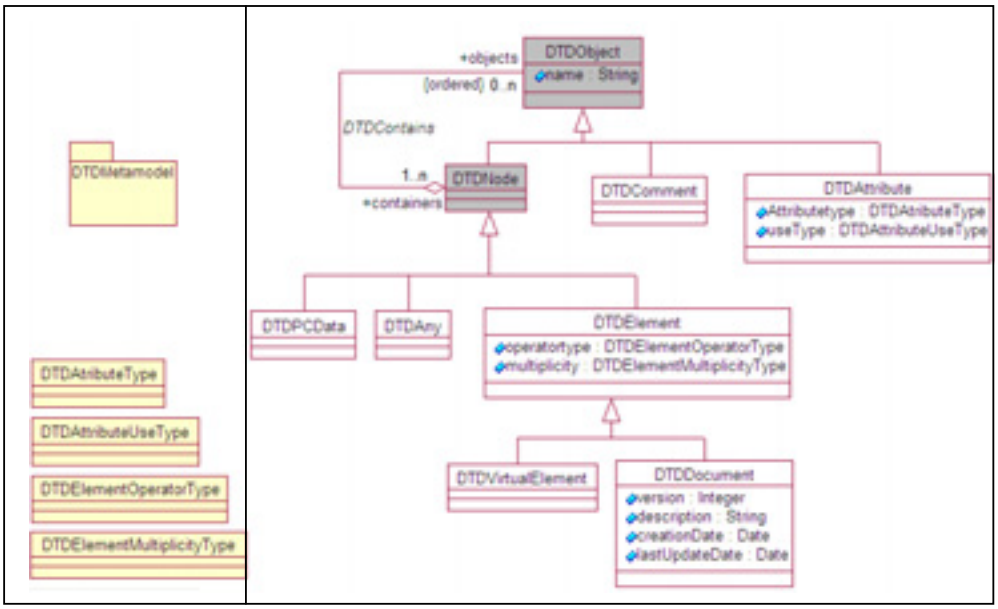

Figura 6.1 - O metamodelo DTD proposto – DTDMetamodel

A Figura 6.2 apresenta o exemplo do tipo enumerado *DTDAttributeType* descrito em XMI.

```
- <Model:EnumerationType xmi.id="a3D2A1CA403AF" name="DTDAttributeUseType"
    annotation="" isAbstract="false" isLeaf="true" isRoot="true"
    visibility="public_vis">
    <Model:EnumerationType.labels>REQUIRED</Model:EnumerationType.labels>
    <Model:EnumerationType.labels>IMPLIED</Model:EnumerationType.labels>
    <Model:EnumerationType.labels>DEFAULT</Model:EnumerationType.labels>
    <Model:EnumerationType.labels>FIXED</Model:EnumerationType.labels>
  </Model:EnumerationType>
```

Figura 6.2 – O tipo enumerado DTDAttributeUseType do metamodelo DTD descrito em XMI

O padrão de mapeamento de MOF para Java através de JMI define como um tipo enumerado de MOF é implementado em Java. A classe que representará o tipo enumerado não possui um método *create* público e cada *label* do tipo enumerado é transformado em uma constante objeto. Esta constante é pública e representa um objeto do tipo enumerado. A Listagem 6.1 apresenta o código Java para o tipo enumerado *DTDAttributeUseType* do metamodelo DTD.

Seguindo as etapas da Seção 5.2, foi gerado o documento XMI do metamodelo DTD. Foram realizadas algumas mudanças no documento para refletir melhor o metamodelo. Por exemplo, a criação dos tipos enumerados. Depois, foi gerado o conjunto de interfaces para acesso e gerenciamento dos metadados, instâncias do metamodelo XML. Finalmente, foram criadas as classes para mapeamento entre o metamodelo DTD em MOF para um documento DTD. A tarefa contrária, ou seja, importar os metadados contidos em um documento DTD não foi implementada pelo fato de não existir *parser* que manipule um documento DTD. Os *parsers* XML não possuem interfaces para manipular os documentos DTD.

Listagem 6.1 - Código Java para a implementação de tipos enumerados do MOF

```
Public final class DtdattributeUseTypeEnum implements DtdattributeUseType {
    public static final DtdattributeUseTypeEnum REQUIRED = new
            DtdattributeUseTypeEnum("REQUIRED");
    public static final DtdattributeUseTypeEnum IMPLIED = new
            DtdattributeUseTypeEnum("IMPLIED");
    public static final DtdattributeUseTypeEnum DEFAULT = new
            DtdattributeUseTypeEnum("DEFAULT");
    public static final DtdattributeUseTypeEnum FIXED = new
            DtdattributeUseTypeEnum("FIXED");
    private DtdattributeUseTypeEnum(java.lang.String literalName) {
        this.literalName = literalName; ...
}
```

Como foi dito anteriormente, a classe *DTDVirtualElement* não possui um mapeamento direto para o padrão DTD, ela foi construída apenas para auxiliar na construção das expressões.

Por exemplo, a seguinte DTD *<!ELEMENT Table (Metadata,(Data1|Data2))>* pode ser armazenada no repositório MOF como os seguintes objetos:

- Primeiro são criados o objeto do tipo *DTDDocument* que representará o elemento raiz *Table* e mais três objetos do tipo *DTDElement* que representarão os elementos *Metadata*, *Data1* e *Data2*.
- É necessário criar ainda um objeto do tipo *DTDVirtualElement* que representa a expressão *(Data1|Data2)*.

A Listagem 6.2 apresenta o código Java para criar o documento DTD. Para cada elemento

criado em uma DTD é necessário dizer se os filhos de tal elemento serão uma seqüência, escolha, ou nenhuma das alternativas. Este último caso será utilizado somente quando o número de filhos de tal elemento for igual a um. Se os filhos de um determinado elemento utilizam mais de um tipo de operador (SEQUENCE, CHOICE, EMPTY), ou mais de um tipo de multiplicidade, então devem ser criados elementos do tipo *DTDVirtualElement* que agrupem estes subelementos até ficar apenas um tipo de operador e um tipo de multiplicidade. Por exemplo, os subelementos de *Table* são *(Metadata,(Data1|Data2))*. Estes subelementos utilizam dois tipos de operadores (SEQUENCE e CHOICE), desta maneira esta expressão deve ser decomposta em duas expressões para ser armazenada no repositório : *<!ELEMENT Table (Metadata, ElementoVirtual)>* e *<!ELEMENT ElementoVirtual (Data1|Data2)>*. O utilitário que implementa a exportação de documentos DTD ignora os elementos virtuais, ou seja, eles são substituídos por seus subelementos que não são elementos virtuais.

Listagem 6.2 - Código Java para criar uma expressão em uma DTD utilizando elemento virtual

```
1 – Dtddocument docroot =
      fdoc.createDtddocument("Table",DtdelementOperatorTypeEnum.SEQUENCE,
      DtdelementMultiplicityTypeEnum.ONE,1,"","29/10/2002","29/10/2002");
2 – Dtdelement e1 =  felement.createDtdelement("Metadata",
                              DtdelementOperatorTypeEnum.SEQUENCE,
                              DtdelementMultiplicityTypeEnum.ONE);
3 – Dtdelement e2 =  felement.createDtdelement("Data1",
                              DtdelementOperatorTypeEnum.SEQUENCE,
                              DtdelementMultiplicityTypeEnum.ONE);
4 – Dtdelement e3 = felement.createDtdelement("Data2",
                              DtdelementOperatorTypeEnum.SEQUENCE,
                              DtdelementMultiplicityTypeEnum.ONE);
5 – DtdvirtualElement  ev1 =  felementV.createDtdvirtualElement("DataV",
                                    DtdelementOperatorTypeEnum..CHOICE,
                                    DtdelementMultiplicityTypeEnum.ONE);
//faz as ligações entre os objetos
6 – factorycontains.add(e1,docroot);
//adiciona o elemento ev1 ao elemento raiz
7 – factorycontains.add(ev1,docroot);
// adiciona o elemento e2 ao elemento virtual ev1
8 – factorycontains.add(e2,ev1);
// adiciona o elemento e3 ao elemento virtual ev1
9 – factorycontains.add(e3,ev1);
```

Os documentos DTD armazenados no repositório MOF são muito importantes na validação dos metadados armazenados no metamodelo XML. A próxima seção apresenta um metamodelo para o padrão RDF.

6.2 Metamodelo RDF

Foi visto no Capítulo 3 que RDF é o padrão oficial do W3C para a descrição de metadados.

Foi dito que os dados RDF são descritos em XML e podem ser apresentados em forma de grafos, onde cada *node* é um recurso e cada arco é um relacionamento entre esses recursos. Os recursos podem ser literais ou não. Em RDF um *statement* é um relacionamento triplo entre um recurso, uma propriedade e o seu valor, que é um outro recurso. Como foi visto no Capítulo 4, o MOF não permite relacionamentos ternários, desta forma, um *statement* foi modelado como uma classe e não como uma associação. Através do metamodelo RDF, os usuários poderão gerenciar metadados RDF no repositório MOF.

O metamodelo RDF está apresentado na Figura 6.3 e mostra estes conceitos que são apresentados a seguir:

- *RDFResource* é a classe mais genérica do metamodelo RDF e possui um atributo *uri,* que identifica unicamente cada recurso do metamodelo. Todo objeto instância do metamodelo é um recurso.
- *RDFLiteral* - é a classe que representa um literal, subclasse de *RDFResource*. Ela possui um atributo *literaltype*, do tipo *EnumerationType*, que representa o tipo do literal. Foram modelados apenas os tipos básicos como *integer*, *string*, *boolean* e *decimal*.
- *RDFNoLiteral* – a classe que representa um recurso que não é literal.
- *RDFDocument* – É a classe que representa a raiz de um documento RDF. Também é subclasse de *RDFResource*. Ela possui alguns atributos referentes ao documento RDF como a versão, descrição, data de alteração e criação. Para cada documento RDF armazenado no repositório haverá um objeto raiz, instância de *RDFDocument*.
- *RDFContainer, RDFSeq, RDFBag, RDFAlt* – estas classes representam *containers* RDF. Os contanainers RDF oferecem uma maneira de manipular coleções de recursos. A classe *RDFContainer* é abstrata e superclasse de *RDFSeq*, *RDFBag* e *RDFAlt*. A classe *RDFSeq* representa um conjunto de recursos ordenados. *RDFBag* representa um conjunto de recursos não ordenados e *RDFAlt* representa um conjunto de recursos distintos.
- *RDFStatement* – é a classe que representa um *statement* RDF.
- *RDFProperty* - As propriedades utilizadas nos relacionamentos entre os recursos são representadas por esta classe no metamodelo RDF.

- *RDFNameSpace* – Esta classe representa os namespaces utilizados no documento RDF. RDF utiliza *XML Namespace* para permitir que as suas *statements* referenciem um vocabulario RDF particular. Normalmente, todo documento RDF possui pelo menos um namespace, que é o do próprio RDF.

Além das classes, as associações deste metamodelo são descritas a seguir:

- *RDFDocNamespace* – Representa a associação entre os documentos RDF e os *namespaces* utilizados pelos mesmos. Ou seja, através desta associação é possível saber quais *namespaces* um determinado documento RDF utiliza. Ou, ao contrário, quais documentos RDF utilizam um determinado *namespace*. Um documento RDF pode estar associado a *nenhum* ou *n namespaces*, e vice-versa.

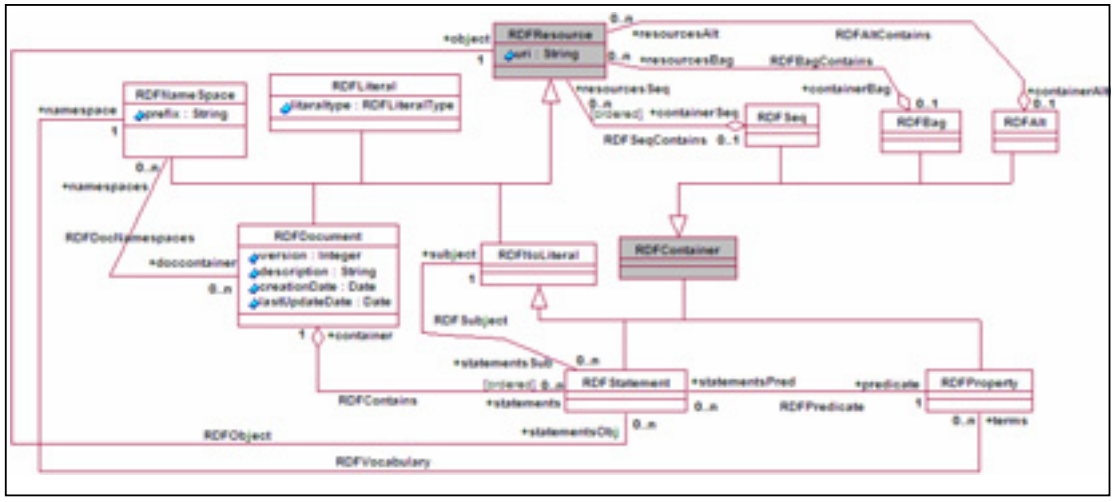

Figura 6.3 – O metamodelo RDF proposto – RDFMetamodel

- *RDFContains* – Representa a associação entre os *statements* e o seu documento *container*. Um documento RDF é composto por *nenhum* ou *n statements*. Um *statement* deve estar contido em apenas um documento RDF.

- *RDFVocabulary* – Representa a associação entre um determinado *namespace* e o seus termos. Um determinado *namespace* possui *nenhum* ou *n* termos definidos nele. Estes termos estão sendo utilizados através das propriedades. Um termo, através de uma propriedade, deve estar contido em apenas um *namespace*. Note-se que, para saber quais termos de um determinado *namespace* são utilizados em um determinado documento RDF, é necessária uma consulta mais elaborada. É necessário fazer uma interseção entre o conjunto de termos do *namespace* e o conjunto dos termos

utilizados pelas propriedades que fazem parte do documento RDF.

- *RDFSubject* - é a associação entre um *statement* e um recurso não literal. Cada *statement*, instância da classe *RDFStatement*, deve obrigatoriamente estar associado a um objeto, instância de *RDFNoLiteral* que será seu *subject*. Por esta associação pode-se consultar também, quais *statements* possuem um dado objeto do tipo *RDFNoLiteral* como *subject*.
- *RDFObject* – é a associação entre um *statement* e um recurso. Cada *statement*, instância da classe *RDFStatement,* deve obrigatoriamente estar associado a um objeto, instância de *RDFResource,* que será seu *object*. Por esta associação pode-se consultar também, quais *statements* possuem um dado objeto do tipo *RDFResource* como *object*.
- *RDFPredicate* – é a associação entre um *statement* e uma propriedade. Cada *statement*, instância da classe *RDFStatement,* deve obrigatoriamente estar associado a um objeto, instância de *RDFPredicate,* que será seu *predicate*. Por esta associação pode-se consultar também, quais *statements* possuem um dado objeto do tipo *RDFPredicate* como *predicate*.
- *RDFSeqContains, RDFBagContains e RDFAltContains* – Representa as associações entre os container e os recursos. Foi construída uma associação para cada tipo de *container,* pois todas possuem características diferentes. Por exemplo, *RDFSeqContains* é ordenada e pode conter valores repetidos, as interfaces Java terão estruturas do tipo *java.util.list* para representar armazenar o conjunto de recursos de um determinado objeto *RDFSeqContains*. Já *RDFBagContains* não é ordenada, assim, o conjunto de recursos serão armazenados em uma estrutura *java.util.collection*. A Tabela 6.2 apresenta o mapeamento dos *Containers* RDF para o metamodelo RDF.

Tabela 6.2 - Mapeamento dos *containers* RDF para o MOF

Atributos/Valores MOF	Is_ordered	Is_unique	Interfaces JMI
Seq	True	False	*java.util.List*
Bag	False	False	*java.util.Collection*
Alt	False	True	*java.util.Collection*

Após gerar as interfaces através do mapeamento *MOF-> Java* com JMI, foram construídas classes utilitárias que fazem o mapeamento de documentos RDF para o metamodelo RDF e

vice-versa. Para fazer a exportação é necessário consultar o documento dentro do repositório MOF, consultar as *statements,* através da associação *RDFContains,* e gerá-las em XML.

A próxima seção apresenta um metamodelo para o padrão RDF Schema.

6.3 Metamodelo RDFS - RDF Schema

Como foi visto no Capítulo 3, RDF Schema é o padrão criado pelo W3C para permitir o usuário criar os seus próprios vocabulários RDF. Desta forma, cada aplicação poderá ter seu próprio vocabulário e compartilhá-lo com outras aplicações. A seguir, é apresentada uma descrição dos componentes do metamodelo RDFS. A representação em UML é apresentada na Figura 6.4.

- *RDFSResource* – É a classe que representa um recurso em RDF Schema. Um recurso pode ser qualquer entidade que possa ser identificada por uma *URI (Uniform Resource Identifier).* Esta classe possui dois atributos: *uri*, que representa a URI do recurso, e *comment* que representa algum comentário que possa ser realizado sobre tal recurso.
- *RDFSClass* – Representa uma classe em RDF Schema. Uma classe em RDF Schema corresponde ao conceito genérico de tipo ou categoria, similar ao conceito utilizado em linguagens de programação.

Figura 6.4 – O metamodelo RDFS proposto – RDFSMetamodel

- *RDFSProperty* - Representa um propriedade em RDF Schema.
- *RDFSDocument* – É a classe que representa o documento RDF Schema. Todo documento RDFS possui um objeto instância desta classe.
- *RDFSLabel* – Esta classe representa um *label* em RDF Schema. Cada recurso possui

um ou mais *labels*, que é a forma como este recurso é representado numa determinada língua, por exemplo, português, inglês, entre outras.

- *RDFSNamespace* - Representa um namespace em RDF Schema. Similar à classe *RDFNamespace* de RDF.

As propriedades de RDF Schema foram mapeadas para associações em MOF e são descritas a seguir:

- *RDFSType* – Esta associação representa a propriedade *type* e indica que um recurso, instância da classe *RDFSResource,* é membro de uma classe instância de *RDFSClass*.
- *RDFSSubClassOf* - Esta associação representa a propriedade *SubClassOf* de RDF Schema. RDF Schema permite herança múltipla, ou seja, uma classe pode possuir várias super-classes.
- *RDFSSubPropertyOf* - Representa a propriedade *SubPropertyOf* de RDF Schema e especifica que uma determinada propriedade RDF é uma especialização de uma outra propriedade. Desta forma, a associação *RDFSSubpropertyOf* representa uma associação entre dois objetos, instâncias da classe *RDFSProperty*.
- *RDFSDocNamespaces* - Representa a associação entre os documentos RDFS e os *namespaces* utilizados pelos mesmos. Ou seja, através desta associação é possível saber quais *namespaces* um determinado documento RDFS utiliza. Ou ao contrário, quais documentos RDFS utilizam um determinado *namespace*. É similar à associação *RDFDocNamespaces* do metamodelo RDF.
- *RDFSIsDefinedBy* – Todo recurso que é declarado no documento RDFS possui um namespace. Em RDFS isto é declarado através da propriedade *isDefinedBy*. A associação *RDFSIsDefinedBy* representa esta propriedade, ou seja, todo recurso, instância da classe *RDFSResource,* está associado a um *namespace* instância da classe *RDFSNamespace*.
- *RDFSContains* – Esta associação liga os recursos, instâncias da classe *RDFSResource* e o seu documento *container*, instância da classe *RDFSDocument*. Um documento RDFS pode conter *zero* ou *n* recursos definidos.
- *RDFSRange* – representa a propriedade *range* de RDFS. Esta propriedade restringe a qual classe uma propriedade se aplica. *RDFSRange* representa a associação entre um

objeto instância da classe *RDFSClass* e outro instância da classe *RDFSProperty* do metamodelo RDFS.

- *RDFSDomain* – representa a propriedade *domain* de RDFS. Esta propriedade restringe os valores que uma propriedade pode possuir. *RDFSDomain* representa a associação entre um objeto, instância da classe *RDFSProperty,* e um recurso, instância da classe *RDFSResource*, visto que qualquer recurso pode ser valor de uma propriedade.
- *RDFSLabelContains* – Esta associação diz quais *labels* estão associados a um determinado recurso. Como foi visto, um recurso possui vários *labels*. Cada um representa como tal recurso é apresentado em uma determinada língua.

Após construir o metamodelo RDFS, seguindo as etapas apresentadas na Seção 5.2, foi gerado um conjunto de interfaces, através do mapeamento *MOF->Java,* para que clientes possam construir seus vocabulários RDF Schema no repositório MOF, utilizando programas Java.

A próxima seção discute um metamodelo MOF para o padrão XSLT.

6.4 Metamodelo XSLT

Como foi visto no Capítulo 3, o padrão XSLT é utilizado na extração e transformação de conteúdo XML. XML concentra-se apenas na estrutura do documento, sem considerar a forma como a informação é apresentada. Porém, para a apresentação desses documentos, é necessário um formato de apresentação. XSLT facilita este processo e possui também como características:

- Conversão de uma classe de documentos XML em outra, com estruturas diferentes,
- Extração de informação a partir de documentos XML,
- Publicação de grandes conjuntos de documentos.

O metamodelo XSLT apresentado é composto por um conjunto de classes, associações, tipos enumerados, entre outros, que representam um subconjunto do padrão XSLT em MOF. É um subconjunto, pois uma grande parte dos elementos XSLT não foi modelada em MOF, eles serão representados no metamodelo XSLT como *nodes*, semelhante ao metamodelo XML. Um documento XSLT é composto por um elemento *stylesheet*. Este possui um conjunto de outros elementos que são filhos imediatos, chamados *top-level*. Os elementos *top-level* foram

modelados como classes no metamodelo MOF. Dentre esses elementos está o *template*. Um documento XSLT possui um conjunto de *templates*. Cada um possui duas partes: a) um "*pattern*" ou "*matching part*", que identifica o *node* XML no documento e; b) um "*action*", que contém o formato ou estilo a ser aplicado no *node* XML. O *"pattern"* ou *"matching part"* foi modelado como um atributo da classe *XSLTTemplate*, enquanto que o *action* foi modelado como uma associação entre as classes *XSLTNode* e *XSLTTemplate*. Uma seção *Action* pode conter tanto elementos que pertençam ao padrão XSLT quanto outros elementos que não pertençam. Desta forma, foi modelado como conteúdo de um *action* um conjunto de *nodes*, que possui como única restrição estar na sintaxe XML e ser bem formado. Um *XSLTNode* é semelhante a um *XMLNode* modelado no metamodelo XML. Uma das alternativas seria utilizar as mesmas classes do metamodelo XML para representar esses *nodes* que fazem parte de um *action* XSLT. Porém, esta alternativa teria como pré-condição o compartilhamento dessas classes através de um pacote comum a ambos os metamodelos, pois sem isto não seria possível construir associações entre objetos cujas classes pertencem a pacotes diferentes e que não possuam um superpacote comum. No início do projeto, optou-se por construir cada metamodelo em um pacote diferente.

A seguir é apresentada uma descrição dos principais componentes do metamodelo XSLT.

- *XSLTObject* é uma classe abstrata e superclasse de todas as outras classes do metamodelo.
- *XSLTStylesheet* é a classe que representa o elemento raiz de um documento XSLT. Cada documento XSLT armazenado no repositório MOF será representado por um objeto instância desta classe. Ela possui alguns atributos importantes, como o atributo *id* (identificador do documento); os atributos *docversion* e *xsltversion* representam, respectivamente, as versões do documento XSLT e a do padrão XSLT utilizado.
- *XSLTContent* representa o conteúdo de um documento XSLT. Ela é abstrata e possui como subclasses todos as classes referentes aos elementos *top-level* do padrão XSLT.

Os elementos *top-level* do padrão XSLT foram transformados em classes, que são subclasses de *XSLTContent.* A seguir é apresentada uma breve descrição dessas classes.

- *XSLTKey* – esta classe representa as *Keys* do padrão XSLT. As *keys* representam uma maneira de trabalhar com documentos que possuem referências implícitas entre seus elementos. Os atributos ID, IDREF e IDREFs do padrão XML suportam este

mecanismo explicitamente [CLA1999]. Uma *Key* XSLT é representada por três valores: *name*, *match* e *use*, todos representados como atributos da classe *XSTLKey* do metamodelo XSLT.

- *XSLTVariable* e *XSLTParam* – Essas duas classes representam, respectivamente, as variáveis e parâmetros do padrão XSLT. Essas classes representam valores que estão associados a uma variável ou parâmetro. Elas possuem os atributos *name* e *expresssion*, que são respectivamente o nome da variável ou parâmetro e a expressão que retornará o valor de tal variável ou parâmetro [CLA1999].
- *XSLTInclude e XSLTImport* - São classes que representam os mecanismos de combinação de *stylesheets* do padrão XSLT. Ambas possuem um atributo *href* que representa a URI referente a *stylesheet* a ser incluída ou importada [CLA1999].
- *XSLTPreservespace e XSLTStripspace* – estas classes representam os elementos *top-level* do padrão XSLT *strip-space* e *preserve-space*. Ambos possuem um atributo *elements* que representará a lista de elementos do documento XML onde são ou não preservados espaços em branco. Essa lista de elementos é armazenada como uma *string* onde os elementos estão separados por espaços em branco [CLA1999].
- *XSLTDecimalFormat* – Esta classe possui atributos que controlam a formatação dos números decimais no documento. Estes atributos são utilizados pela função *format-number* do padrão XSLT.
- *XSLTNamespaceAlias* – Esta classe representa o elemento *top-level namespace-alias* que declara uma URI de um *namespace* como um *alias* de um outro *namespace* [CLA1999].
- *XSLTOutput* – É a classe que representa o elemento *top-level output* de XSLT, e possui um conjunto de atributos que configura o tipo de saída do processador XSLT. Um dos atributos desta classe é *method,* que foi modelado como um *EnumerationType* de MOF e pode possuir os valores xml, html e txt.
- *XSLTAttributeSet* – Esta classe representa um conjunto de atributos que podem ser utilizados juntos através de um elemento do elemento XSLT *attribute-set*.

Além das classes, o metamodelo possui também as associações que são descritas a seguir:

- *XSLTContentContains* representa uma associação entre a classe *XSLTStylesheet*, referente à raiz do documento XSLT, e o seu conteúdo, a classe *XSLTContent*. É um

relacionamento *um* para *n*, pois um objeto do tipo *XSLTStylesheet* pode conter n objetos do tipo *XSLTContent.*

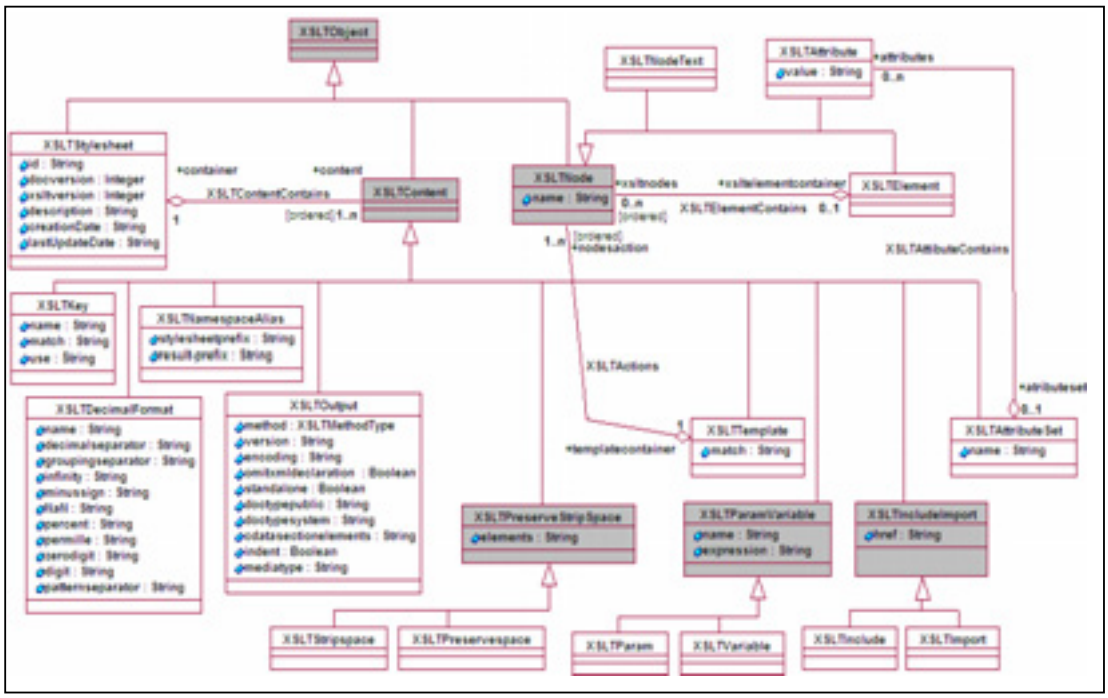

Figura 6.5 – O metamodelo XSLT proposto – XSLTMetamodel

- *XSLTActions* – Associação entre as classes *XSLTTemplate* e *XSLTNode*. Representa o conjunto de *nodes* que faz parte de uma seção *action* de um determinado *template* XSLT. Ela tem multiplicidade *um* para *n*, ou seja, uma *template* está associada a *n nodes*, formando um *action*. Um node está associado a apenas um *template*. É uma associação ordenada, pois a ordem dos *nodes* que formam o *action* é importante para o resultado final do processamento XSLT.

- *XSLTElementContains* – Esta associação é semelhante a *XMLContains* do metamodelo XML. *XSLTElementContains* associa *um* objeto instância de *XSLTElement* a *n* objetos instâncias da classe *XSLTNode*. Com esta associação, elementos XSLT podem conter outros elementos, atributos ou *nodes* texto. É uma associação ordenada, pois elementos devem ser mantidos na ordem, apesar dos atributos não serem necessários.

- *XSLTAttributeContains* – Representa a associação entre as classes *XSLTAtributeSet* e *XSLTAttribute*. É uma associação não ordenada, pois a ordem dos atributos não é importante.

A Figura 6.5 apresenta o metamodelo XSLT na notação UML. As classes em cinza são abstratas.

Seguindo a metodologia apresentada na Seção 5.2, foram geradas as interfaces a partir do metamodelo e compiladas para que ferramentas clientes consigam construir documentos XSLT no repositório MOF.

6.5 Os metamodelos e a Arquitetura de metadados da OMG

A Tabela 6.3 apresenta os metamodelos na *OMG Meta Data Architecture*.

Tabela 6.3 – A arquitetura de metadados da OMG e os metamodelos propostos

Nível MOF	Termos Utilizados	Exemplos
M3	Meta-Metamodelo	Modelo MOF
M2	Metamodelo, Meta-Metadados	XML, DTD, RDF, RDF Schema, XSLT.
M1	Modelos, Metadados	Metadados RDF, Metadados em XML, Metadados em RDF Schema, Metadados em XSLT.
M0	Dados, Objetos	Dados

Os metamodelos farão parte da camada M2 e terão como instâncias os metadados da camada M1. A seguir será apresentado, para cada metamodelo, o que representará a camada M2, M1 e M0.

- Para o metamodelo XML na camada M2, a camada M1 será formada por documentos XML e a camada M0 será formada por dados descritos pelos metadados na camada M1. Por exemplo, um documento XML que armazena um esquema relacional. Os dados serão tabelas do usuário armazenados no Banco de Dados, os metadados serão o esquema relacional armazenado no documento XML, e o metamodelo que descreve os metadados será o XML.
- Para o metamodelo DTD na camada M2, a camada M1 será formada por documentos DTD que descreverão as estruturas de documentos XML na camada M0. Por exemplo, a DTD que descreve a estrutura dos documentos XML que armazenam esquemas relacionais. Os dados na camada M0 serão XML, os metadados na camada M1 serão

DTD, e o metamodelo será o metamodelo DTD.

- Para o metamodelo XSLT na camada M2, a camada M1 será formada por documentos XSLT que descrevem regras para transformação dos documentos XML na camada M0. Por exemplo, o documento XSLT que contém regras para transformar os esquemas relacionais dos documentos XML para instruções SQLs, a fim de serem executadas no SGBD.
- Para o metamodelo RDF na camada M2, a camada M1 será formada pelos documentos RDF que descrevem recursos na camada M0, que podem ser XML, HTML, TXT ou qualquer outro padrão. Por exemplo, um documento RDF que possui metadados sobre uma determinada página HTML. A página HTML estará na camada M0, os dados RDF estarão na camada M1 e o metamodelo RDF na M2.
- Para o metamodelo RDFS na camada M2, a camada M1 será formada pelos documentos RDFS que definem os vocabulários utilizados pelos documentos RDF na camada M0. Por exemplo, o documento RDFS que contém o padrão para descrição de documentos Web Dublin Core [BMB2002]. Os documentos RDF que utilizam este vocabulário estarão na camada M0.

Os conceitos de dados e metadados são relativos, e dependem da aplicação. Dependendo do ponto de vista, um determinado padrão poderá estar em uma camada ou em outra. Por exemplo, do ponto de vista do metamodelo XML, os documentos XML estão na camada M1. Já em relação aos outros metamodelos, estes mesmos documentos estão na camada M0. Os documentos RDF estão na camada M1 em relação ao metamodelo RDF, enquanto que para o metamodelo RDFS estes mesmos documentos estão na camada M0.

6.6 Considerações Finais

Este capítulo apresentou os padrões DTD, XSLT, RDF e RDF Schema modelados em MOF. Foi apresentada uma descrição dos componentes de cada metamodelo, e por último foram realizadas algumas considerações sobre os metamodelos construídos e a arquitetura de metadados da OMG.

Qualquer ferramenta que suporte o padrão MOF poderá importar esses metamodelos e gerenciar os dados e metadados descritos por esses padrões. Esses metamodelos poderão ser melhorados, acrescentando-se novas funcionalidades como suporte a *namespace* [BTH1999],

XLink [XLP2000] e XPointer [XLP2000]. Com *XLink*, por exemplo, é possível construir relacionamentos entre objetos de documentos XML diferentes. Além disso, novos metamodelos poderão ser acrescentados ao repositório como o XML Schema e XQuery que não foram abordados neste trabalho.

7 Estudo de Caso

Os capítulos 5 e 6 apresentaram um conjunto de metamodelos que representam os padrões do W3C para representação de metadados. Foi gerado um conjunto de documentos XMI referentes a tais metamodelos. A ferramenta MDR importou tais documentos XMI para o repositório MOF. Até então, os metamodelos eram instâncias do MOF. Logo após foram geradas as interfaces, através do mapeamento *MOF -> Java* referentes a tais metamodelos. Essas interfaces foram compiladas e instaladas como um módulo da ferramenta *Netbeans*. Após essas atividades, foi possível instanciar tais metamodelos, ou seja, gerar metadados. O objetivo deste capítulo é apresentar alguns exemplos para facilitar o entendimento desta dissertação.

Neste capítulo serão apresentados dois exemplos ilustrando como o repositório poderá ser utilizado por outras ferramentas para o gerenciamento de metadados descritos pelos padrões modelados.

O primeiro exemplo a ser apresentado se refere ao gerenciamento dos metadados gerados pela metodologia *Fast Cube*, abordada em outra dissertação de mestrado [SAN2002]. Neste exemplo, serão utilizados os metamodelos XML, DTD e XSLT.

O segundo exemplo utiliza o padrão RDF Schema. Como foi visto no Capítulo 3, RDF Schema define um conjunto de conceitos, classes propriedades, recursos, entre outros. E o próprio padrão é construído baseado nesses construtores. Desta forma, o próprio metamodelo RDFS poderá armazenar o padrão RDF Schema, conforme será mostrado no estudo de caso.

7.1 O Fast Cube

A metodologia *FastCube* [SAN2002] foi criada como parte de um trabalho de mestrado e tem como objetivo auxiliar os projetistas de *Data warehouse* no processo de construção e/ou prototipação de *Data marts*. Ela faz parte de um conjunto de tecnologias desenvolvidas para construção de *Data warehouse* dentro do contexto do Ambiente REDIRIS.

Os dados de entrada desta metodologia são representados por uma tabela chamada de *TabelaAmostra*. Eles são tratados e submetidos à técnicas estatísticas. Este processo gera metadados, chamados também de fragmentos, que descrevem a qualidade dos dados. Os dados e metadados envolvidos no processo de tratamento de dados, aliados aos conhecimentos adquiridos pelo projetista durante esse processo, serão utilizados na geração de um modelo de dados dimensional [KIM1998]. A Figura 7.1 mostra o modelo de classes dos dados e metadados da metodologia.

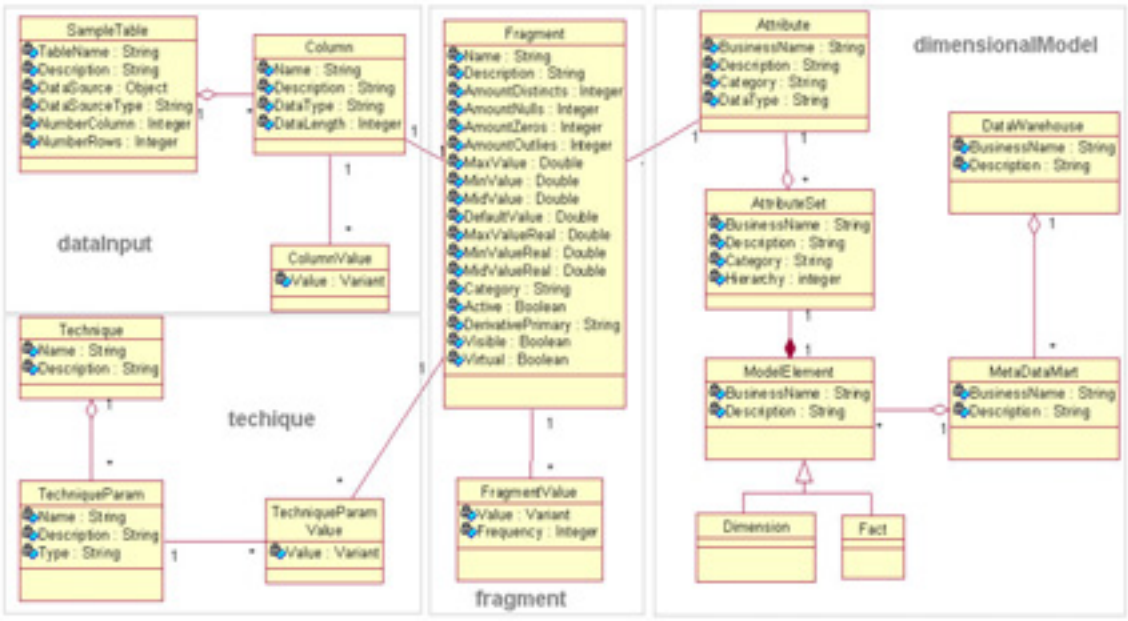

Figura 7.1 - Modelo de classes dos dados e metadados do FastCube [SAN2002]

O modelo da Figura 7.1 está dividido em pacotes. O pacote de dados de entrada *DataInputs* representa os dados e metadados da *TabelaAmostra,* ou seja, ele armazena os dados de entrada da metodologia. É formado pelas classes: *SampleTable* que representa os metadados da *TabelaAmostra*; a classe *Column* representa cada coluna da *TabelaAmostra;* e *ColumnValue* representa os valores de cada coluna. O pacote *Fragment* representa os dados

e metadados do resultado da fragmentação de cada coluna através de técnicas estatísticas. Essas classes podem ser consideradas também metadados das colunas da *TabelaAmostra*.

O pacote *Techniques* representa as técnicas e parâmetros que poderão ser usados para as transformações dos dados no processo de pré-processamento, bem como armazena os valores dos parâmetros das técnicas que foram usadas para gerar uma coluna derivada, resultado de uma transformação.

O pacote *DimensionalModels* é a representação do modelo dimensional que será gerado a partir da modelagem *FastCube*. Esse pacote possui as classes básicas para geração de um *Data Mart*. A classe *Attribute* é gerada a partir de um fragmento que se deseja utilizar para fazer parte de uma dimensão ou de um fato. A classe *AttributeSet* representa o uso de um atributo para uma determinada dimensão ou fato, isso porque um atributo derivado de um fragmento pode participar de vários *Data Marts*. Em um *Data Mart* um atributo pode participar da tabela de fato e em outro pertencer a uma dimensão. O atributo pode também ter diferentes nomes em diferentes situações.

7.1.1 Gerenciando documentos DTD no repositório MOF

Cada pacote do modelo da Figura 7.1 foi transformado em uma DTD e armazenado no repositório MOF, como instância do metamodelo DTD. Para isto foram utilizadas as interfaces do metamodelo DTD geradas a partir do mapeamento *MOF->Java*. De forma semelhante, os dados, que são instâncias do modelo da Figura 7.1, serão armazenados em XML, instâncias do metamodelo XML no repositório MOF. Esses dados serão validados pelas DTD geradas a partir dos pacotes.

A DTD da Figura 7.2 validará os dados XML que representarão os dados do pacote d*atainput* da Figura 7.1, da metodologia *FastCube*. As classes dos pacotes foram transformadas em elementos, os atributos das classes foram transformados em atributos nas DTD; e as associações foram transformadas em elementos aninhados.

Listagem 7.1 – Criando um novo documento DTD no repositório

```
Dtddocument docroot = factorydocument.createDtddocument("DataInput",
      DtdelementOperatorTypeEnum.EMPTY,
      DtdelementMultiplicityTypeEnum.ONE_OR_MORE,1,
      "DTD referente ao pacote DataInput da metodologia FastCube",
      "29/10/2002","29/10/2002");
```

A Listagem 7.1 apresenta o código Java para a criação de um novo documento DTD. Cada

documento armazenado no repositório é representado por uma instância da classe *DTDDocument*. Neste exemplo foi criado o elemento raiz da DTD apresentada na Figura 7.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT DataInput (SampleTable+)>
<!--store the metadata of the sources of data-->
<!ELEMENT SampleTable (Column+)>
<!ATTLIST SampleTable
    SampleTable_ID ID #REQUIRED
    Name CDATA #REQUIRED
    Description CDATA #IMPLIED
    DataSource CDATA #REQUIRED
    ColumnCount CDATA #REQUIRED
    RecordCount CDATA #IMPLIED
>
<!ELEMENT Column (Value*)>
<!ATTLIST Column
    Column_ID ID #REQUIRED
    Name CDATA #REQUIRED
    Description CDATA #IMPLIED
    DataType CDATA #REQUIRED
    DataLength CDATA #IMPLIED
>
<!ELEMENT Value (#PCDATA)>
```

Figura 7.2 – O documento DTD referente ao pacote *DataInput*

O elemento da DTD *DataInput* possui como filho o elemento *SampleTable*. Este por sua vez possui o subelemento *Column*. Desta maneira, é necessário criar no repositório os objetos referentes a esses elementos. Esses objetos serão instâncias da classe *DTDElement* do metamodelo DTD.

Listagem 7.2 – Criando novos elementos DTD no repositório

```
Dtdelement  e1 = factoryelement.createDtdelement("SampleTable",
      DtdelementOperatorTypeEnum.EMPTY,
      DtdelementMultiplicityTypeEnum.ONE_OR_MORE);
Dtdelement e2 =
      factoryelement.createDtdelement("Column",
      DtdelementOperatorTypeEnum.EMPTY,
      DtdelementMultiplicityTypeEnum.ZERO_OR_MORE);
Dtdpcdata p1 = factorypcdata.createDtdpcdata("Value");
```

Além dos elementos *SampleTable* e *Column,* a DTD da Figura 7.2 possui o elemento *Value*. Este é um elemento do tipo *PCData* representado no metamodelo DTD pela classe *DTDPCData*. A Listagem 7.2 apresenta o código para a criação desses objetos no repositório MOF.

Após criar os objetos referentes aos elementos da DTD, é necessário fazer as associações entre esses objetos, criando a hierarquia entre os elementos da DTD. Isto é

realizado através da associação *proxy Dtdcontains.* A Listagem 7.3 apresenta o código Java que faz esta tarefa.

Listagem 7.3 – Criando associações entre os objetos no repositório

```
factorycontains.add(factorycomment.createDtdcomment("store the metadata of
      the sources of data"),docroot);
factorycontains.add(e1, docroot);
factorycontains.add(e2, e1);
factorycontains.add(p1, e2);
```

A DTD possui ainda definições de atributos para os elementos *SampleTable* e *Column*. Os atributos são criados como instâncias da classe *DTDAttribute*. Na criação, são passados como parâmetros o nome do atributo, o tipo que representa o tipo do atributo (CDATA, ID, IDREF, entre outros) e o uso do atributo, que pode ser obrigatório (REQUIRED), não obrigatório (IMPLIED), pode possuir um valor padrão (DEFAULT) ou um valor fixo (FIXED).

Listagem 7.4 – Criando atributos DTD no repositório

```
Factorycontains.add(factoryattr.createDtdattribute("SampleTable_ID",
      DtdatributeTypeEnum.ID, DtdattributeUseTypeEnum.REQUIRED), e1);
factorycontains.add(factoryattr.createDtdattribute("Name",
      DtdatributeTypeEnum.CDATA,DtdattributeUseTypeEnum.REQUIRED),e1);
factorycontains.add(factoryattr.createDtdattribute("Description",
      DtdatributeTypeEnum.CDATA, DtdattributeUseTypeEnum.IMPLIED),e1);
```

A Listagem 7.4 apresenta o código Java para a criação dos atributos no repositório. Neste exemplo, os atributos são criados e associados a um elemento DTD.

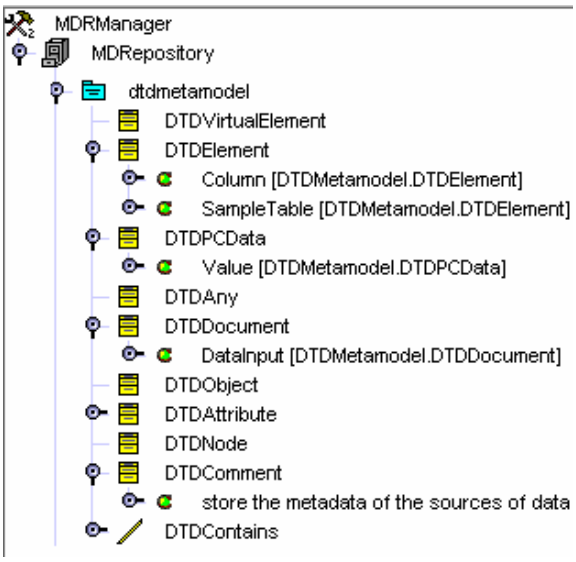

Figura 7.3 - Os metadados no repositório DTD

Quando uma DTD é criada dentro do repositório, seus objetos podem ser visualizados no *browser* de objetos da ferramenta MDR, como mostra a Figura 7.3. O *browser* permite visualizar todas as classes e associações do metamodelo assim como os seus objetos criados.

As DTD podem ser exportadas ou importadas para o repositório no formato XMI. Como foi visto no Capítulo 3, XMI é o formato para o intercâmbio de metamodelos MOF utilizando XML. São transformados para XMI os metadados das camadas M1 e M2 da arquitetura de metadados da OMG explicadas na Seção 4.1. Neste exemplo, as DTD representam metadados da camada M1 e o metamodelo DTD representa a camada M2.

Segundo a especificação de XMI [XMI2000], cada classe do metamodelo é transformada em um elemento XML e seus atributos são transformados em atributos de XML. As associações são transformadas em elementos aninhados. As multiplicidades dos dois *associationEnd* envolvidos numa associação são transformadas na multiplicidade de XML. A Figura 7.4 apresenta uma parte da DTD da Figura 7.2 descrita em XMI.

Figura 7.4 - Representação XMI da DTD da Figura 7.2

A próxima seção mostra como carregar um documento XML para o repositório MOF através do metamodelo XML.

7.1.2 Gerenciando documentos XML no repositório MOF

Da mesma maneira que foram criados os documentos DTD na seção anterior, podem ser criados documentos XML validados por tais DTD e que armazenam as instâncias do modelo apresentado na Figura 7.1.

Figura 7.5 - Dados do pacote *DataInput* em XML

As ferramentas conectam ao repositório MOF, consultam o metamodelo XML e, a partir daí, criam, consultam e alteram documentos no repositório. A Figura 7.5 apresenta um exemplo de documento XML que armazena dados referentes ao pacote *DataInput* da metodologia *FastCube*.

Figura 7.6 - Os metadados no repositório XML

Utilizando os utilitários construídos para importar documentos XML para o repositório MOF, o documento da Figura 7.5 foi importado como instância do metamodelo XML.

Através do *browser* MDR, apresentado na Figura 7.6, o usuário poderá visualizar todos os objetos referentes ao documento XML da Figura 7.5 que foram criados no metamodelo XML. O usuário poderá navegar por esses objetos através das referências. Por exemplo, o objeto *DataLength* da classe *XMLAttribute* possui uma referência para o seu *container* que, no exemplo, é o objeto *Column*, instância da classe *XMLElement*. Já o objeto *Column* possui duas referências: uma para o seu *container* e outra para os objetos filhos.

A próxima seção apresenta um documento XSLT criado no repositório MOF através do metamodelo XSLT.

7.1.3 Gerenciando documentos XSLT no repositório MOF

Como foi visto no Capítulo 3, XSLT é importante para extrair o conteúdo e efetuar transformação em documentos XML. Essas transformações podem ser para um outro documento XML com uma outra estrutura, ou para um outro formato como TXT, HTML, PDF, PS, entre outros. Utilizando o metamodelo XSLT, os usuários podem criar documentos XSLT que possuem regras para a transformação dos documentos XML, instâncias do metamodelo XML, para outros documentos.

```
- <xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <xsl:output method="text" encoding="ISO-8859-1" />
  - <xsl:template match="/DataInput/SampleTable">
      <xsl:text>create table</xsl:text>
      <xsl:value-of select="@Name" />
      <xsl:text>(</xsl:text>
    - <xsl:for-each select="Column">
      - <xsl:if test="position() != 1">
          <xsl:text>,</xsl:text>
        </xsl:if>
        <xsl:value-of select="@Name" />
        <xsl:text />
        <xsl:value-of select="@DataType" />
      - <xsl:if test="@DataType = 'VARCHAR'">
          <xsl:text>(</xsl:text>
          <xsl:value-of select="@DataLength" />
          <xsl:text>)</xsl:text>
        </xsl:if>
      </xsl:for-each>
      <xsl:text>);</xsl:text>
    </xsl:template>
  </xsl:stylesheet>
```

Figura 7.7 – Um documento XSLT para gerar instruções SQL a partir dos dados do documento XML da Figura 7.5

Um exemplo de documento XSLT é apresentado na Figura 7.7. São regras para transformar

os dados XML referentes ao pacote *DataInput* em expressões SQL de criação de tabelas em Banco de Dados.

O resultado desta transformação é a expressão SQL para criação das tabelas no Banco de Dados, que é apresentada na Figura 7.8.

```
Create table tabcovest (INSCRICAO INTEGER, NOME VARCHAR(60), NASCIMENTO INTEGER,
ESTA_NOME VARCHAR(60), ESTA_TIPO INTEGER, ESTA_REDE VARCHAR(60), MEDIA
VARCHAR(60), APROVADO INTEGER, ANO INTEGER, UF VARCHAR(60), CIDADE VARCHAR(60),
BAIRRO VARCHAR(60), SEMESTRE INTEGER, CURSO VARCHAR(60), TURNO VARCHAR(60));
```

Figura 7.8 Instruções SQL geradas a partir dos dados XML da Figura 7.5

Neste exemplo, foi criado um objeto que é instância da classe *XSLTStylesheet* que representa a raiz do documento XSLT. A Listagem 7.5 apresenta o código Java para a criação de um novo documento XSLT no repositório.

Listagem 7.5 – Criando um novo documento XSLT no repositório

```
Xsltstylesheet docroot = factorystylesheet.createXsltstylesheet(
      "documento xslt", 1, 1, "Transforma um esquema relacional em
      instruções SQL para a geração do Banco de Dados",
      "28/10/2002", "28/10/2002");
```

O documento XSLT possui um *template* e um elemento *xsl:output*. Esses elementos serão transformados em dois objetos no metamodelo XSLT que serão instâncias, respectivamente, das classes *XSLTTemplate* e *XSLTOutput*. A Listagem 7.6 apresenta o código Java para a criação dos objetos *template* e *output* e associação dos mesmos ao documento XSLT.

Listagem 7.6 – Criando os objetos *template* e *output* no repositório

```
Xsltoutput output = factoryoutput.createXsltoutput(XsltmethodTypeEnum.text,
      "versao 01", "ISO-8859-1",  false, true, "", "","", false,"");
Xslttemplate template =
      factorytemplate.createXslttemplate("/DataInput/SampleTable");
factorycontentcontains.add(output,docroot);
factorycontentcontains.add(template,docroot);
```

O objeto *output* é criado com todos os parâmetros encontrados na especificação XSLT. O objeto *template* é criado, passando como parâmetro a expressão *matching*. A parte *action* do *template* é representada pela associação *XSLTActions* entre as classes *XSLTTemplate* e *XSLTNode*. A parte *action* do *template* é composta por um conjunto de *nodes* que podem fazer parte do padrão XSLT ou não. Por exemplo, para criar a seguinte instrução *<xsl:text>create table </xsl:text>*, basta criar dois objetos: um instância de *XSLTElement*, representando o elemento *xsl:text*, e um outro, instância de *XSLTText*, representando o texto

create table. A Listagem 7.7 apresenta o código para a criação desses objetos no repositório XSLT.

Listagem 7.7 – Criando *nodes* XSLT no repositório

```
Xsltelement element = factoryelement.createXsltelement("xsl:text");
XsltnodeText text = factorynodetext.createXsltnodeText("create table ");
factoryelementcontains.add(text,element);
factoryactions.add(element,template);
```

Ao realizar o mapeamento do documento XSLT da Figura 7.7 para o metamodelo XSLT do repositório MOF, os metadados poderão ser visualizados, consultados e apagados utilizando o *browser* de objetos, apresentado da Figura 7.9.

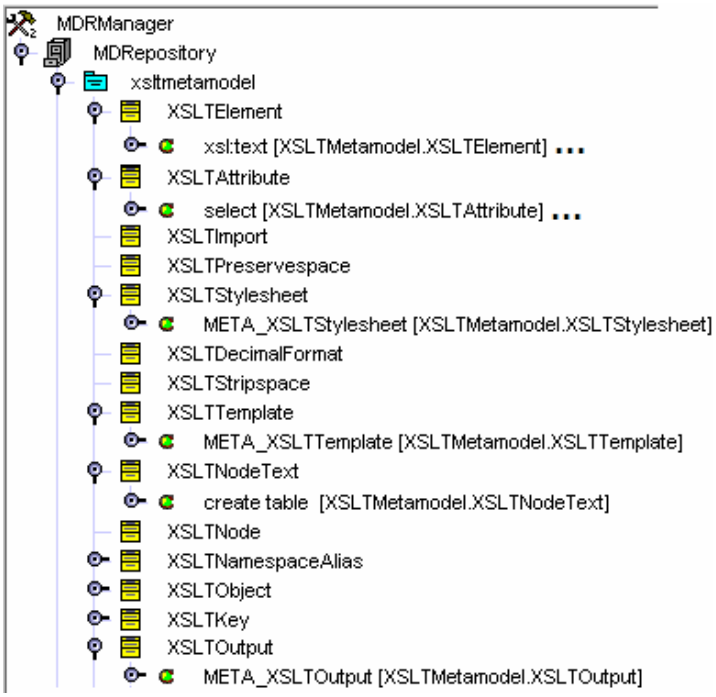

Figura 7.9 - Os metadados no repositório XSLT

A próxima seção apresenta um exemplo utilizando o metamodelo RDFS.

7.2 RDF Schema

Como foi visto no Capítulo 3, RDF Schema permite ao usuário criar seu próprio vocabulário RDF. Ele é descrito em termos de seus próprios conceitos. Desta maneira, é possível utilizar o próprio metamodelo RDFS para armazenar o próprio RDF Schema.

Utilizando as interfaces Java geradas através do mapeamento *MOF->Java* a partir do metamodelo RDFS é possível criar documentos RDF Schema. Os passos para a construção de um documento RDFS no repositório MOF são os seguintes:

- Iniciar o repositório MOF e pesquisar o sub-repositório RDFS, ou seja, o pacote *RDFSMetamodel*
- O pacote é representado pela classe *RdfsmetamodelPackage*, através dele é possível acessar todos os elementos do metamodelo, ou seja, as classes, as associações, os tipos enumerados, entre outros.

7.2.1 Criando o documento RDFS

Para a criação de um novo documento é necessário, dentre outros parâmetros, a URI do documento, a versão do mesmo e as datas de criação e alteração. Para isto, basta invocar o método da classe *proxy RdfsdocumentClass* que cria objetos instância da classe *Rdfsdocument* como apresentado na Listagem 7.8.

Listagem 7.8 – Criando um novo documento RDF Schema no repositório

```
Rdfsdocument docroot =
factorydoc.createRdfsdocument("http://www.cin.ufpe.br/~hls/rdfs.rdf",
      "Definicao do proprio RDFS", 1,
      "Definicao de RDF Schema em RDF Schema", "01/11/2002","01/11/2002");
```

Listagem 7.9 – Criando os objetos *namespaces* no repositório

```
Rdfsnamespace rdf = factorynamespace.createRdfsnamespace(
"http://www.w3.org/1999/02/22-rdf-syntax-ns#", "namespace de rdf","rdf");
Rdfsnamespace rdfs = factorynamespace.createRdfsnamespace(
      "http://www.w3.org/2000/01/rdf-schema#","namespace  de  rdf  schema",
      "rdfs");
factorydocnamespaces.add(rdf,docroot); //associa o namespace rdf ao doc
factorydocnamespaces.add(rdfs,docroot); //associa o namespace rdfs ao doc
```

Como foi visto no Capítulo 3, RDF Schema utiliza *namespace* para facilitar a identificação dos recursos descritos pelo documento. Desta maneira, é necessário criar os objetos *namespaces* do documento. O núcleo do vocabulário RDF Schema é definido no *namespace* conhecido informalmente como *rdfs* e identificado pela URI *http://www.w3.org/2000/01/rdf-schema#*. O núcleo de RDF é definido pelo *namespace rdf* e identificado pela URI *http://www.w3.org/1999/02/22-rdf-syntax-ns#*. Ambos os *namespaces rdfs* e *rdf* são utilizados no documento RDFS. Os objetos *namespace* do metamodelo RDFS são instâncias da classe

Rdfsnamespace. A Listagem 7.9 apresenta o código Java para a criação desses objetos e a associação ao objeto documento, criado anteriormente. Para a criação de um objeto *namespace* é necessário a URI do mesmo e o comentário que é herdado da classe mais genérica do metamodelo RDFS *Rdfsresource.*

7.2.2 Criando os Recursos Classes

Após criar os documentos e os *namespaces* a serem usados, o próximo passo é a criação dos recursos definidos pelo esquema. Esses recursos são classes e propriedades e as restrições (*constraints*) sobre o uso das classes e propriedades. O primeiro recurso a ser criado é a própria classe recurso. Associado ao recurso foi criado dois objetos *label* que representam a maneira como a classe é apresentada em uma determinada língua. É necessário também associar o recurso criado ao *namespace* e ao documento RDFS que o contém. Isto é feito respectivamente, através das associações *proxys RdfsisDefinedBy e Rdfscontains* do metamodelo RDFS.

Listagem 7.10 – Criando a classe *Resource* de RDF Schema no repositório

```
Rdfsclass resource = factoryclass.createRdfsclass("#Resource",
      "representa a classe recurso de RDF Schema");
factorylabelcontains.add(factorylabel.createRdfslabel("Resource","","en"),
      resource);
factorylabelcontains.add(factorylabel.createRdfslabel("Recurso","","pt"),
      resource);
factoryIsDef.add(rdfs,resource);//o recurso esta definido no namespace rdfs
factorycontains.add(resource,docroot)
```

A Listagem 7.10 apresenta o código Java para a criação da classe *Resource* de RDF Schema.

Além de *Resource*, outras duas classes importantes de RDF Schema são as classes *Class* e *Property* que representam, respectivamente, um recurso classe e propriedade. É necessário dizer que estas classes são subclasses da classe *Resource*. Isto foi realizado através da associação *proxy RdfssubClassOf* que diz que uma determinada classe é subclasse de outra. A Listagem 7.11 apresenta o código Java para a criação das classes *Class* e *Property* no repositório.

Além das classes criadas anteriormente, RDFS possui outras classes, algumas delas fazem parte da especificação RDF, ou seja, são definidas pelo *namespace rdf*, e outras fazem parte do próprio RDFS. A Listagem 7.12 apresenta o código Java para a criação das classes de RDF

Schema.

Listagem 7.11 – Criando as classes *Class* e *Property* de RDF Schema no repositório

```
Rdfsclass classe = factoryclass.createRdfsclass("#Class",
      "representa a classe class de RDF Schema");

factorylabelcontains.add(factorylabel.createRdfslabel("Class","","en"),
      classe);
factorylabelcontains.add(factorylabel.createRdfslabel("Classe","","pt"),
      classe);
factoryIsDef.add(rdfs,classe);//o recurso esta definido no namespace rdfs
factorycontains.add(classe,docroot); //adiciona o recurso ao documento
factorysubClassof.add(classe,resource);
Rdfsclass property = factoryclass.createRdfsclass("#Property",
      "representa a classe propriedade de RDF Schema");
factorylabelcontains.add(factorylabel.createRdfslabel("Property","","en"),
      property);
factorylabelcontains.add(factorylabel.createRdfslabel("Propriedade","","pt"
      ),property);
factoryIsDef.add(rdf, property); //o recurso esta definido no namespace rdf
factorycontains.add(property,docroot); //adiciona o recurso ao documento
factorysubClassof.add(property,resource);//a classe Property é subclasse da
                                         // classe Resource
```

Listagem 7.12 – Criando as outras classes de RDF Schema no repositório

```
Rdfsclass literal = factoryclass.createRdfsclass("#Literal",
      "representa a classe literal de RDF Schema");
factoryIsDef.add(rdfs, literal);//o recurso esta definido no namespace rdfs
Rdfsclass statement = factoryclass.createRdfsclass("#Statement",
      "representa a classe Statement de RDF");
factoryIsDef.add(rdf, statement);//o recurso esta definido no namespace rdf
Rdfsclass container = factoryclass.createRdfsclass("#Container",
      "representa a classe Container de RDF");
factoryIsDef.add(rdfs, container);
Rdfsclass bag = factoryclass.createRdfsclass("#Bag","classe Bag de RDF");
factoryIsDef.add(rdf, bag);//o recurso esta definido no namespace rdf
Rdfsclass seq = factoryclass.createRdfsclass("#Seq","classe Seq de RDF");
factoryIsDef.add(rdf, seq);//o recurso esta definido no namespace rdf
Rdfsclass alt = factoryclass.createRdfsclass("#Alt","classe Alt de RDF");
factoryIsDef.add(rdf, alt);//o recurso esta definido no namespace rdf
//cria a hierarquia de classes
factorysubClassof.add(literal,resource); //Literal é subclasse de Resource
factorysubClassof.add(statement,resource);
factorysubClassof.add(container,resource);
factorysubClassof.add(bag,container);//Bag é subclasse de Container
factorysubClassof.add(seq,container);
factorysubClassof.add(alt,container);
```

7.2.3 Criando os Recursos *Constraints*

O padrão RDF permite construir *statements* restringindo os valores de determinadas propriedades e classes. RDF Schema possui mecanismo que diz que um determinado recurso é uma *constraint*. São as classes *ConstraintResource* e *ConstraintProperty* e as propriedades

range e *domain*, conforme foi visto no Capítulo 3. A Listagem 7.13 apresenta o código Java para a criação das classes *constraints* de RDF Schema e a Listagem 7.14 apresenta o código para a criação das propriedades *range* e *domain*.

Listagem 7.13 – Criando as classes *constraints* de RDF Schema no repositório

```
Rdfsclass constraintresource =
      factoryclass.createRdfsclass("#ConstraintResource",
                  "representa a classe constraint resource de RDF Schema");
factorylabelcontains.add(factorylabel.createRdfslabel("ConstraintResource",
      "","en"),constraintresource);
factoryIsDef.add(rdfs, constraintresource);//definido no namespace rdfs
factorycontains.add(constraintresource,docroot);//adiciona ao documento
factorysubClassof.add(constraintresource,resource);
Rdfsclass constraintproperty =
      factoryclass.createRdfsclass("#ConstraintProperty",
      "representa a classe constraint property de RDF Schema");
factorylabelcontains.add(factorylabel.createRdfslabel("ConstraintProperty",
      "","en"),constraintproperty);
factoryIsDef.add(rdfs, constraintproperty);
factorycontains.add(constraintproperty,docroot);
factorysubClassof.add(constraintproperty,constraintresource);
```

Além de criar as propriedades e relacionar a qual documento e *namespace* pertencem, é necessário dizer que são instâncias da classe *ConstraintProperty* e definir algumas *constraints* sobre as mesmas. Algumas *constraints* foram definidas sobre as propriedades *rdfs:range* e *rdfs:domain* listadas a seguir:

- O *rdfs:domain* da propriedade *rdfs:range* é a classe *rdf:Property,* instância de *Rdfsclass* no metamodelo RDFS. Isto indica que a propriedade *rdfs:range* se aplica a recursos do tipo *rdf:Property*;
- O *rdfs:range* da propriedade *rdfs:range* é a classe *rdfs:Class*, instância de *Rdfsclass* no metamodelo RDFS.
- O *rdfs:domain* de *rdfs:domain* é a classe *rdf:Property,* instância da classe *Rdfsproperty* do metamodelo RDFS. Isto indica que a propriedade *rdfs:domain* pode possuir como valores, recursos do tipo *rdf:Property*;
- O *rdfs:range* da propriedade *rdfs:domain* é a classe *rdfs:Class*, instância da classe *Rdfsclass* no metamodelo RDFS.

As *constrainsts* de RDF Schema são criadas no metamodelo RDFS através das associações *proxys Rdfsrange* e *Rdfsdomain* do metamodelo RDFS, conforme apresentado na Listagem 7.15.

Listagem 7.14 – Criando as propriedades *range* e *domain* de RDF Schema no repositório

```
Rdfsproperty prange = factoryproperty.createRdfsproperty("#range",
      "representa a propriedade range de RDF Schema");
factorylabelcontains.add(factorylabel.createRdfslabel("range","","en"),
                                                              prange);
factoryIsDef.add(rdfs, prange); //o recurso esta definido no namespace rdfs
factorycontains.add(prange,docroot); //adiciona o recurso ao documento
Rdfsproperty pdomain = factoryproperty.createRdfsproperty("#domain",
      "representa a propriedade domain de RDF Schema");
factorylabelcontains.add(factorylabel.createRdfslabel("domain","","en"),
      pdomain);
factoryIsDef.add(rdfs, pdomain); //esta definido no namespace rdfs
factorycontains.add(pdomain,docroot);// //adiciona o recurso ao documento
```

Listagem 7.15 – Criando algumas *constraints* de RDF Schema no repositório

```
factorytype.add(prange,constraintproperty);//range é um ConstraintProperty
factorytype.add(pdomain,constraintproperty);//d é um ConstraintProperty
factorydomain.add(property,prange);//d de range é a classe rdf:Property
factoryrange.add(classe,prange);//r de range é a classe rdf:Class
factorydomain.add(property,pdomain);//d de domain é do tipo rdf:Property
factoryrange.add(classe,pdomain);//r de domain é do tipo rdf:Class
```

7.2.4 Criando os Recursos Propriedades

Além das classes definidas anteriormente, RDF Schema possui um conjunto de propriedades que definem os relacionamentos entre essas classes. Essas propriedades serão instâncias da classe *Rdfsproperty* do metamodelo RDFS.

Listagem 7.16 – Criando outras propriedades de RDF Schema no repositório

```
Rdfsproperty ptype = factoryproperty.createRdfsproperty("#type”,"");
Rdfsproperty psubclassof =
     factoryproperty.createRdfsproperty("#subClassOf","");
Rdfsproperty psubpropertyof =
     factoryproperty.createRdfsproperty("#subPropertyOf",
      "representa a propriedade subPropertyOf de RDF Schema");
Rdfsproperty pcomment =
      factoryproperty.createRdfsproperty("#comment",”comment de RDFS");
Rdfsproperty plabel =
      factoryproperty.createRdfsproperty("#label",
      "representa a propriedade label de RDFS");
Rdfsproperty pseealso =
      factoryproperty.createRdfsproperty("#seeAlso",
      "seeAlso de RDF Schema");
Rdfsproperty pisdefinedby =
      factoryproperty.createRdfsproperty("#isDefinedBy",
      ”isDefinedBy de RDFS");
Rdfsproperty psubject =
      factoryproperty.createRdfsproperty("#subject","subject de RDF");
          Rdfsproperty ppredicate =
      factoryproperty.createRdfsproperty("#predicate",
      "representa o predicate de RDF");
Rdfsproperty pobject =
      factoryproperty.createRdfsproperty("#object",
      "representa o object de RDF");
```

A Listagem 7.16 apresenta o código Java para a criação dessas propriedades no repositório MOF. Algumas dessas propriedades são definidas pelo padrão RDF, outras são definidas pelo próprio RDF Schema, mas todas são usadas no documento RDF Schema.

7.2.5 Criando *Constraints* sobre as Classes e Propriedades de RDFS

Como foram mostradas no Capítulo 3, as *constraints* de RDFS servem para restringir o uso das propriedades, inclusive as próprias propriedades de RDF Schema.

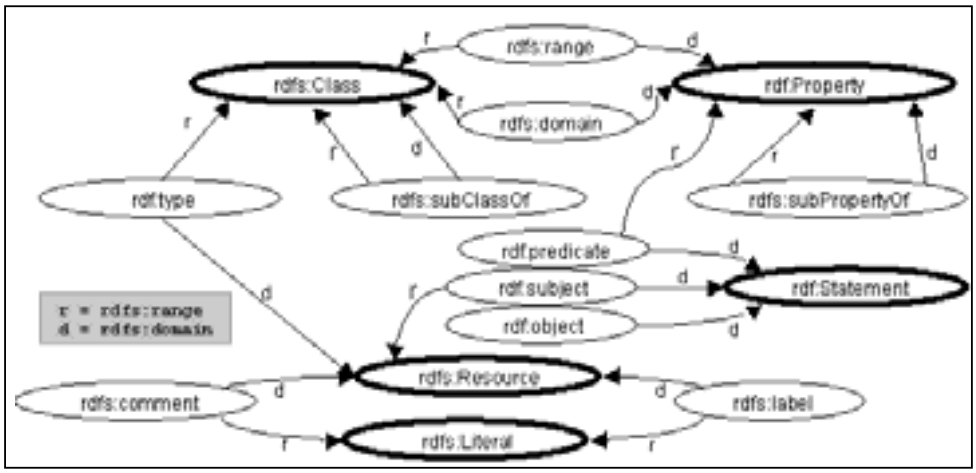

Figura 7.10 – O modelo de *Constraints* de RDF Schema [BRGU2000]

A Figura 7.10 apresenta o modelo de *constraints* aplicados às propriedades de RDF Schema. A Listagem 7.17 apresenta o código Java para a criação dessas *constrainsts* no repositório MOF através do metamodelo RDFS. As *constraints* relacionadas às propriedades *rdfs:range* e *rdfs:domain* já foram apresentadas na seção anterior.

Listagem 7.17 – Criando o modelo de *constraints* de RDF Schema no repositório

```
factoryrange.add(classe,ptype);//range da prop type é rdfs:Class
factorydomain.add(resource,ptype);//dominio de type é do tipo rdfs:Resource
factoryrange.add(classe,psubclassof);//range de subClassOf é rdfs:Class
factorydomain.add(classe,psubclassof);//dominio de subClassOf é rdfs:Class
factoryrange.add(property,psubpropertyof);//r de subPropertyOf é Property
factorydomain.add(property,psubpropertyof);//d de subPropOf é Property
factoryrange.add(literal,pcomment);//r de comment é um literal
factorydomain.add(resource,pcomment);//d de comment é rdfs:Resource
factoryrange.add(literal,plabel);//r da propriedade label é rdfs:Literal
factorydomain.add(resource,plabel);//d de label é rdfs:Resource
factoryrange.add(resource,psubject);//r de subject é rdfs:Resource
factorydomain.add(statement,psubject);//d de subject é rdfs:Statement
factoryrange.add(property,ppredicate);//r de predicate e rdfs:Property
factorydomain.add(statement,ppredicate);//d de subject é rdfs:Statement
factorydomain.add(statement,pobject);//d de object é rdfs:Statement
```

Após a criação do documento no repositório RDFS, é possível visualizar os objetos no *browser* MDR como apresentado na Figura 7.11.

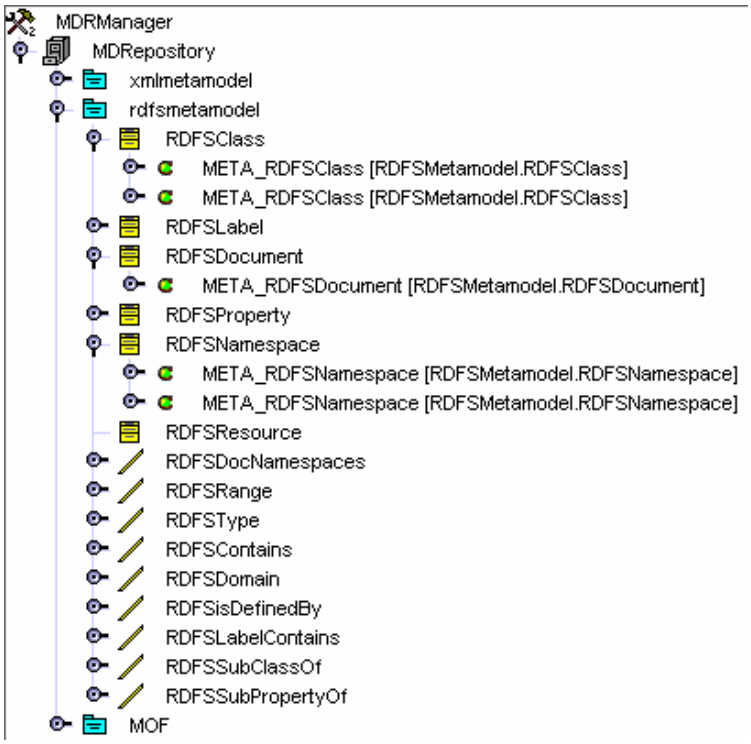

Figura 7.11 - Os metadados no repositório RDFS

Alguns padrões de metadados como Dublin Core [BMB2002] possuem uma representação em RDF Schema. O Dublin Core define um conjunto de termos que são utilizados como metadados em documentos da Web. Assim, os documentos RDF podem descrever os recursos como páginas Web, utilizando os termos definidos do Dublin Core através de RDF Schema.

7.3 Considerações Finais

Este capítulo apresentou um estudo de caso utilizando os metamodelos construídos nos Capítulos 5 e 6 e implementados pela ferramenta MDR. Foi apresentado como as interfaces dos metamodelos construídos podem ser utilizadas para gerenciar metadados em um repositório MOF.

Qualquer ferramenta que implemente o MOF poderia importar os metamodelos e gerar um conjunto de interfaces para gerenciamento dos metadados. Poderia importar e exportar estes

metadados utilizando o padrão XMI, ou usar os utilitários que mapeiam um padrão específico para o seu metamodelo. Por exemplo, poderia importar os metadados de um documento XML utilizando o utilitário que carrega um documento XML para o repositório MOF.

Foram apresentados dois estudos de casos: o primeiro, baseado em metadados gerados pela metodologia *Fast Cube* do ambiente REDIRIS, e o segundo, mostrou como construir um documento RDF Schema no repositório. O próximo capítulo apresenta as conclusões do trabalho.

8 Conclusões

8.1 Considerações Finais

Este trabalho apresentou um protótipo de uma solução de metadados adequada para o ambiente REDIRIS ou outros ambientes. Para isto, foi realizado um estudo abrangente sobre o tema metadados e as suas aplicações, bem como sobre os padrões utilizados na modelagem, descrição e intercâmbio de metadados.

Foi constatado que os padrões do W3C como XML, DTD, RDF e RDF Schema são padrões independentes de plataforma, semi-estruturados, suportados por uma variedade de ferramentas e adequados para representar metadados das mais diversas fontes. Foi proposto um conjunto de metamodelos MOF construídos a partir de padrões do W3C. Estes metamodelos fazem parte da solução de metadados proposta para o ambiente REDIRIS.

Foi gerado um conjunto de documentos XMI a partir destes metamodelos. Qualquer repositório MOF poderá importar estes documentos e passará a ter suporte ao gerenciamento dos metadados descritos por estes metamodelos. Além dos metamodelos, os metadados que são instâncias desses metamodelos também poderão ser exportados e importados por qualquer repositório MOF. Este trabalho utilizou a ferramenta MDR com JMI.

Os metamodelos foram construídos segundo as especificações de seus correspondentes padrões. Eles não foram construídos apenas com o objetivo de gerenciar os metadados, mas também de preservar o mapeamento entre os objetos construídos em MOF e os conceitos de suas especificações. Por exemplo, no padrão XSLT existe o conceito de *template*, então foi construído uma classe *XSLTTemplate* no metamodelo XSLT que representa este conceito. Nos padrões RDF e RDF Schema existem os conceitos de recurso, de maneira semelhante,

foram construídas classes que representam este conceito em seus respectivos padrões.

As *constraints* OCL foram utilizadas apenas para realizar validação dos objetos dentro de um mesmo metamodelo. Outras *constraints* poderiam ser definidas para fazer a validação entre objetos de metamodelos diferentes. Por exemplo, entre um objeto do tipo *XMLDocumentType* do metamodelo XML e um objeto do tipo *DTDDocument* do metamodelo DTD. Durante a criação ou alteração de um objeto do tipo *XMLDocumentType*, poderia existir algum tipo de validação que verificaria se a DTD que o documento XML referencia existe no repositório DTD. De maneira semelhante, poderiam ser utilizados entre os repositórios XML e XSLT, RDF e RDF Schema, ou qualquer outra validação entre objetos de repositórios diferentes.

Foi apresentado um estudo de caso com dois exemplos, ilustrando como o repositório poderá ser utilizado por outras ferramentas para o gerenciamento de metadados descritos pelos padrões modelados. O primeiro exemplo apresentado utiliza os metamodelos XML, DTD e XSLT para gerenciamento dos metadados gerados pela metodologia *Fast Cube*. O segundo utilizou o metamodelo RDFS para criar o próprio documento RDF Schema.

8.2 Principais Contribuições

Como principais contribuições deste trabalho destacamos:

- O projeto e a implementação de um metamodelo que representa o padrão XML em MOF. Através deste metamolo, os usuários poderão gerenciar metadados, descritos no padrão XML, em repositórios MOF;
- O projeto e a implementação de um metamodelo que representa o padrão DTD em MOF. Com este metamodelo, os usuários poderão gerenciar documentos DTD em repositórios MOF;
- O projeto e a implementação de um metamodelo que representa o padrão XSLT em MOF. Através dele, os usuários poderão gerenciar documentos XSLT em repostórios MOF;
- O projeto e a implementação de um metamodelo que representa o padrão RDF em MOF. O metamodelo RDF permite gerenciar metadados, descritos no padrão RDF, em repositórios MOF;
- O projeto e a implementação de um metamodelo que representa o padrão RDF

Schema em MOF;

- A criação de um repositório de dados e metadados genérico para o ambiente REDIRIS. Devido à flexibilidade dos padrões XML, é possível armazenar tanto dados quanto metadados no repositório MOF e este repositório pode ser utilizado por diversas aplicações que desejem representar seus dados e metadados em XML e utilizar interfaces padrões para intercâmbio e gerenciamento dos mesmos;
- A definição de um conjunto de interfaces comuns para acesso a documentos XML, DTD, XSLT, RDF e RDF Schema. Atualmente, cada padrão possui a sua própria interface, algumas são definidas pelos próprios desenvolvedores das ferramentas que suportam estes padrões;
- Uma alternativa comum para intercâmbio e representação dos dados e metadados que estão representados nos diversos padrões. Cada padrão possui uma forma de representação dos seus metadados. Alguns possuem a sintaxe XML outros não. A modelagem desses padrões em MOF permite que todos os dados e metadados sejam representados em XMI.

8.3 Trabalhos Futuros

Os principais trabalhos futuros que poderão ser desenvolvidos a partir deste são:

- Implementar os utilitários que fazem o mapeamento entre os padrões RDF Schema e XSLT e os seus respectivos metamodelos. Estes utilitários são importantes, pois oferecem uma forma a mais de intercâmbio dos metadados, além do XMI. Às vezes, o usuário necessita exportar e importar os metadados para os seus padrões originais.
- Implementar o suporte a *Namespace*, *XLink*, *XPointer e XPath*. Alguns metamodelos como o RDF e RDF Schema possuem suporte a *namespace*, porém não foi implementado nenhum mecanismo de verificação se tais termos utilizados em um determinado documento pertencem ao conjunto de termos definidos pelo *namespace*. Isto poderia ser realizado através de *constraints* OCL. A implementação de XLink e XPointer seriam importantes para permitir relacionamentos entre elementos do mesmo documento ou de documentos diferentes. Isto em MOF significa permitir relacionamentos entre objetos de um mesmo metamodelo ou de metamodelos diferentes. A especificação MOF 1.3 utilizada pela ferramenta MDR não suporta

relacionamentos entre objetos que pertencem a pacotes diferentes, ou que não possuam um *superpacote* em comum. O MOF 2.0 prevê isto, porém ainda está em desenvolvimento;

- Integrar os metamodelos atuais. Algumas classes e associações poderiam ser utilizadas em mais de um metamodelo. Isto requer a criação de pacotes comuns que seriam compartilhados pelos metamodelos. Isto não foi feito durante o trabalho, pois os metamodelos foram modelados e implementados um de cada vez. Por exemplo, as classes *XMLDocument*, *DTDDocument*, *RDFDocument*, *RDFSDocument* poderiam possuir uma superclasse comum.
- Implementação de outros padrões como XML Schema e XQuery. O padrão XML Schema ofereceria uma alternativa para a validação dos documentos XML. XQuery ofereceria uma linguagem de consulta e integração de documentos XML. No repositório atual, os documentos XML Schema e XQuery podem ser armazenados como documentos XML, pois ambos possuem a sintaxe XML.
- Realização do processo de validação dentro do próprio repositório, através de constraints OCL. Atualmente, este processo é realizado fora do repositório, através de um *parser* XML. Ou seja, um documento XML para ser validado por um documento DTD, ambos armazenados no repositório MOF, os mesmos devem ser exportados para as suas representações originais (XML e DTD) para que possam ser submetidos a um parser XML para o processo de validação. Uma forma mais econômica seria fazer a validação dentro do próprio repositório MOF. Por exemplo, quando o usuário for construir um novo documento XML no repositório, ele poderia dizer qual documento DTD valida tal documento XML. À medida que os objetos referentes ao novo documento XML vão sendo criados, o próprio repositório MOF poderia ir verificando se estes objetos estão de acordo com a estrutura definida pela DTD. De forma semelhante, poderia ser implementado para XML Schema. A vantagem de utilizar OCL como meio de validação é que as mesmas regras OCL podem ser transportadas juntamente com os metamodelos e podem ser suportadas por qualquer ferramenta que implemente a especificação MOF com OCL.
- Realização do processo de transformação dentro do próprio repositório. De maneira similar ao processo de validação, atualmente, as transformações XSLT devem ser executadas fora do repositório através de um *parser* XSLT. A realização das

transformações dentro do repositório seria mais econômica.

- Submissão desses metamodelos ao W3C e à OMG para que sejam efetivados como padrões similar o CWM.

9 REFERÊNCIAS BIBLIOGRÁFICAS

[ABK2000] R. Anderson, M. Birbeck, M. Kay, S. Livingstone, B. Loesgen, D. Martin, A. Mohr, N. Ozu, B. Peat, J. Pinnock, P. Stark, K. Williams. "Professional XML". 2000. Wrox Press.

[ABS2000] ABITEBOUL, Serge; BUNEMAN, Peter; SUCIU, Dan. "Gerenciando dados na Web". 2000. São Paulo. Editora Campus.

[AHAY2001] AHMED, Kal; AYERS, Danny; et al. "Professional XML Metadata". 2001. UK. Wrox Press Ltda.

[BAR1999] BARRETO, Cássia. "Modelo de Metadados para a Descrição de Documentos Eletrônicos na Web". Tese de mestrado – UFRJ. http://genesis.nce.ufrj.br/dataware/. Agosto, 1999.

[BCF2002] BOAG, Scott; CHAMBERLIN, Don; FERNANDEZ, F, Mary. "XQuery 1.0: An XML Query Language – W3C Working Draft". http://www.w3.org/XML/Query /. November, 2002.

[BCS1999] BATORY, Don; CARDONEI, Rich; SMARAGDAKIS, Yannis. "Object-Oriented Frameworks and Product-Lines". 1st Software Product-Line Conference, Denver, Colorado. August, 1999.

[BMB2002] BECKETT, Dave; MILLER, Eric; BRICKLEY, Dan. "Expressing Simple Dublin Core in RDF/XML". http://dublincore.org/documents/2002/07/31/dcmes-xml/. July, 2002.

[BOU2000] BOURRET, R. P. "XML and Databases". http://www.rpbourret.com/xml/XMLAndDatabases.htm. 2000.

[BOU2001] BOURRET, R. P. "XML Databases Products"

http://www.rpbourret.com/xml/XMLDatabaseProds.htm. 2001.

[BRAY1998] BRAY, Tim. "RDF and Metadata". http://www.xml.com/pub/a/98/06/rdf.html. June, 1998.

[BRGU2000] BRICKLEY, Dan; GUHA, R.V. "Resource Description Framework (RDF) Schema Specification 1.0 - W3C Candidate Recommendation". http://www.w3.org/TR/rdf-schema. March, 2000.

[BTH1999] BRAY, Tim; HOLLANDER, Dave; LAYMAN, Andrew. "Namespaces in XML". http://www.w3.org/TR/1999/REC-xml-names-19990114/. January, 1999.

[CBNW1997] CHAMPION, Mike; BYRNE, Steve; NICOL, Gavin; WOOD, Lauren. "Document Object Model Level 1 – W3C Recommendation", November, 1997.

[CHAN2000] CHANG, Daniel, T. "CWM Enablement Showcase: Warehouse Metadata Interchange Made Easy Using CWM". http://www.cwmforum.org/paperpresent.htm. December, 2000.

[CHDA1997] CHAUDHURI, S; DAYAL, U. "An Overview of Data Warehousing and Olap Tecnology", SIGMOD Record, New York, v.26, nº 1, pg.65-74, Março 1997.

[CHEN2001] CHEN, Zhengxin. "Intelligent Data Warehousing: from Data Preparation to Data Mining". 2001. CRC Press, USA.

[CLA1999] CLARK, J. "XSL Transformations (XSLT) Version 1.0 – W3C" http://www.w3.org/TR/1999/REC-xslt-19991116. November, 1999.

[CWM2001] Common Warehouse Metamodel Specification, Volumes 1 & 2. http://www.omg.org/. Veja também em http://www.cwmforum.org/. February, 2001.

[DAN2002] DANNER, M. "XML Extender for DB2 Version 7". http://service2.boulder.ibm.com/devtools/news0800/art19.htm. 2002.

[DEMA2002] DEDIC, Svata; MATULA, Martin. "Metamodel for the Java language". Disponível em: http://java.netbeans.org/models/java/java-model.html. 2002.

[DMOF2001] "DMof - An OMG Meta Object Facility Implementation" http://www.dstc.edu.au/Products/CORBA/MOF/. June, 2001.

[FALL2002] FALLSIDE, C., David. "XML Schema – W3C Recommendation". http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/. May, 2001.

[GON1999] GONÇALVES, Andrea. "METASIG: Ambiente de Metadados para Aplicações de Sistemas de Informações Geográficos". Tese de mestrado – Instituto Militar de Engenharia. http://genesis.nce.ufrj.br/dataware/. Agosto, 1999.

[HARA2000] HAI, Hong; RAHM, Erhard. "On Metadata Interoperability in Data Warehouses". Report Nr 01; Department of Computer Science, University of Leipzig. March, 2000.

[HIG2001] HIGGINS, S. "Oracle 9i Application Developer's Guide – XML". http://download-east.oracle.com/otndoc/oracle9i/901_doc/appdev.901/a88894/title.htm. 2001.

[HJE2001] HJELM, Johan. "Creating the Semantic Web with RDF". 2001. New York. John Willey & Sons, Inc.

[INM2000] INMON, W. H. "Metadata in the Data warehouse". Disponível em: http://63.170.41.42/library/whiteprs/earlywp/ttmeta.pdf. 2000.

[IWG1999] INMON, W. H; WELCH, J. D; GLASSEY, Katherine. "Gerenciando Data Warehouse". 1999. Makron Books.

[JMI2002] Java Metadata Interface, JSR-40 Home Page: http://java.sun.com/aboutJava/communityprocess/jsr/jsr_040_jolap.html. March, 2002.

[KAY2002] KAY, Michael. "XSL Transformations (XSLT) Version 2.0 – W3C Working Draft". http://www.w3.org/TR/2002/WD-xslt20-20020816/. August, 2002.

[KIM1998] KIMBALL, Ralph, REEVES, Laura, ROSS, Margy. "The Data Warehouse LifeCycle Toolkit". 1998. New York-USA: John Willey & Sons, Inc.

[LASW1999] LASSILA, Ora; SWICK, Ralph R. "Resource Description Framework (RDF) Model and Syntax Specification". http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/. February, 1999.

[MAIN2000] MARCO, David; INMON, W. H. "Building and Managing the Metadata Repository". 2000. New York. John Wiley & Sons, Inc.

[MAR2001] MARINO, Maria Teresa. "Integração de Informações em Ambientes Científicos na Web: Uma abordagem baseada na Arquitetura RDF". Tese de mestrado – UFRJ. http://genesis.nce.ufrj.br/dataware/. Abril, 2001.

[MDA2001] OMG Architecture Board MDA Drafting Team, "Model-Driven Architecture: A Technical Perspective", ftp://ftp.omg.org/pub/docs/ab/01-

02-01.pdf. February, 2001.

[MDA2002] OMG Model-Driven Architecture Home Page: disponível em http://www.omg.org/mda/index.htm. March, 2002.

[MDR2002] Sun Microsystems. "Metadata Repository Home" http://mdr.netbeans.org/. 2002.

[MEGG2001] MEGGINSON, David et al. "SAX 1.0: The Simple API for XML". Disponível em: http://www.saxproject.org/. 2001.

[MOF1999] OMG Meta Object Facility Specification, Version 1.3. http://www.dstc.edu.au/Research/Projects/MOF/rtf/. http://www.omg.org/. September, 1999.

[OBJ2001] ObjectStore. http://www.objectdesign.com/htm/object_prod.asp. 2001.

[PCTM2001] POOLE,John; CHANG,Dan; TOLBERT, Douglas; MELLOR, David. "Common Warehouse Metamodel: An Introduction to the Standard for Data Warehouse Integration". 2001. New York. John Willey & Sons, Inc.

[POOL2001] POOLE, John. "Model-Driven Architecture: Vision, Standards And Emerging Technologies", Workshop on Metamodeling and Adaptive Object Models, April, 2001.

[PYL1999] PYLE, Dorian. "Data Preparation for Data Mining". 1999. Morgan Kaufmann Publishers Inc., San Francisco.

[SAN2002] SANTOS, Roberto A. F. "Metodologia e uso de técnicas de exploração e análise de dados na construção de data warehouse". Dissertação de mestrado UFPE. Setembro, 2002.

[SBB2002] SANTOS, Hélio L.; BATISTA, Maria da C. M.; BARROS, Roberto, S. M. "Publishing Theses and Dissertations: An Approach using XML". iiWAS2002.

[SIL2000] SILVA, Luís Alexandre. "Geração Dinâmica de Interfaces de Bibliotecas Digitais Baseadas em Metadados". Tese de mestrado – Instituto Militar de Engenharia. http://genesis.nce.ufrj.br/dataware/. Julho, 2000.

[SMVV2000] STAUDT, Martin; VADUVA, Anca; VETTERLI, Thomas. "The Role of Metadata for Data Warehousing". University of Zurich, Departament of Computer Science. 2000

[TAM1999] TAMINO. "Documentation Overview". Disponível em http://www.cs.uni-essen.de/dawis/teaching/ss2000/nsdb/tamino/help/overview.htm. 1999.

[TANN2002] TANNENBAUM, Adrienne. "Metadata Solutions: Using Metamodels, Repositories, XML and Enterprise Portals to Generate Information on Demand". 2002. New York. Addison Wesley.

[TOLB2000] TOLBERT, Doug. "CWM: A Model-based Architecture For Data Warehouse Interchange", Workshop on Evaluating Software Architectural Solutions, University of California, Irvine. http://www.isr.uci.edu/events/wesas2000/. May, 2000.

[UML2001] OMG Unified Modeling Language Specification, Version 1.4. http://cgi.omg.org/docs/formal/01-09-67.pdf. September, 2001.

[VASS2000] VASSILLIADIS, P. "Gulliver in the Land of Data Warehouse : Practical Experiences and Observations of a Researcher". Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW2000), Estocolmo, Suécia, June, 2000.

[VDO2001] VDOVJAK, R, HOUVEN, G-J. "RDF Based Architecture for Semantic Integration of Heterogeneous Information Sources". Proceedings of the International Workshop on Information Integration on the Web (WIIW'2001), pg51-57, Rio de Janeiro, Brazil, April, 2001.

[XLP2000] W3C; W3C XML Pointer, XML Base and XML Linking, http://www.w3.org/XML/Linking. December, 2000.

[XMI2000] Object Management Group, XML Metadata Interchange Specification, Version 1.1, http://www.omg.org/. June, 2000.

[XPA1999] W3C; XML Path Language (XPath); http://www.w3.org/TR/xpath. November, 1999.

[W3C2002] "World Wide Web Consortiun". www.w3c.org, visitado em 22/02/2002.

[WURI2000] W3C. "Uniform Resource Identifier (URI) Activity Statement". Disponível em: http://www.w3.org/Addressing/Activity. July, 2000.