UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA e COMPUTAÇÃO

Um Compilador para um Sistema de Gerenciamento de Metadados baseado em MOF

Autor: Adriana Maria Cunha de Melo Figueiredo

Orientador: Prof. Manuel de Jesus Mendes, Dr. Ing.

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: **Engenharia de Computação**.

Banca Examinadora

Prof. Dr. Eleri Cardozo	DCA/FEEC/Unicamp
Prof. Dr. Ivan Luiz Marques Ricarte	DCA/FEEC/Unicamp
Prof. Dr. Adalberto Nobiato Crespo	CenPRA

Campinas, SP

Novembro/2004

FICHA CATALOGRÁFICA ELABORADA PELA

BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

F469c

Figueiredo, Adriana Maria Cunha de Melo

Um compilador para um sistema de gerenciamento de metadados baseado em MOF / Adriana Maria Cunha de Melo Figueiredo. -- Campinas, SP: [s.n.], 2004.

Orientador: Manuel de Jesus Mendes.

Dissertação (mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Modelos e construção de modelos. 2. Compiladores (Programas de computador). 3. Metadados. 4. Engenharia de software. I. Mendes, Manuel de Jesus. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

RMS - BAE

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato(a): Adriana Maria Cunha de Melo Figueiredo

Data da Defesa: 16 de novembro de 2004
Título da Tese: "Um Compilador para um Sistema de Gerenciamento de Metadados Baseado em MOF"

Prof. Dr. Manuel de Jesus Mendes (Presidente): _____

Prof. Dr. Adalberto Nobiato Crespo: _____

Prof. Dr. Eleri Cardozo: _____

Prof. Dr. Ivan Luiz Marques Ricarte: _____

Resumo

Metadado é elemento fundamental no suporte a integração e interoperabilidade de sistemas de computação distribuídos. Uma ampla interoperação entre sistemas heterogêneos e distribuídos é alcançada através do compartilhamento de metadados descrevendo a semântica dos sistemas e suas capacidades. Metamodelo é a técnica que provê um maior nível de abstração para gerenciar a heterogeneidade no nível de metadados. Diversos metamodelos podem existir em um ambiente distribuído e a interoperabilidade neste caso só é possível quando o sistema de gerenciamento de metadados suporta esta diversidade. *Meta-Object Facility* (MOF) é um padrão aberto para a definição e manipulação de metadados e metamodelos.

Neste trabalho é proposto um sistema de repositório para manipular metadados e metamodelos baseados na tecnologia MOF e, especificamente, é descrito um compilador da linguagem *Meta-Object Definition Language* para a descrição de metamodelos MOF.

O resultado deste trabalho será aplicado no contexto de governo eletrônico, um ambiente heterogêneo, complexo, multi-organizacional e que caracteriza-se pela forte presença de soluções departamentalizadas e individualizadas, combinando vários sistemas operacionais, linguagens de programação, tecnologias de rede e bancos de dados.

Palavras Chave: metadado, metamodelo, arquitetura de metadados, *Meta-Object Facility*

Abstract

Metadata is critical to support distributed systems integration and interoperability. A wide interoperation among heterogeneous and distributed systems is possible through the sharing of metadata describing the definitions of system semantics and capabilities. Metamodel is the mechanism that provides a higher level of abstraction to manage heterogeneity at the metadata level. Different metamodels can exist in a distributed environment and interoperability in this context is possible if the metadata management system supports this diversity. Meta-Object Facility (MOF) specification is an open standard with facilities to define and manipulate metadata and metamodels.

This work proposes a repository system to define and manipulate metadata and metamodel based on the MOF technology, specifically, it describes a compiler to MODL, a textual language to describe MOF metamodels.

The outcome of this work will be applied in the electronic government context, a heterogeneous and complex environment comprised of multiple organizations each one with its own solution to the government domain, combining different operating systems, programming languages, and databases.

Key Words: metadata, metamodel, metadata architecture, Meta-Object Facility

O homem nasceu para aprender, aprender

enquanto o tempo lhe permita

João Guimarães Rosa

Ao meu marido Clênio.

Aos meus filhos Marcos e Daniel.

AGRADECIMENTOS

A Deus, por cada minuto de graça concedida para que eu chegasse até aqui.

À Nicota e Antenor, mãe e saudoso pai, pelo carinho e pela oportunidade de estar aqui.

Ao meu marido Clênio, pelo amor, compreensão e carinho dispensados enquanto me dediquei a este trabalho.

Aos meus filhos, Marcos e Daniel pela compreensão durante os eternos momentos de minha ausência.

Em especial, agradeço ao meu orientador Prof. Manuel de Jesus Mendes, por ter me aceito no programa de mestrado, pelo direcionamento proporcionado e pelo exemplo constante na busca do conhecimento

Ao Centro de Pesquisas Renato Archer (CenPRA), pelo tempo disponibilizado para o desenvolvimento deste trabalho e estímulo ao aperfeiçoamento de seu corpo técnico.

Aos meus colegas de trabalho da DSSD/CenPRA, com quem pude contar em todos os momentos.

A Unicamp por me aceitar como aluna e pelos valiosos ensinamentos ministrados.

Aos Professores Eleri Cardozo e Ivan Ricarte da Unicamp e Adalberto Crespo do CenPRA, pela gentileza de aceitarem o convite para participar da avaliação deste trabalho.

Aos meus amigos de toda a vida, que, com sua presença, pequenos gestos ou palavras, contribuíram para a minha chegada em mais esta etapa.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

API	Application Programming Interface
CDIF	Case Data Interchange Format
Corba	Common Object Request Broker Architecture
CWM	Common Warehouse Metamodel
EDOC	Enterprise Distributed Object Computing
GRM	Gerenciador de um Repositório MOF
HUTN	Human Usable Textual Notation
IDL	Interface Definition Language
JMI	Java Metadata Interface
MDA	Model-Driven Architecture
MODL	Meta-Object Definition Language
MOF	Meta-Object Facility
OCL	Object Constraint Language
OIM	Open Information Model
OMG	Object Management Group
PIM	Platform Independent Model
PSM	Platform Specific Model
UML	Unified Modelling Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
WSDL	Web Service Description Language

1 INTRODUÇÃO

1.1 Caracterização do Problema

Metadado significa dado que descreve dado e, em um primeiro instante, é associado aos esquemas de banco de dados. Todavia, a distinção entre dado e metadado tem modificado bastante e, segundo [Frankel 2003], o termo metadado deve incluir também, citando apenas alguns exemplos: (i) modelos de classes; (ii) modelos de *workflow*; (iii) regras de transformação de dados; (iv) APIs expressas em Corba IDL, Java, WSDL, etc.

No contexto de sistemas de computação distribuídos, metadado é elemento fundamental no suporte à integração e interoperabilidade. APIs padronizadas têm sido o mecanismo mais utilizado para interoperabilidade entre estes sistemas. No entanto, uma ampla interoperação entre sistemas heterogêneos e distribuídos somente é alcançada através do compartilhamento de metadados descrevendo a semântica dos sistemas e suas capacidades [Poole 2001].

Enquanto metadados provêem abstrações para organizar e gerenciar dados do mundo real, metamodelos, ou seja, modelos que descrevem modelos[1] , provêem um maior nível de abstração para gerenciar a heterogeneidade do nível de metadados [Tan 2002].

Metamodelo é um mecanismo poderoso para lidar com a heterogeneidade de ambientes distribuídos, provendo sintaxe e semântica comum para que componentes destes ambientes manipulem e entendam os modelos e, consequentemente os dados, de outros componentes [Kobryn 1999]. Diversos metamodelos podem existir em um ambiente distribuído e a interoperabilidade neste caso só é possível quando há suporte para esta diversidade. Uma arquitetura de metadados atende a este requisito, provendo um meta-metamodelo que permite a descrição e interpretação de diferentes metamodelos.

Apesar da importância dos metadados na construção de sistemas de informação avançados, sua gerência é, geralmente, deficiente. Dentre as dificuldades enfrentadas cita-se [Staudt 1999, Auth 2002]: (i) soluções proprietárias; (ii) repositório centralizado; (iii) semântica do metadado embutida no software; e (iv) ausência de um mecanismo eficaz para interoperabilidade.

[1] Nesta dissertação os termos metadados e modelo são intercambiáveis

Uma arquitetura de metadados é uma infra-estrutura que suporta diferentes padrões de metadados e a definição de novos padrões quando necessário. *Meta-Object Facility* (MOF), uma especificação do consórcio *Object Management Group* (OMG), adota uma arquitetura de metadados multi-camadas e oferece uma abordagem genérica para a representação e o gerenciamento de metadados em ambientes distribuídos. Para tanto, MOF define uma linguagem para definição de metamodelos e um *framework* genérico para o gerenciamento de metadados com interfaces Corba IDL padronizadas que possibilitam o acesso homogêneo aos metadados. *XML Metadata Interchange* (XMI), também do consórcio OMG, complementa a especificação MOF, provendo um mecanismo para o intercâmbio de metadados. XMI define um padrão para a representação de metadados descritos na linguagem MOF em XML. A arquitetura de metadados adotada pelo consórcio OMG, alcançou a comunidade Java quando esta, em meados de 2002, ratificou a especificação *Java Metadata Interface* (JMI) que mapeia para a plataforma Java o *framework* genérico definido por MOF.

1.2 Motivação

A Divisão de Software para Sistemas Distribuídos (DSSD) do Centro de Pesquisas Renato Archer (CenPRA) tem dedicado esforços para criar um ambiente colaborativo para serviços de governo eletrônico tendo como base uma plataforma aberta de serviços e metamodelos padronizados como mecanismo de integração e interoperabilidade.

O uso das tecnologias MOF, JMI e XMI é considerado estratégico para alcançar este objetivo uma vez que elas definem um padrão aberto para a representação de metadados e facilitam o desenvolvimento de componentes de software que requeiram o gerenciamento e intercâmbio de metadados. Neste contexto, foi especificado o sistema GRM, integrando as tecnologias MOF, JMI e XMI.

1.3 Objetivos do Trabalho

O objetivo principal desta dissertação é o desenvolvimento de um sistema de gerenciamento de metadados baseado nas tecnologias MOF, JMI e XMI. Como objetivos específicos tem-se:

1. a especificação deste sistema;
2. desenvolvimento de uma ferramenta para compilar metamodelos MOF descritos na linguagem *Meta-Object Description Language* (MODL).
3. demonstrar a importância de MOF no contexto de uma plataforma de governo eletrônico.

A relevância deste trabalho de pesquisa encontra-se em sua contribuição para o entendimento das especificaçãoes MOF e JMI e na especificação e desenvolvimento de um sistema de repositório que suporte estas tecnologias.

1.4 Organização da Dissertação

O restante desta dissertação está organizado da seguinte forma:

O Capítulo 2 aborda os principais conceitos ligados a área de metadados, especificamente para atender ao contexto deste trabalho. *Model-Driven Architecture*, uma nova abordagem para o desenvolvimento de software e que tem MOF como uma de suas tecnologias básicas, é brevemente descrita neste capítulo.

No Capítulo 3 é apresentada com detalhes a especificação MOF e o seu relacionamento com as especificações XMI e JMI. As novas funcionalidades de MOF 2.0 também são abordadas.

No Capítulo 4 são apresentados os requisitos e a arquitetura de um sistema de repositório baseado em MOF e discutido o projeto e implementação de um compilador da linguagem MODL.

No capítulo 5 é apresentado um exemplo de utilização do compilador no contexto de uma plataforma de governo eletrônico. Metamodelos descritos em MODL são apresentados.

O capítulo 6 apresenta as conclusões desta dissertação, juntamente com sua contribuição e trabalhos futuros. Em seqüência, estão as referências bibliográficas.

2 FUNDAMENTOS: METADADOS E REPOSITÓRIO DE METADADOS

2.1 Introdução

O conceito de metadados é antigo, porém, o papel de metadados e sua importância no ambiente computacional é novo e os principais fatores que têm motivado esta mudança são:

(i) mudanças contínuas da tecnologia;

(ii) aumento da complexidade dos processos de negócio;

(iii) processamento distribuído e;

(iv) comércio eletrônico.

Com o advento de sistemas distribuídos abertos e heterogêneos, criou-se a necessidade de integração, federação e interoperação de metadados descritos por diferentes modelos [Kerhervé 1997]. Este novo contexto impôs aos modelos o requisito de extensibilidade para assegurar a evolução destes para atender as necessidades de um domínio específico.

Mais recentemente, a proposta de uma infra-estrutura para o desenvolvimento de sistemas de software orientados a modelos (*Model-Driven Architecture* – MDA) define um repositório de metadados como um de seus componentes centrais.

Neste capítulo, serão apresentados inicialmente, na seção 2.2, conceitos básicos relacionados a metadados. A arquitetura de meta-modelagem em quatro camadas é descrita e para exemplificá-la, são apresentados dois padrões relevantes na área de engenharia de software. Em seguida, uma revisão dos conceitos de orientação a objetos é apresentada, uma vez que esta é a tecnologia predominante no desenvolvimento atual de software. Por último, nesta seção, são apresentados os fundamentos de um sistema de repositório de metadados.

Finalizando o capítulo, na seção 2.3 é descrita a proposta para desenvolvimento de sistemas de software orientados a modelos. A importância de um repositório de metadados em MDA é abordada.

2.2 Conceitos Básicos

2.2.1 Metadados

O termo meta se aplica a qualquer linguagem, notação ou sistema formal com recursos suficientes para definir a si próprio [Atkinson 1997]. No contexto de modelagem orientada a objetos, a proposta é utilizar conceitos de modelagem orientada a objetos para essa descrição. Nesta dissertação o termo meta-objeto é considerado sinônimo de metadado.

Em uma abordagem tradicional, o propósito do uso de metadados é facilitar o acesso, o gerenciamento, o processamento e o compartilhamento de uma grande coleção de dados estruturados e/ou não-estruturados [Kerhervé 1997]. Atualmente, a importância da utilização de metadados é observada em diferentes áreas, aplicações e tecnologias emergentes. Metadados são considerados essenciais em ambientes de *data warehouse*, possibilitando determinar a origem de um dado em particular, as transformações ocorridas nos dados, etc. Mais recentemente, em ambientes de sistemas distribuídos, metadados emergem como elementos fundamentais para o intercâmbio de informações entre sistemas e a descrição de recursos normalmente complexos encontrados neste tipo de ambiente.

A definição de metadados está fortemente relacionada com os conceitos de modelagem e metamodelagem. No contexto de desenvolvimento de sistemas de software, modelagem refere-se à descrição de metadados estruturados representando entidades — e os relacionamentos entre elas — manipuladas por aplicações e sistemas em tempo de execução. Uma coleção destes metadados forma um modelo da aplicação com detalhes suficientes para permitir instanciação e introspecção das entidades manipuladas pela aplicação. Meta-modelagem refere-se à representação de metamodelos, que são modelos que descrevem modelos, definindo as construções usadas na modelagem de uma aplicação ou sistema. [Costa 2001].

No contexto de engenharia de software, metadados têm sido freqüentemente relacionados às ferramentas CASE, documentando modelos da aplicação. O surgimento de tecnologias orientadas a objeto aumentou a necessidade do uso de metadados. Em tempo de compilação, metadados tais como definição de classes e métodos, são obtidos e preservados para serem usados em tempo de execução. É utilizado, neste caso, o conceito de reflexão que significa que

um software possui informações que descrevem a si mesmo e, em tempo de execução, pode acessá-las e usá-las [Staudt 1999].

Um padrão de metadados define um conjunto de descritores organizados segundo um modelo e empregado para descrever as entidades de um domínio de aplicação [Barreto 1999]. Conjuntos, sub-conjuntos e super-conjuntos de metadados têm sido propostos ou adaptados para atender domínios específicos tais como: sistemas geográficos, tecnologia da informação, biblioteca digital, etc. A necessidade de se compartilhar diferentes padrões sob a perspectiva de um ambiente integrado fez com que surgissem as arquiteturas genéricas de metadados, fornecendo uma organização para metadados oriundos de diferentes padrões. As arquiteturas de metadados, que serão tratadas na próxima seção, consideram os diversos padrões de metadados como abordagens complementares, que podem ser integradas com a finalidade de atender os requisitos de comunidades distintas.

2.2.2 Arquitetura de Metadados

É consenso que o uso de metadados é um mecanismo que facilita a integração e o intercâmbio de informação entre sistemas heterogêneos. É consenso também que um padrão de metadados comum, para uso universal, é uma solução improvável [Frankel 2003, Lagoze 1996]. Uma arquitetura de metadados provê uma infra-estrutura onde diferentes iniciativas para padronização de metadados coexistem.

As arquiteturas de metadados *Warwick Framework*, *Meta Content Framework, Resource Description Framework* e *Resource Description Framework Schema* são apresentadas com detalhes em [Barreto 1999]. Nesta seção é descrita a arquitetura de metamodelagem em quatro camadas porque esta é a arquitetura adotada no padrão *Meta-Object Facility* (MOF), objeto de estudo desta dissertação.

Uma arquitetura de metamodelagem usa metamodelos como base do seu mecanismo de abstração. A arquitetura proposta por Kerhervé [Kerhervé 1997] foi definida a partir da constatação de que os esforços, além da padronização, deveriam ser aplicados na integração, federação e interoperabilidade de padrões de metadados existentes e na extensibilidade destes. Segundo Kerhervé, para alcançar estes objetivos deve-se concentrar em um nível de abstração

acima, ou seja, na modelagem de metadados ou, nos elementos que definem os modelos. A arquitetura de metamodelagem proposta por Kerhervé é ilustrada na Figura 2.1 e descrita a seguir.

A camada de Meta-metamodelo forma a base da arquitetura de metamodelagem, definindo uma linguagem para a especificação de metamodelos, possibilitando desta forma, a representação homogênea e a interoperabilidade das camadas inferiores. Um meta-metamodelo pode definir múltiplos metamodelos, e podem haver múltiplos meta-metamodelos associados a cada metamodelo. Exemplos de elementos desta camada são: meta-meta-metadados tais como entidade, associação, restrição semântica, etc.

Um Metamodelo define a linguagem para especificar os modelos da camada imediatamente abaixo. Exemplos de elementos da camada de metamodelagem são: meta-metadados descrevendo os elementos que podem ser usados no modelo, isto é, classe, atributo, relacionamento, etc.

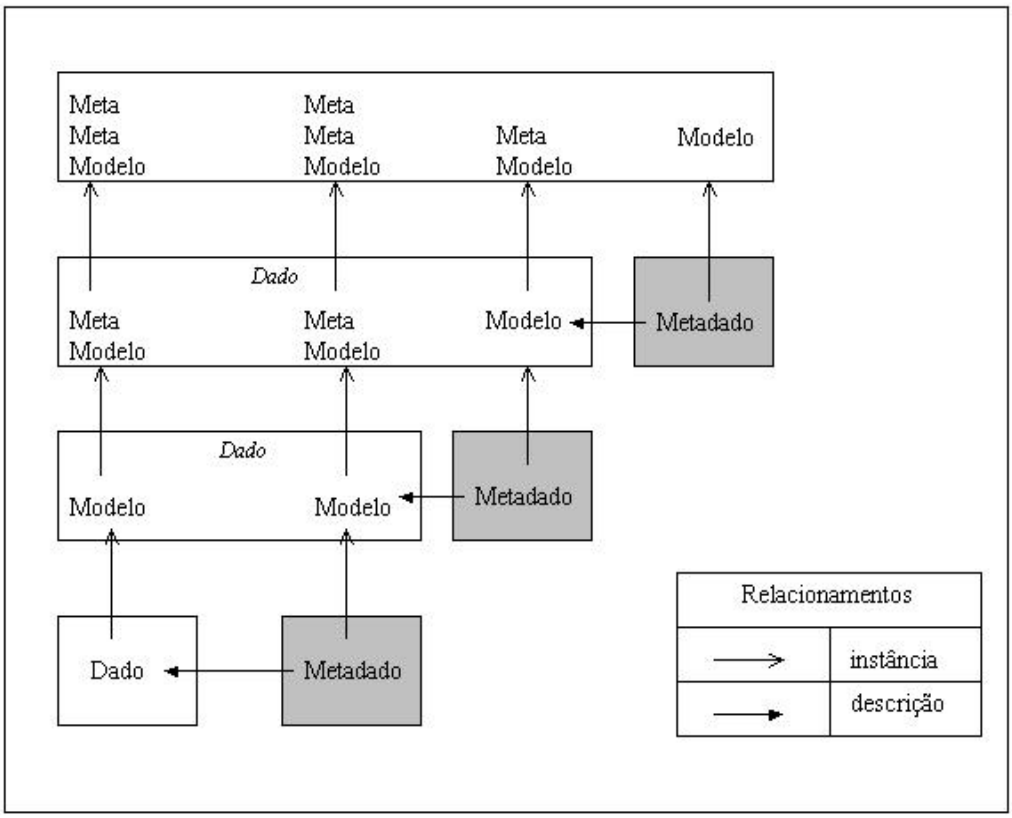

Figura 2.1 - Arquitetura de metamodelagem em quatro camadas [Kerhervé 1997]

A camada de modelos define uma linguagem que descreve a semântica e a estrutura de um domínio de informação. Exemplos da camada de modelo são: metadados representando uma classe específica de um domínio, a definição de um atributo, relacionamentos, etc.

A camada de dados constitui-se das entidades manipuladas em tempo de execução pela aplicação ou sistema modelado. Exemplos na camada de dados são: objetos, métodos, valor de um dado, valor de um atributo, etc.

Os padrões OIM, CDIF, UML e MOF são exemplos de importantes iniciativas que adotaram a arquitetura de metamodelagem em quatro camadas. Nas seções a seguir são apresentados detalhes sobre CDIF e OIM. Embora estes dois padrões tenham tido seu desenvolvimento descontinuado, ambos são trabalhos relevantes no contexto de compartilhamento de metadados relacionados ao desenvolvimento de software. Uma característica comum aos padrões OIM e CDIF e, talvez o ponto fraco destas iniciativas, foi o enfoque na padronização dos metamodelos, como poderá ser visto a seguir. MOF, ao contrário, teve o enfoque na padronização do meta-metamodelo para suportar descrição de diferentes metamodelos e portanto, oferecer uma abordagem mais aberta e flexível [Costa 2001].

Originalmente, o meta-metamodelo para a definição de UML era a própria linguagem UML. Atualmente, é utilizado o meta-metamodelo da especificação MOF e, portanto, UML é considerado um metamodelo MOF. O capítulo 3 desta dissertação apresenta uma completa descrição do padrão MOF.

2.2.3 Open Information Model – OIM

Open Information Model (OIM) [MDC 1999] é um padrão de metadados proposto inicialmente por um grupo de empresas, entre elas, empresas líderes de mercado na área de desenvolvimento de software. Em 1998 a especificação do padrão foi transferida para a organização *Meta Data Coalition* (MDC), um consórcio de empresas criado com o objetivo de definir, implementar e evoluir um formato padrão para o intercâmbio de metadados bem como os mecanismos de suporte necessários. Em 2002, a MDC uniu-se ao consórcio OMG e o padrão OIM foi utilizado no desenvolvimento do padrão *Common Warehouse Metamodel* (CWM) [OMG 2003b]. Mesmo descontinuado, este padrão ainda vale como objeto de pesquisa, uma vez

que foi projetado para atender os seguintes requisitos: (i) extensibilidade; (ii) alinhamento com os padrões então existentes; e (iii) o uso das melhores práticas para a especificação dos metadados.

O padrão OIM utiliza o padrão UML como meta-metamodelo para definição, representação e extensão de metamodelos de domínios específicos. Na especificação OIM, o termo sub-modelo é utilizado ao invés do termo metamodelo e, o termo modelo é usado para indicar o domínio dos metadados. Sub-modelos são agrupados nos seguintes modelos: Análise e Projeto, Engenharia de Negócio, Objetos e Componentes, Banco de Dados e *Data Warehouse* e Gerenciamento do Conhecimento.

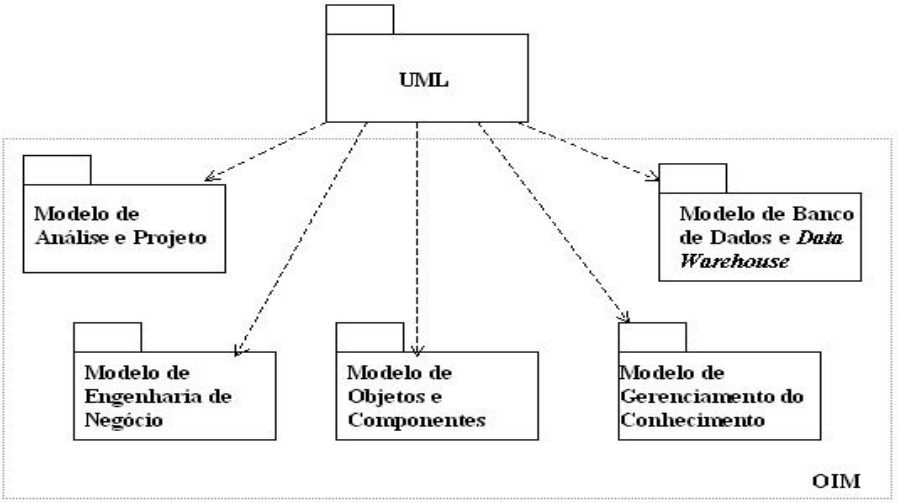

Figura 2.2 - Metamodelos do padrão OIM

Na Figura 2.2 foi utilizada a notação UML para mostrar os diferentes modelos do padrão OIM e seu relacionamento com o padrão UML. Uma breve descrição dos modelos é apresentada em seguida.

- Modelo de Análise e Projeto: cobre o domínio da modelagem orientada a objeto e projeto de sistemas de software. O modelo provê conceitos para descrever problemas e soluções ao longo do ciclo de vida do software;

- Modelo de Bancos de Dados e *Data Warehousing*: consiste de metadados para gerenciamento de esquemas para projetos de bancos de dados, reuso de esquemas e *data warehouse*;

- Modelo de Objetos e Componentes: cobre os diferentes aspectos envolvidos no ciclo de vida de desenvolvimento de componentes;

- Modelo de Engenharia de Negócios: provê os metadados necessários para capturar os objetivos, a organização e a infra-estrutura de um negócio como também os seus processos e regras de negócio;

- Modelo de Gerenciamento do Conhecimento: provê metadados necessários para capturar a terminologia de negócio, suas semânticas, seus relacionamentos e mapeamentos para o modelo físico.

A ênfase do padrão OIM é prover suporte à interoperabilidade entre ferramentas de desenvolvimento e repositórios através da descrição formal de metadados. Gerenciamento de metadados e mecanismos de acesso não são especificados no padrão.

2.2.4 Case Data Interchange Format - CDIF

CDIF (*CASE Data Interchange Format*) [EIA 1994] é uma família de padrões de metadados proposta para promover o intercâmbio de dados entre ferramentas CASE e repositórios. A Figura 2.3 ilustra os principais componentes do padrão CDIF.

O meta-metamodelo CDIF estabelece as regras para a definição do Metamodelo Integrado CDIF e do Formato de Transferência CDIF. O Metamodelo Integrado CDIF, ilustrado na Figura 2.4, é dividido em partes denominadas *Subject Areas*.

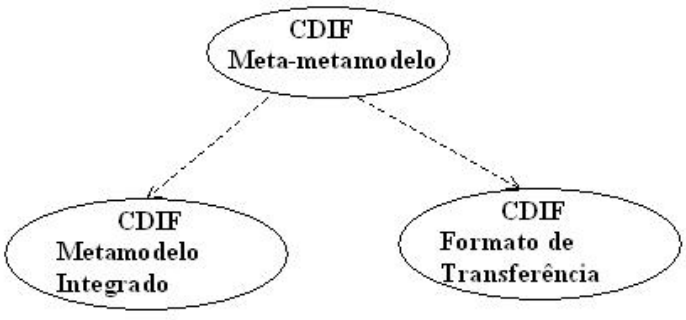

Figura 2.3 – Componentes do padrão CDIF

Fundação e a Área Comum representam partes compartilhadas pelas outras áreas. Fundação é composta por meta-objetos tais como, meta-entidades, meta-atributos e meta-relacionamentos, essenciais para a transferência de modelos. A Área Comum contém meta-objetos que representam abstrações comuns às outras áreas e que não são essenciais para a transferência de modelos.

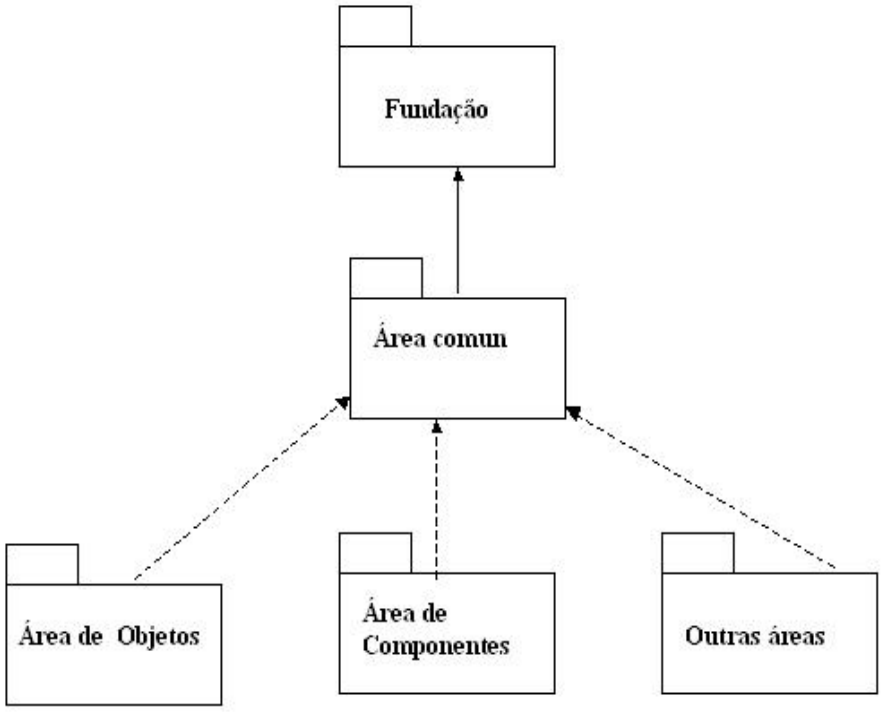

Figura 2.4 - Metamodelo Integrado CDIF

Uma área é composta por uma coleção de meta-objetos que descreve técnicas específicas encontradas em ferramentas CASE, como modelagem do fluxo de dados e análise e projeto orientado a objeto. Meta-objetos de uma área podem ser compartilhados por outras áreas. CDIF provê um mecanismo para extensão de uma área para ferramentas que suportem o padrão. Neste caso, a ferramenta exportadora "avisa" a ferramenta importadora sobre as extensões antes da transferência dos metadados.

CDIF suporta diferentes formatos para transferência de dados. O metamodelo Formato de Transferência CDIF permite a definição de um formato de transferência através da descrição de uma sintaxe e suas regras de codificação.

2.2.5 Considerações sobre Metamodelagem Orientada a Objetos

Metamodelos vêm assumindo um papel de fundamental importância em ambientes distribuídos. Esta seção apresenta os conceitos relacionados a metamodelagem, no contexto de ambientes de objetos, com a finalidade de apresentar os conceitos básicos adotados em MOF. Os principais conceitos de modelagem orientada a objetos são revistos a seguir [Atkinson 1997]:

- **Objeto**: um objeto encapsula um estado, representado por um conjunto de atributos, e comportamento, representado por um conjunto de métodos. Um objeto também tem um identificador único;

- ***Link***: representa um relacionamento entre objetos;

- ***Template*/instância**: as características dos objetos são definidas coletivamente para um conjunto de objetos similares. A descrição coletiva é denominada *template* e pode ser instanciada várias vezes, criando instâncias idênticas quanto a estrutura e o comportamento. Tradicionalmente, o *template* de um objeto é conhecido por classe, o *template* de *link* é conhecido por associação, o *template* de método é conhecido por operação e o *template* de atributo é conhecido por propriedade;

- **Generalização**: ou especialização, é o relacionamento entre uma classe e uma ou mais versões refinadas desta classe;

- **Instância_de**: é o relacionamento entre um *template* e uma instância. É um relacionamento um-para-muitos, uma vez que uma instância só pode ser instância_de um *template*;

- **Membro_de**: este relacionamento se aplica aos diferentes conjuntos de instâncias que uma classe pode possuir. Os diferentes conjuntos ocorrem devido ao relacionamento Generalização, que permite que um objeto seja membro de uma classe da qual não é uma instância direta. É um relacionamento de muitos-para-muitos, uma vez que, uma classe pode ter muitas super-classes;

- **Polimorfismo**: este mecanismo permite que, em tempo de execução, um objeto possa desempenhar o papel de uma instância de uma classe da qual ele é membro;

- **Tipo/Interface**: quando classes são usadas para suportar o mecanismo de polimorfismo, elas são denominadas tipos. Elas descrevem a interface de uma abstração sem definir a maneira como a interface será implementada.

Em uma arquitetura de modelagem em multi-camadas, o relacionamento instância_de entre uma camada e a camada imediatamente abaixo pode ser realizado de duas maneiras: metamodelagem rigorosa (*strict*) ou metamodelagem fraca (*loose*). Na metamodelagem rigorosa, todo elemento de um modelo do nível *Mn* é instância_de exatamente um elemento do modelo do nível *Mn+1*. Na metamodelagem fraca, um modelo do nível *Mn* é instância_de um modelo do nível *Mn+1*.

A maioria das propostas baseadas em uma arquitetura multi-camadas utiliza a metamodelagem fraca, permitindo que um *template* e suas instâncias coexistam no mesmo nível. Por exemplo, a presença de objetos e classes ou *links* e associações no mesmo nível. A metamodelagem rigorosa não permite esta coexistência. Segundo Atkinson [Atkinson 1997], quando a metamodelagem é fraca, os níveis de modelagem não passam de pacotes agrupando elementos de modelagem logicamente relacionados.

2.2.6 Repositório de Metadados

Um sistema de repositório ou, simplesmente repositório, é um sistema para armazenar e gerenciar metadados. O conteúdo de um repositório pode ser composto por artefatos tais como softwares, definições de interfaces, documentos, mapas, sistemas de informação, etc. Repositórios têm sua origem em sistemas de controle de versão de arquivo utilizados principalmente por equipes de desenvolvimento de software. Em um próximo passo relevante, repositórios foram incorporados às ferramentas CAD (*Computer Aided Design*) e CASE (*Computer Aided Software Engineering*) com o objetivo de gerenciar os modelos de informação e os artefatos gerados por estas ferramentas. Na década de 90, o uso de repositórios foi estendido a diversas tecnologias tais como, *groupware*, hipertexto, multimídia e *data warehouses* e passaram a gerenciar metadados sobre qualquer tipo de recurso. Atualmente, repositórios são utilizados em sistemas de aplicações empresariais para gerenciar os metadados de toda a organização, incluindo sua infra-estrutura.

Assim como muitos sistemas de software, sistemas de repositórios têm sido desenvolvidos orientados a objeto. Ou seja, estes sistemas armazenam, acessam e manipulam objetos ao invés de registros ou entidades. Segundo Bernstein [Bernstein 1998], um sistema de repositório de metadados é definido como uma infra-estrutura composta por uma base de (meta) dados, um gerenciador de repositório, um modelo de informação e um conjunto de ferramentas integradas A Figura 2.5 apresenta esta infra-estrutura.

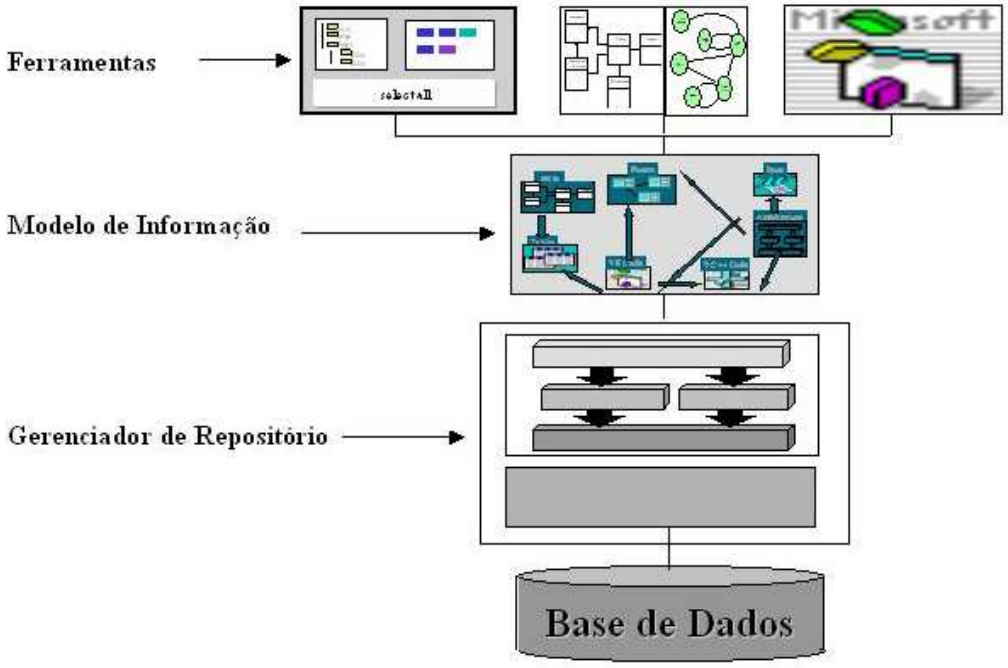

Figura 2.5 – Infra-estrutura de um sistema de repositório [Bernstein 1998]

A base de dados de um repositório pode ser qualquer sistema de persistência de dados como, por exemplo, sistema de arquivos, banco de dados relacional, banco de dados orientado a objetos, etc. As funções relacionadas ao gerenciamento de banco de dados – *view*, controle de acesso, integridade, concorrência, etc – devem ser fornecidas pelo repositório, ou, mais tipicamente, pelo sistema gerenciador do banco de dados suportado.

O gerenciamento de metadados requer funções adicionais, além daquelas fornecidas pelos sistemas convencionais de gerenciamento de banco de dados. O gerenciador de repositório é uma camada de software acima da base de dados e que provê funções específicas a um sistema de repositório [Bernstein 1994, 1998]. As funções listadas a seguir são desejáveis mas a presença de cada uma dependerá do propósito do repositório.

- ***Check-in*/*Check-out***: permite o usuário copiar um objeto do repositório para sua área de trabalho e devolvê-lo quando não for mais necessário. Esta função é importante para atividades de longa duração (dias, semanas ou meses) onde o uso de transação é uma opção impraticável.

- **Controle de versão**: metadados passam por revisões ao longo de sua vida. O repositório deve manter este histórico e permitir a identificação, a busca e a comparação de versões de um metadado.

- **Gerenciamento de configuração**: permite o gerenciamento de coleções de metadados relacionados. Uma configuração define uma visão física dos metadados relacionados. Por exemplo, metadados de diferentes versões relacionados em uma determinada configuração. Devem existir no repositório mecanismos para a identificação e a representação de configurações e mecanismos para a definição de restrições sobre uma configuração.

- **Notificação**: alterações em um metadado ou um pedido de *check-out*, são exemplos de notificações que o repositório pode prover.

- **Gerenciamento de contexto**: um contexto define uma visão semântica dos metadados armazenados no repositório. Ou seja, os metadados estão relacionados para uma determinada tarefa ou contexto. Semelhante ao gerenciamento de configuração, o repositório de metadados deve prover mecanismos para a identificação e a representação de contextos.

- **Controle de ciclo de vida**: considera as fases pelas quais um metadado passa. Por exemplo, na área de engenharia de software, uma aplicação passa pelas fases de requisitos, especificação, projeto, implementação, teste e integração. Um repositório de metadados deve suportar um modelo de *workflow* para alterar o estado de um metadado manual ou automaticamente (por meio de notificação).

- **Gerenciamento de relacionamentos**: um relacionamento deve ser tratado como um metadado e como tal, o repositório deve suportar propriedades e semântica que, de outro modo, teriam de ser providos pelas ferramentas.

O modelo de informação especifica a estrutura e a semântica dos metadados armazenados e, através deste modelo, as ferramentas acessam o conteúdo da base de dados. Para que o repositório seja aberto e facilmente extensível, é importante a adoção de um padrão que adote uma arquitetura de metadados. A arquitetura em camadas, como a apresentada neste capítulo, oferece uma organização para o uso e manutenção dos metadados e também, uma abordagem aberta onde metamodelos existentes podem ser estendidos e novos metamodelos podem ser incluídos.

Um repositório deve permitir a integração de diferentes ferramentas, tais como compiladores, ferramentas de modelagem, *browsers*, etc. Um conjunto bem definido de APIs deve permitir: (i) a criação, a atualização e a remoção de artefatos do repositório; (ii) a construção de objetos que suportem funções do repositório tais como versão, configuração, regras e contexto; (iii) a representação dos relacionamentos de uma maneira flexível; (iv) a navegação entre os objetos do repositório e; (v) mecanismo para consulta.

A interoperabilidade de metadados é uma funcionalidade fundamental de um repositório. Um formato padrão para o intercâmbio de informação entre ferramentas e outros repositórios também deve ser fornecido por um sistema de repositório.

O conceito gerenciamento de tipos é considerado um refinamento do conceito gerenciamento de metadados. Os metadados no gerenciamento de tipos descrevem especificamente entidades usadas na construção de sistemas de software. As funções de um repositório de tipos são as mesmas de um repositório de metadados acrescidas do gerenciamento de nomes que associa um nome único a uma definição de tipo, suportando, desta maneira, busca e verificação dinâmica de tipos [Costa 2001]. O repositório de interfaces Corba IDL é um exemplo de um repositório de tipos.

2.3 Arquitetura Orientada a Modelos

Model-Driven Architecture (MDA) [OMG 2003d] é uma proposta do consórcio OMG, dando continuidade à incessante busca por soluções para o problema de integração de sistemas. Em ambientes altamente distribuídos e em constante renovação tecnológica, a iniciativa MDA visa melhorar a portabilidade e interoperabilidade de aplicações, aumentar o reuso de

componentes, reduzir o custo do desenvolvimento de software e facilitar a integração de aplicações legadas. Para alcançar este objetivo, MDA propõe separar a especificação da funcionalidade da aplicação da especificação de sua implementação em uma determinada plataforma, utilizando modelos formais e semiformais e transformações entre os modelos resultantes. As principais categorias de modelos utilizadas em MDA são os modelos independentes de plataforma (*Platform Independent Model* - PIM) e os modelos específicos de plataforma (*Platform Specific Model* - PSM).

No contexto MDA, um modelo é a representação (ou parte dela) de uma função, estrutura e/ou comportamento de um sistema em uma linguagem com uma sintaxe e semântica bem definidas e possivelmente, regras de análise e inferência. Exemplos de modelo são diagramas UML, interfaces IDL, código fonte, etc. Uma plataforma é definida como um conjunto de tecnologias que provê um conjunto coerente de funcionalidades através de interfaces e padrões de uso. Detalhes da implementação das funcionalidades da plataforma são irrelevantes para sistemas e aplicações que utilizam a plataforma [OMG 2003d].

Os modelos PIM são representados numa linguagem de modelagem também independente de plataforma, como UML. Esses modelos são traduzidos, em seguida, para modelos PSM utilizando algum mapeamento de modelos PIM para linguagens de implementação ou plataforma (p.ex. J2EE, CORBA, .NET), utilizando regras formais e semiformais de transformação. Quatro tipos de transformações são identificadas em MDA [OMG 2003d]:

- PIM-PIM : usada quando modelos independentes de plataforma são especializados, filtrados ou refinados no ciclo de vida do desenvolvimento de software. Por exemplo, a transformação de um modelo da fase de análise para um modelo da fase de projeto.

- PIM-PSM : esta transformação ocorre quando o modelo PIM se encontra suficientemente refinado, para ser projetado em uma infra-estrutura de execução. A projeção é baseada nas características da plataforma escolhida.

- PSM-PSM : esta transformação se aplica quando é necessário um refinamento de um modelo dependente de plataforma. Este é o caso de componentes que são empacotados selecionando serviços e suas configurações. Uma vez empacotados, a implantação é feita especificando os dados iniciais, hardware, configuração do *container*, etc.

- PSM-PIM : esta transformação é necessária quando se deseja abstrair uma implementação já existente em um modelo independente de plataforma.

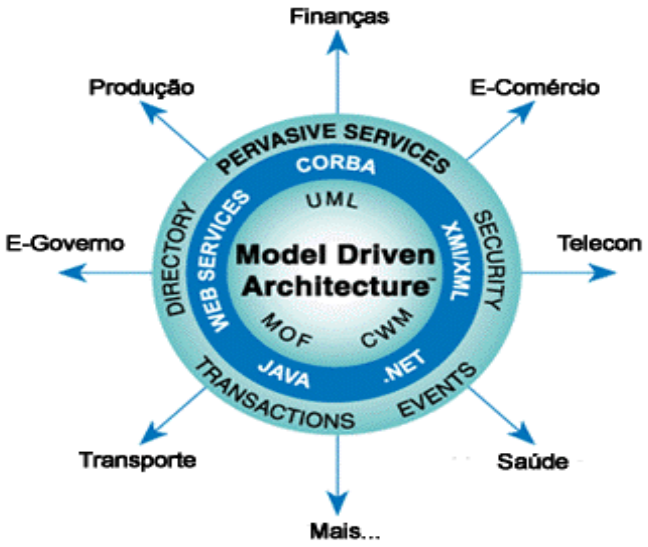

Figura 2.6 – Arquitetura Orientada a Modelos

A Figura 2.6 apresenta um exemplo de uma arquitetura MDA onde são empregados padrões do consórcio OMG e plataformas *middleware* disponíveis no mercado. Neste exemplo, o núcleo MDA é composto por uma linguagem para modelagem (UML), um repositório de metadados extensível (MOF), um mecanismo para intercâmbio de metadados em um formato padrão (XMI) e, por último, um conjunto de metamodelos descrevendo tecnologias estabelecidas tais como, *data warehouse* e *business intelligence*, e metamodelos que promovam a integração de sistemas legados (CWM). Plataformas *middleware* suportadas por esta arquitetura devem ser modeladas em modelos PSM contendo regras de mapeamento que permitirão a transformação de um modelo PIM para o modelo PSM. As características e a arquitetura de serviços genéricos e abrangentes (*pervasive services*) tais como serviço de diretório, persistência, segurança e transação, são capturadas primeiro em modelos PIM e, quando claramente definidas, são gerados os modelos PSM para as plataformas *middleware* suportadas. Por último, diferentes domínios são representados por modelos normativos PIM expressos em UML e por modelos normativos PSM também expressos em UML mas com definições de interfaces para uma plataforma alvo. A base comum sob estes modelos permite a geração automática de parte do código da implementação.

Um repositório de metadados é componente fundamental para uma solução MDA. É essencial que os diferentes metamodelos e a enorme quantidade de modelos sejam gerenciados de maneira eficiente e padronizada [Frankel 2003]. Repositórios federados e distribuídos devem ser considerados, indicando o uso de mecanismos para replicar ou particionar os metadados.

O ambiente de uma solução MDA é muito dinâmico e devido a isso o repositório deve prover acesso rápido e ser sempre ativo. Ou seja, propagar alterações na sua base de dados para as aplicações interessadas, através de um mecanismo de notificação de evento.

MDA é uma tecnologia emergente, anunciada em fevereiro de 2002. Muitas aplicações da arquitetura MDA, neste primeiro momento, visam resolver o problema de interoperabilidade entre aplicações, ferramentas e base de dados através do intercâmbio de modelos. A longo prazo, Poole [Poole 2001] visualiza o uso de MDA nas seguintes áreas:

- Sistemas Baseado em Conhecimento: a funcionalidade de um sistema tornar-se-á gradualmente baseada em conhecimento, capaz de automaticamente descobrir propriedades em comum de um domínio diferente, tomar decisões inteligentes e criar e armazenar inferências;

- Arquitetura Dinâmica: representa o fim dos mapeamentos, em que modelos serão diretamente interpretados pelo software. Por exemplo, uma nova versão de um modelo fará com que todos os sistemas de software tenham seu comportamento alterado, de modo a refletir as mudanças no modelo;

- Sistemas Adaptativos: a arquitetura dinâmica e sistemas baseados em conhecimento produzirão sistemas altamente dinâmicos que poderão ser atualizados, ou pelo próprio sistema, ou por pessoas especializadas no domínio, não necessariamente um especialista em software.

2.4 Considerações Finais

Os conceitos de arquitetura de metadados e sistemas de repositório apresentados neste capítulo são essenciais para o entendimento do conteúdo dos próximos capítulos. Os padrões

CDIF e OIM, assim como MOF, adotaram uma arquitetura de metadados em quatro camadas como mecanismo para integrar diferentes padrões de metadados.

Metadados são armazenados em um sistema de repositório que possui funções específicas tais como, controle de versão e controle do ciclo de vida do metadado. Por fim, este capítulo introduziu a arquitetura MDA, abordando seus principais conceitos.

3 *META-OBJECT FACILITY*

3.1 Introdução

A especificação Me*ta-Object Facility* (MOF) [OMG 2002a] apresenta a solução tecnológica do consórcio OMG para o gerenciamento de metadados em um ambiente distribuído. Os objetivos de MOF são: (i) definir uma linguagem abstrata para descrição de metamodelos de diversos domínios e (ii) definir um modelo de programação padronizado para o gerenciamento de modelos instanciados de um metamodelo, com interfaces padronizadas para acesso e manipulação destes metadados.

Originalmente, MOF foi proposto para gerenciar metamodelos definidos para apoiar as fases de análise e projeto de sistemas de software, mais especificamente, para gerenciar os metamodelos UML, Corba IDL e outros do consórcio OMG. A palavra *facility* tinha então, o significado do contexto Corba, ou seja, MOF seria um conjunto de serviços do *middleware* Corba, que poderia ser compartilhado por diversas aplicações. No entanto, o escopo de MOF foi expandido e metamodelos de diferentes domínios, como por exemplo, *data warehouse*, também passaram a ser descritos em MOF.

A primeira especificação MOF, versão 1.1, foi ratificada em novembro de 1997, juntamente com a versão 1.1 da especificação UML. Desde então, foram realizadas diversas revisões, resultando nas especificações MOF 1.3 em junho de 1999 e MOF 1.4 em abril de 2002. Atualmente, o consórcio OMG tem dedicado esforços na elaboração da especificação MOF 2.0, onde o enfoque principal é criar uma fundação sólida para a iniciativa MDA.

Dois pontos de vista distintos são identificados na utilização de MOF:

Ponto de vista de modelagem: é o ponto de vista de um projetista de sistemas que utiliza MOF para definir metamodelos e modelos que descrevem metadados de um determinado domínio;

Ponto de vista dos dados: é o ponto de vista do programador que utiliza MOF para descobrir a estrutura e semântica de um recurso para, por exemplo, suportar o mecanismo de reflexão do sistema de software em desenvolvimento.

Este capítulo está organizado da seguinte maneira: as seções 3.2, 3.3 e 3.4 estão relacionadas com o objetivo (i) da especificação MOF, ou seja, definir uma linguagem para a descrição de diferentes metamodelos. Na seção 3.2 é apresentada a arquitetura de metamodelagem adotada em MOF, na seção 3.3 são apresentadas as construções da linguagem abstrata definida na especificação MOF e, na seção 3.4, o Modelo MOF da versão 1.4 [OMG 2002a] é descrito através desta linguagem abstrata.

A seção 3.5 está relacionada com o objetivo (ii) da especificação que define um modelo de programação padronizado para o gerenciamento de metadados com interfaces padronizadas. São apresentados os mapeamentos de MOF para as tecnologias Corba IDL e Java. Os mapeamentos para XMI e a notação HUTN também são abordados nesta seção.

Por último, na seção 3.6, são apresentados os requisitos da versão 2.0 de MOF cujos trabalhos de especificação estão em andamento no momento.

3.2 Arquitetura de Metamodelagem de MOF

O objetivo principal de MOF é prover uma capacidade aberta de modelagem de informações, suportando a definição de padrões e propostas de metadados novos ou já existentes. Conforme apresentado na seção 2.2.2, a adoção de uma arquitetura de metadados é essencial para se alcançar este objetivo.

A Figura 3.1 ilustra a arquitetura em quatro camadas adotada pelo consórcio OMG. Entre as diferenças desta arquitetura e outras mais tradicionais destacam-se: (i) o uso do paradigma de objetos para a modelagem e; (ii) a auto-descrição do meta-metamodelo utilizando a mesma linguagem definida para a descrição de metamodelos.

Existe uma estreita relação de MOF com UML: MOF utiliza a notação UML para a visualização dos metamodelos e as construções que compõem sua linguagem para definição de metamodelos são um sub-conjunto das construções UML.

A camada superior M3, é o nível de meta-metamodelo e corresponde ao Modelo MOF, a linguagem fixa usada para definição de metamodelos. A próxima camada, M2, é composta por metamodelos definidos através do Modelo MOF. Um metamodelo consiste de uma linguagem

com um conjunto de construções (meta-meta-objetos) para a modelagem de modelos de um determinado domínio. UML, EDOC e CWM são exemplos de metamodelos padrões definidos pelo consórcio OMG para o domínio de desenvolvimento de sistemas de software. No entanto, a camada M2 não é restrita a metamodelos padrões. Novos metamodelos podem ser definidos e os existentes podem ser estendidos a fim de atender a requisitos específicos de um domínio.

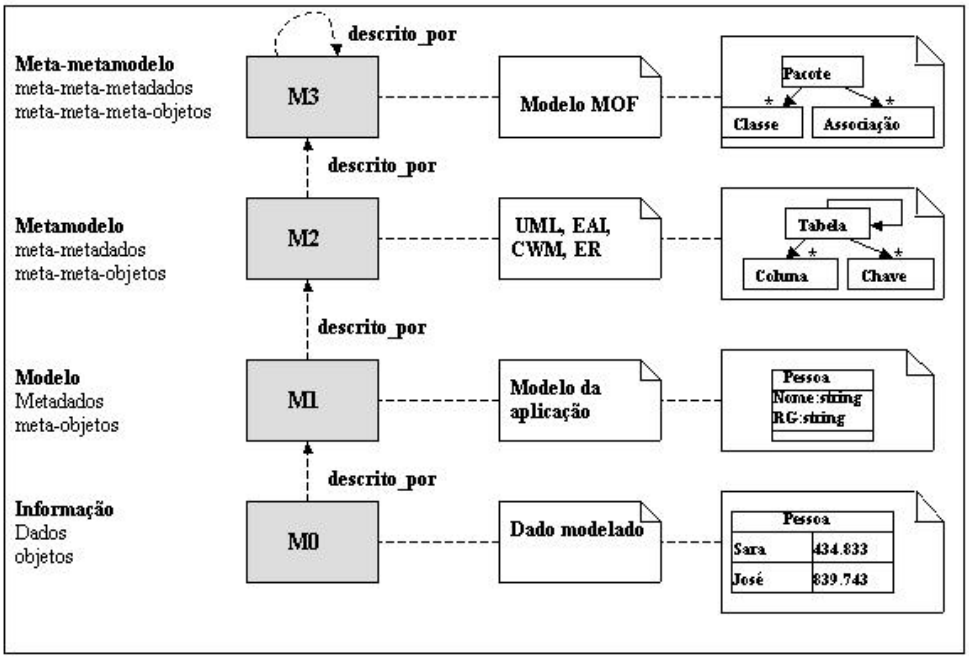

Figura 3.1 - **Arquitetura de metamodelagem MOF**

A camada M1 corresponde aos meta-objetos e consiste de modelos representando sistemas ou aplicações. Cada modelo em M1 é definido de acordo com um único metamodelo. Por último, na camada M0 encontram-se as instâncias ou dados descritos pelos modelos M1.

3.3 Construções para Metamodelagem

A atividade de metamodelagem consiste em definir modelos de informação para metadados. MOF utiliza a técnica de modelagem orientada a objetos e provê um conjunto de construções que é, essencialmente, um subconjunto das construções UML. Alguns termos aparecem em diferentes níveis de modelagem podendo causar confusão. Por exemplo, uma classe em UML é uma instância da classe class do metamodelo UML que por sua vez é instância da classe class de MOF. Por último, a classe class em MOF é descrita por si mesma. Para evitar este

tipo de confusão, o desejável seria utilizar o prefixo meta, mas o texto perderia em legibilidade. Nesta dissertação, optou-se pelo uso explícito do nível junto ao termo (classe-M1, classe-M2, etc) quando o contexto é dúbio.

As construções descritas a seguir são definidas na especificação MOF 1.4 e representam meta-meta-meta-objetos para definição de metamodelos.

3.3.1 Classe (*Class*)

Uma classe–M3 descreve um tipo de um metamodelo. Exemplos de tipos são uma classe do metamodelo UML e uma interface no metamodelo Corba IDL. Uma classe pode conter as construções atributo, operação, referência, exceção, constante, tipo de dados e restrição. Estas construções são definidas nas sub-seções a seguir. Classes-M1 têm suas propriedades e comportamento descritos por classes-M2 e, estas, por sua vez, têm suas propriedades e comportamento descritos por classes-M3.

MOF permite que uma classe seja especializada através do mecanismo de herança. Assim como UML, o termo generalização é utilizado para descrever um relacionamento de herança. Uma classe pode ser subclasse de uma ou mais classes.

Uma classe pode ser definida como abstrata (*abstract*), sendo usada somente para herança. Nenhuma instância pode ser criada de uma classe abstrata. Por outro lado, uma classe pode ser definida como folha (*leaf*) ou raiz (*root*). Declarando uma classe como folha previne-se a criação de subclasses e declarando uma classe como raiz previne-se a criação de superclasses.

3.3.2 Associação (*Association*)

Uma associação-M3 é usada para expressar um relacionamento em um metamodelo. Exemplos de relacionamento nos níveis inferiores são herança entre classes em UML e o relacionamento de *containment* entre interfaces e operações em Corba IDL. As instâncias de uma associação são denominadas *links*.

Em MOF os relacionamentos são binários, associando apenas um par de instâncias de classes. Cada instância desempenha um papel (*association end*) no relacionamento. Um papel

possui as propriedades: nome, tipo (agregação ou agregação por composição), multiplicidade para definir o número de instâncias de uma classe que podem desempenhar o papel e o tipo de ordem no conjunto de instâncias (*set*, *unordered set*, *bag* e *list*). Uma associação pode ser derivada, indicando que os métodos para manipulação deste objeto serão implementados fora do contexto de MOF.

O modelo de relacionamento suportado por uma associação é denominado orientado à consulta, pois as operações são aplicadas a todo o conjunto de instâncias associadas. É um modelo eficiente para uma busca global mas complexo, quando se deseja atualizar um relacionamento específico.

3.3.3 Pacote (*Package*)

Um pacote-M3 é um mecanismo de reuso e modularização, visando facilitar a definição de metamodelos complexos. Um pacote pode ser definido como folha (*leaf*) ou raiz (*root*), cujos significados são análogos aos usados em classes. O Modelo MOF suporta quatro mecanismos para composição e reuso de metamodelos:

- **Generalização**: de modo análogo às classes, um pacote pode ser uma generalização de um ou mais pacotes. Um sub-pacote herda todos os elementos do metamodelo pertencente ao super-pacote. Regras para prevenção de colisão de nomes são definidas no Modelo MOF.

- **Aninhamento**: um pacote pode conter outros pacotes que por sua vez também podem conter outros pacotes. Um pacote aninhado define um novo escopo para a definição de outros elementos do metamodelo.

- **Importação**: este mecanismo permite que um pacote use elementos definidos no pacote importado. Como a generalização, é um mecanismo de reuso, mas, neste caso, apenas os elementos desejados são reusados, ao contrário da generalização, quando todos os elementos do super-pacote são acessíveis ao sub-pacote.

- **Agrupamento** (*Cluster*): este mecanismo é semelhante ao de importação. A diferença ocorre no nível M1 conforme explicado a seguir.

Enquanto no nível M2 um pacote é uma construção para modularização, no nível M1, a instância de um pacote age como o invólucro para as instâncias dos metadados e define semânticas diferentes para os mecanismos descritos acima. No caso de uma generalização, um sub-pacote tem a capacidade de criar e manipular as instâncias de classes e associações herdadas. Um pacote aninhado não pode ser diretamente instanciado, ele existe em conjunção com o pacote que o contém. Não existe um relacionamento explícito entre o pacote que importa e o importado. Um pacote que importa não tem a capacidade de criar instâncias de um pacote importado. A aplicação cliente pode obter a instância do elemento importado através de uma instância em separado do pacote importado. Esta restrição de um pacote importado não ocorre em um pacote agrupado que comporta-se como estivesse aninhado no pacote agrupador. Ou seja, o ciclo de vida de uma instância agrupada depende do ciclo de vida do pacote que a agrupou. Entretanto, diferentemente de um pacote aninhado, é possível criar uma instância independente do pacote agrupado.

3.3.4 Restrição (*Constraint*)

Uma restrição restringe o estado e o comportamento de um ou mais elementos de um metamodelo. É um mecanismo que permite a extensão de um metamodelo, acrescentando regras de consistência não suportadas pelo Modelo MOF. As principais propriedades de uma restrição são: nome, linguagem, expressão e tipo de verificação. A expressão da restrição é expressa em uma linguagem e pode ser avaliada no contexto do metamodelo para verificar a sua validade. A especificação MOF recomenda a linguagem formal *Object Constraint Language* (OCL) para expressar uma restrição. Mas, a princípio, uma linguagem natural pode ser usada. O tipo de verificação pode ser imediata (*immediate*) ou adiada (*deferred*). Caso seja imediata, a restrição é verificada quando uma instância do elemento do metamodelo a que se aplica é criada ou alterada. Caso a restrição seja do tipo adiada, não existe um momento específico para a execução da verificação, que pode realizar-se, por exemplo, ao final de uma série de atualizações de atributos. Estes tipos de verificação são apenas uma recomendação da especificação MOF.

3.3.5 Atributo (*Attribute*)

Um atributo-M3 descreve uma propriedade de um tipo de um metamodelo. As principais propriedades de um atributo são: nome, tipo, multiplicidade e escopo. A multiplicidade de um

atributo indica quantos valores o atributo pode ter: opcional, único ou múltiplos valores. O escopo de um atributo pode ser nível de instância indicando que cada instância mantém seu próprio valor ou nível de classificador indicando que existe apenas um valor da propriedade para todas as instâncias da classe.

A propriedade tipo de um atributo pode ser um tipo de classe ou um tipo de dado. Caso seja um tipo de classe, o atributo define uma associação entre classes e o acesso a uma instância associada é feito através de operações *get* e *set* do atributo. Este modelo de relacionamento, denominado orientado à navegação, tem a vantagem de ser simples mas é ineficiente quando, por exemplo, se deseja fazer uma busca global no conjunto de instâncias associadas.

Um atributo pode ser derivado significando que não haverá geração automática de interfaces para sua manipulação e os valores a serem conferidos ao atributo serão implementados fora do contexto MOF.

3.3.6 Operação (*Operation*)

Uma operação-M3 é um mecanismo para acessar um comportamento associado a uma classe. Uma operação não especifica o comportamento, apenas define nomes e os tipos utilizados na chamada da operação. As principais propriedades de uma operação são: nome, parâmetros, exceções, parâmetro de retorno e escopo. De modo análogo ao atributo, o escopo de uma operação poder ser no nível de instância ou no nível de classificador. Uma operação definida a nível de instância é chamada somente na instância e apenas os valores de seus atributos podem ser modificados. Uma operação definida a nível de classificador pode ser chamada por qualquer instância e pode ser aplicada a qualquer instância da classe.

Os parâmetros de uma operação possuem as propriedades: nome, tipo, direção e multiplicidade. MOF não prevê suporte à geração de código de uma operação.

3.3.7 Referência (*Reference*)

Uma referência-M3 é uma construção semelhante a um atributo, mas tem o propósito de fornecer um mecanismo alternativo para acessar associações. Operações de acesso, equivalentes

às operações de acesso a atributos (*get* e *set*), são definidas para acessar e manipular o papel da associação referenciada. As propriedades de uma referência são: nome e papel referenciado.

3.3.8 Tipo de Dado (*Datatype*)

A construção tipo de dado é utilizada para representar os seguintes tipos de dados:

- **Tipo de dado primitivo**: o Modelo MOF define tipos de dados primitivos no pacote *PrimitiveTypes*. Na versão 1.3 da especificação, os tipos primitivos suportados são os mesmos definidos em Corba. Na versão 1.4, os tipos de dados primitivos suportados são apenas *boolean*, *integer*, *string*, *long*, *float* e *double,* o que permite a definição de metamodelos independentes de tecnologia;

- **Construtor de tipo de dado**: para a construção de um tipo de dado mais complexo, o Modelo MOF suporta enumeração (*enumeration*), estrutura (*struct*), coleção (*collection*) e *alias*.

3.3.9 Exceção (*Exception*)

Uma exceção define a assinatura de uma exceção que pode ser causada por uma operação.

3.3.10 Constante (*Constant*)

Uma constante define a ligação de um nome a um valor constante.

3.3.11 Tag

Uma *tag* permite estender a linguagem abstrata definida por um metamodelo. Consiste de um nome e de um valor e pode ser associada a qualquer elemento do metamodelo, com o objetivo de fornecer informação específica de um contexto como, por exemplo, versão do elemento.

3.4 O Modelo MOF

O Modelo MOF é o meta-metamodelo da camada M3 que define uma linguagem abstrata para descrever metamodelos da camada M2. O Modelo MOF define modelos do tipo entidade-

relacionamento. Ou seja, contém construções para criar os objetos de um metamodelo e os relacionamentos entre eles.

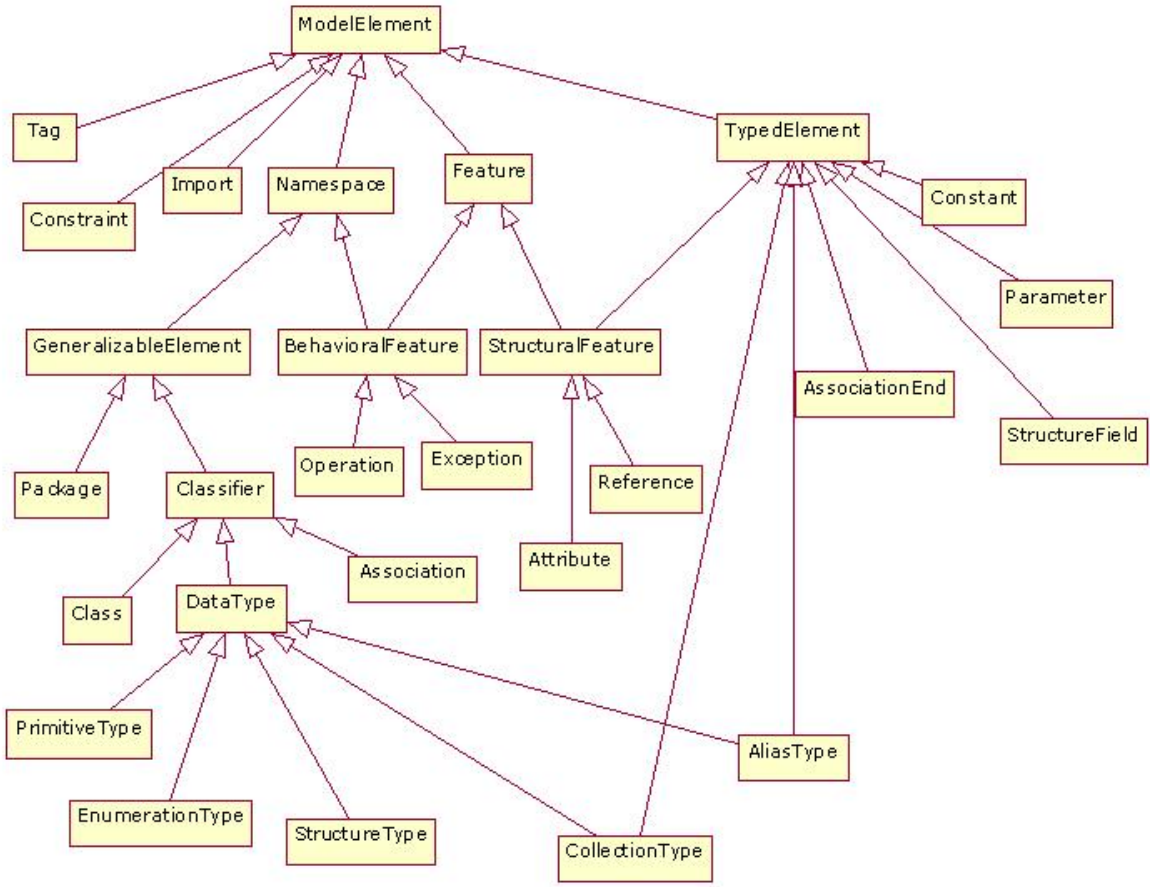

Figura 3.2 – Diagrama de classes do Modelo MOF

O diagrama de classes da Figura 3.2 apresenta a estrutura hierárquica das classes que definem o Modelo MOF. Estas classes estão agrupadas em um único pacote denominado Modelo (*Model*). É essencial entender MOF em termos de seu modelo e as semânticas associadas. Portanto, nas próximas seções, as principais classes do Modelo MOF são descritas com mais detalhes

No diagrama de classes da Figura 3.3 é apresentada a estrutura central do Modelo MOF cujas principais classes e associações são descritas a seguir. Tipos de dados, exceções e exemplos de restrições do Modelo MOF também são apresentados.

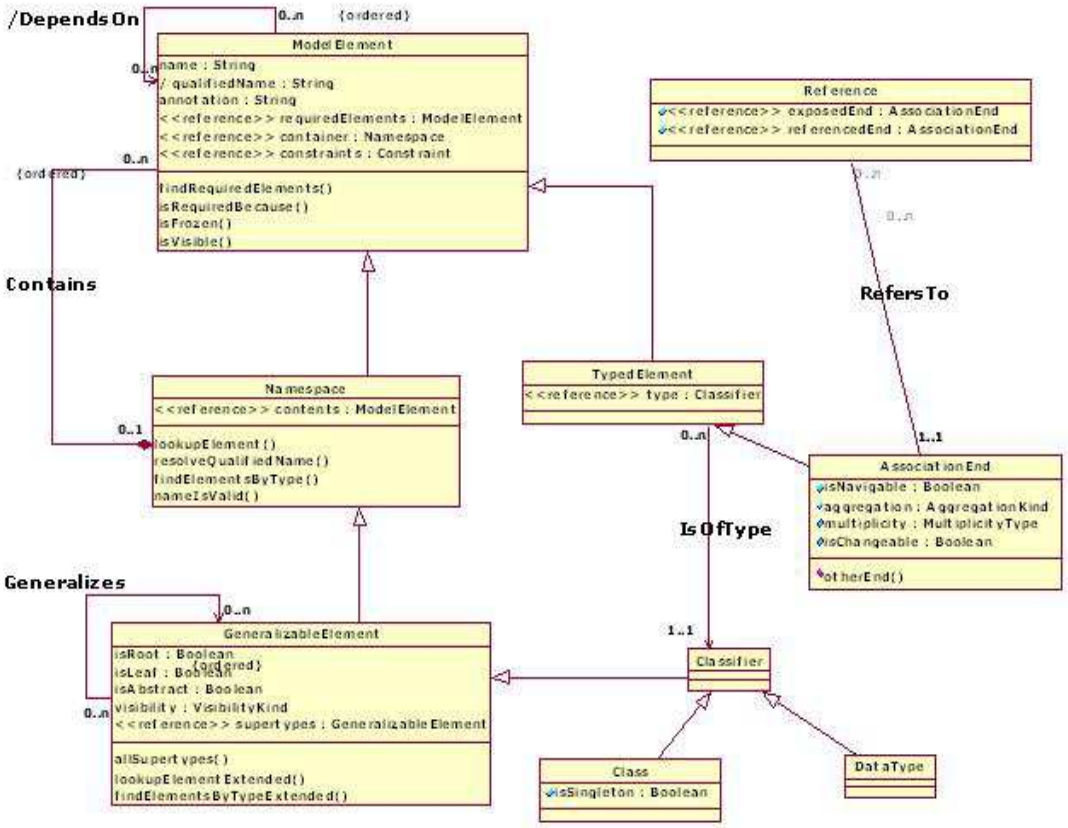

Figura 3.3 – Principais classes e associações do Modelo MOF

3.4.1 Principais Classes do Modelo MOF

As principais classes do Modelo MOF são :

- ***ModelElement***: é a classe raiz do Modelo MOF, estabelecendo um nome e uma descrição para todo elemento de um metamodelo;

- ***Namespace***: esta classe caracteriza um elemento que pode conter um conjunto de outros elementos. Uma instância de *namespace* pode conter vários elementos mas um elemento deve estar contido em apenas um *namespace*. O Modelo MOF define restrições para esta classe tais como: (i) os elementos que podem estar contidos em um *namespace* — por exemplo, um pacote pode conter classes e associações mas não pode conter atributos, que por sua vez, podem pertencer apenas a uma classe — e (ii) a unicidade do nome de um elemento em um *namespace*;

- ***GeneralizableElement***: é a classe comum a todas as outras classes que suportam o relacionamento generalização;

- ***TypedElement***: é a classe comum as classes *Attribute*, *Parameter* e *Constant* cuja definição exige a especificação de um tipo de dado;

- ***Classifier***: é a classe comum às classes *Class* e *DataType* que definem tipos;

- ***Class***: é o elemento fundamental para descrever um metamodelo. Todo elemento de um metamodelo é descrito por uma classe *Class*.

3.4.2 Principais Associações do Modelo MOF

As principais associações do Modelo MOF, ilustradas na Figura 3.3, são descritas a seguir:

- ***Generalizes***: esta associação relaciona um elemento *GeneralizableElement* e os seus elementos-filho em um modelo hierárquico;

- ***IsOfType***: esta associação relaciona um elemento *TypedElement* ao elemento *Classifier* que define seu tipo;

- ***DependsOn***: esta é uma associação derivada e relaciona um elemento *ModelElement* a outros que dele dependem;

- ***RefersTo***: esta associação relaciona uma referência a um papel (*association end*);

- ***Contains***: é a principal associação do Modelo MOF e é usada para relacionar, por exemplo, uma classe com suas operações e atributos, uma operação e seus parâmetros, etc. O Modelo MOF define restrições para esta associação, eliminando casos problemáticos e sem lógica. A Figura 3.4 ilustra as associações permitidas.

3.4.3 Tipos de Dados do Modelo MOF

Os principais tipos de dados do Modelo MOF são :

- ***PrimitiveType***: *integer*, *string*, *boolean*, *float*, *double* e *long*;

- ***MultiplicityType***: é uma estrutura de dados utilizada para especificar a multiplicidade de atributos, parâmetros, papéis e referências. Os campos desta estrutura são: *lower* (limite inferior), *upper* (limite superior), *isOrdered* (indica se o conjunto de valores é ordenado) e *isUnique* (para a unicidade de valores).

- ***VisibilityKind***: é um tipo de dado enumeração com os valores: *public*, *private* e *protected*;

- ***DirectionKind***: é um tipo de dado enumeração com os valores: *in*, *out*, *inout* e *return*;

- ***ScopeKind***: é um tipo de dado enumeração com os valores: *instance_level* e *classifier_level*;

- ***EvaluationKind***: é um tipo de dado enumeração com os valores: *immediate* e *deferred*;

- ***AggregationKind***: é um tipo de dado enumeração com os valores: *none* e *composite*.

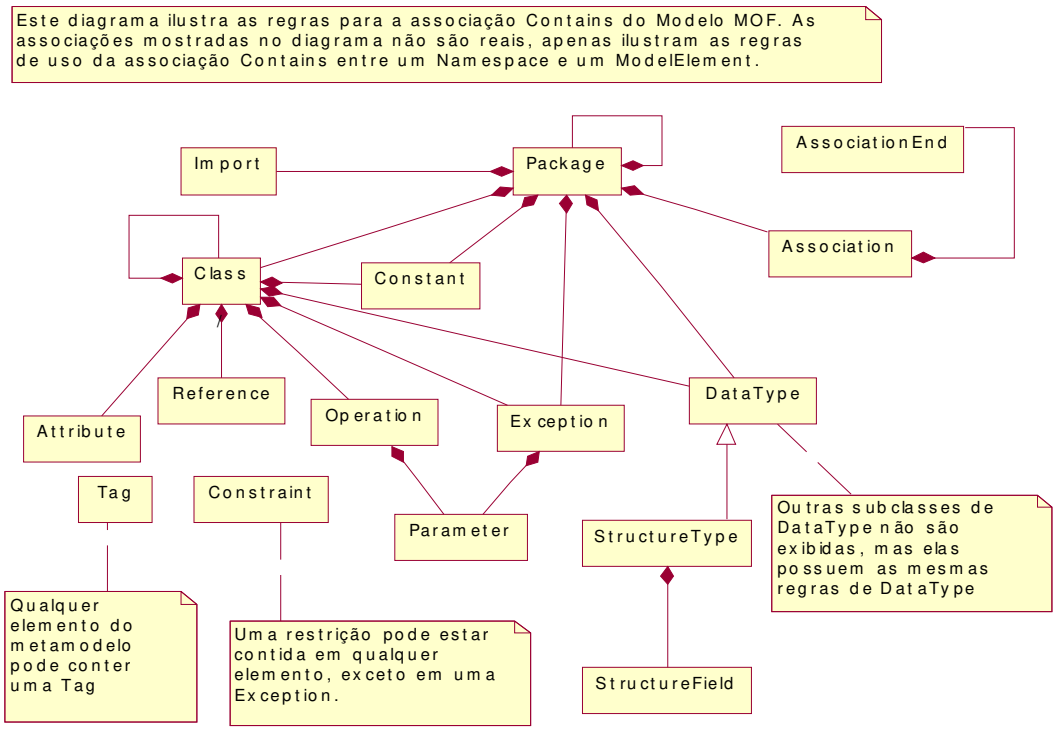

Figura 3.4 – Regras da associação *Contains*

3.4.4 Exceções do Modelo MOF

As exceções definidas no Modelo MOF são :

- ***NameNotFound***: esta exceção ocorre quando a *lookup ()* da classe *Namespace* é executada e o elemento pesquisado não é encontrado;

- ***NameNotResolved***: esta exceção ocorre quando a operação *resolveQualifiedName ()* da classe *Namespace* não resolve o nome qualificado passado como parâmetro.

3.4.5 Restrições do Modelo MOF

O Modelo MOF define um conjunto de restrições para classes, tipos de dados, operações, atributos e associações derivadas. Embora todas estas restrições estejam expressas em OCL na especificação MOF, não existe a exigência de que esta linguagem seja usada na implementação.

Tabela 3.1 – Restrição *ContentNamesMustNotCollide*

```
context Namespace

inv: self.contents.forAll(

     e1, e2 | e1.name = e2.name implies r1 = r2)
```

A Tabela 3.1 mostra a restrição *ContentNamesMustNotCollide* descrita na linguagem OCL. Esta é uma restrição imediata que assegura a unicidade de nome entre os elementos de um *namespace*.

3.4.6 *Tags* do Modelo MOF

A Tabela 3.2 relaciona algumas das *tags* padronizadas pelas especificações MOF, XMI e JMI.

Tabela 3.2 – *Tags* padronizadas

Especificação	Tag	Descrição
MOF	`org.omg.mof.idl_prefix`	É associada a um pacote e define um prefixo para o módulo a ser gerado.
JMI	`javax.jmi.packagePrefix`	É associada a um pacote e define um prefixo para as interfaces a serem geradas.
XMI	`org.omg.xmi.namespace`	Determina um XML *namespace* que por sua vez, torna-se a primeira qualificação de um pacote.

3.5 Mapeamentos dos Metamodelos MOF

O segundo objetivo da especificação MOF é definir uma infra-estrutura com interfaces padronizadas, para acesso e manipulação de um metamodelo e de seus correspondentes modelos. Para alcançar este objetivo, são definidos mapeamentos de metamodelos MOF para tecnologias de implementação, tais como interfaces Corba IDL e interfaces Java, gerando um conjunto de APIs que provê acesso uniforme e consistente aos metadados.

A Figura 3.5 apresenta os mapeamentos de um metamodelo MOF para as tecnologias Corba, Java e XMI. Um mapeamento abstrato, independente de qualquer tecnologia, também é definido na especificação MOF. Os mapeamentos concretos, ou seja, para uma tecnologia de implementação, são realizados mantendo-se um alinhamento com o mapeamento abstrato.

No mapeamento abstrato é definido o modelo computacional do nível M1. Os principais conceitos deste modelo são:

- todo objeto do nível M1 tem uma identidade intrínseca e permanente, independente de suas propriedades;

- em M2, um tipo de dado define um conjunto finito de valores. Em M1, estes valores não são considerados objetos. O mapeamento concreto define o mapeamento dos tipos em M2 para tipos da tecnologia em questão;

- um domínio de instâncias M1 é denominado extensão. A extensão de uma classe-M2 contém todas as suas instâncias. A extensão de uma associação contém todos os *links* criados. A extensão de um pacote contém as extensões das classes, associações e pacotes aninhados, agrupados e herdados.

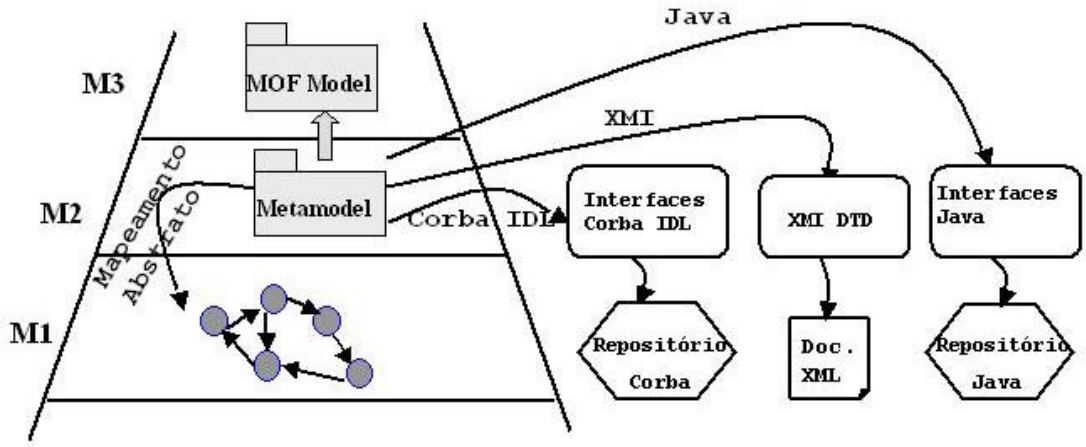

Figura 3.5 – Mapeamentos de um Metamodelo MOF

O modelo de programação proposto pelos mapeamentos para Corba e Java é composto por interfaces genéricas e interfaces específicas do metamodelo mapeado. As interfaces específicas permitem que um usuário crie, atualize e consulte instâncias (modelos) do metamodelo mapeado. As interfaces genéricas são chamadas de interfaces reflexivas, pois, além de oferecerem funcionalidades comuns a todos os metamodelos, implementam introspecção entre as camadas da arquitetura.

A especificação MOF contempla o mapeamento para interfaces Corba IDL. A especificação *Java Metadata Interface* (JMI) [JCP 2002] define o mapeamento de um metamodelo MOF para interfaces Java. As próximas seções descrevem os mapeamentos para estas tecnologias, apresentando o modelo de programação das interfaces genéricas e específicas.

3.5.1 Mapeamento para Corba IDL

O mapeamento para interfaces Corba IDL define um modelo de programação comum para manipular as instâncias de um metamodelo. Este modelo é composto pelos seguintes tipos de

meta-objetos: *Instance*, *Class Proxy*, *Association*, *Package* e *Package Factory*. A Figura 3.6 ilustra este modelo através de um exemplo.

Neste exemplo, à esquerda, no nível M2, é definido um metamodelo representado pelo pacote **P**, composto da classe **C1** e da associação **A**. No centro é apresentado o resultado do mapeamento dos elementos do metamodelo **P** para os cinco tipos de meta-objetos do nível M1. O diagrama da direita mostra os relacionamentos que existem entre estes meta-objetos.

O pacote **P** em M2 é mapeado para um objeto do tipo *Package* e um objeto do tipo *Package Factory*. *Package* é uma coleção de meta-objetos descritos pelo metamodelo e *Package Factory* é responsável pela criação do objeto *Package*. A regra de formação dos nomes destes meta-objetos é a seguinte: o nome do objeto do tipo *Package* é o nome do elemento em M2 mais o sufixo *Package* e o nome do objeto do tipo *Package Factory* é o nome do elemento em M2 mais o sufixo *PackageFactory.* Desta forma, conforme ilustrado na Figura 3.6, o pacote **P** em M2 é mapeado para os objetos PPackage e PPackageFactory em M1.

Uma classe em M2 é mapeada para um objeto do tipo *Class Proxy* e um objeto do tipo *Instance*. O Objeto *Class Proxy* é criado com o nome da classe em M2 mais o sufixo Class. O objeto *Instance* é criado com o mesmo nome da classe em M2. *Class Proxy* instancia objetos do tipo *Instance* e mantêm-nos como uma coleção (extensão). Atributos do tipo nível de classificador são definidos em um objeto *Class Proxy*. A interface de um objeto do tipo *Instance* fornece operações para destruir uma instância e manipular os atributos e referências definidas na classe-M2.

Uma associação em M2 é mapeada para um objeto do tipo *Association* que recebe o mesmo nome do elemento em M2. A interface de um objeto *Association* mantém todos os *links* instanciados em M1 e define operações para consulta, inserção, atualização e remoção de um *link* e uma operação para retornar o conjunto inteiro de *links*.

O módulo reflexivo provê a funcionalidade básica para os meta-objetos descritos acima e assegura que as operações definidas nas interfaces específicas não violam as semânticas definidas pelo meta-objeto. A especificação MOF define quatro interfaces reflexivas que são herdadas por objetos do nível M1, conforme ilustrado na Figura 3.7. Nesta figura é utilizado o metamodelo **P** descrito na Figura 3.6.

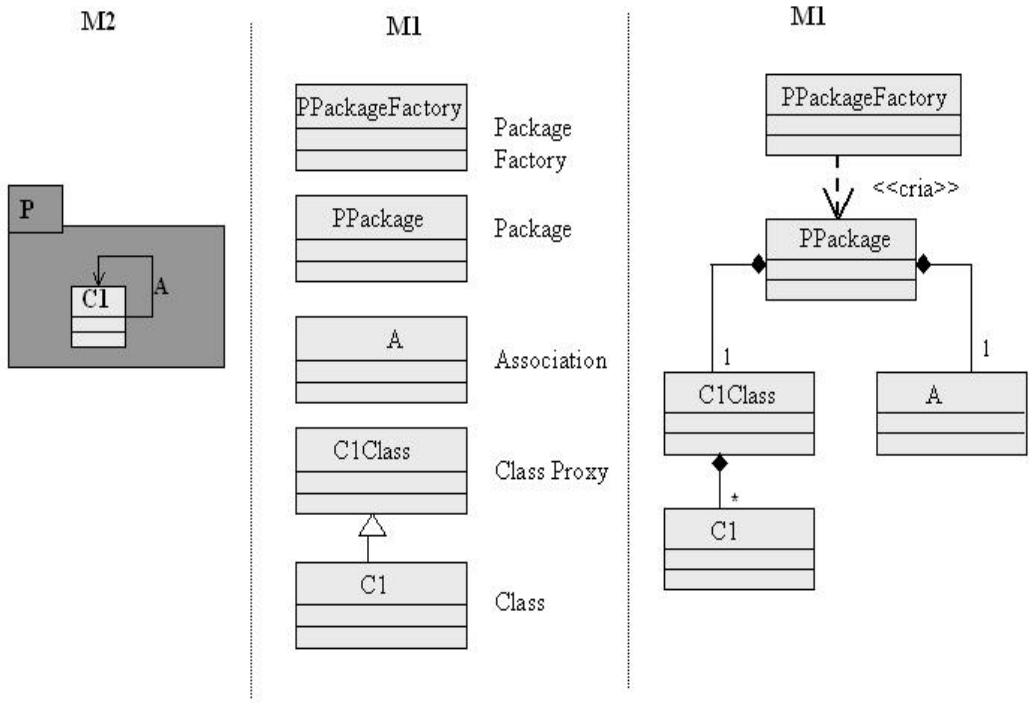

Figura 3.6 – Relacionamentos entre os níveis M1 e M2

RefBaseObject é uma interface abstrata que é indiretamente herdada por todos as outras interfaces. *RefObject*, *RefPackage* e *RefAssociation* provêem, respectivamente, as operações comuns a todos objetos do tipo *Instance*, *Package* e *Association*. A Tabela 3.3 provê uma lista parcial com as principais operações das interfaces reflexivas do mapeamento para Corba.

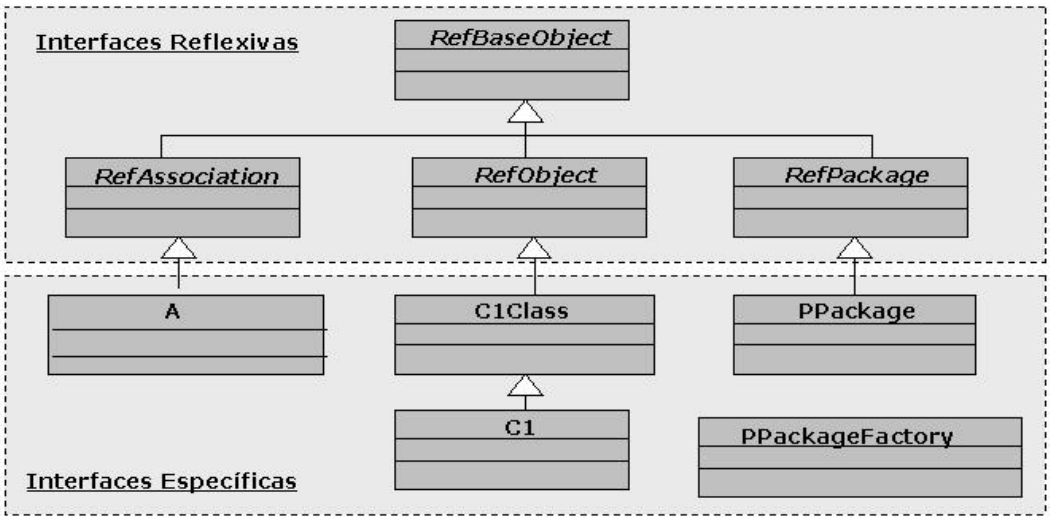

Figura 3.7 – Hierarquia de Interfaces Corba-IDL

Tabela 3.3 – Operações das interfaces reflexivas

Interface	Operação	Descrição
RefBaseObject		
	refMofId	retorna o identificador único do objeto
	refMetaObject	retorna o elemento do metamodelo que descreve o objeto
	refItself	verifica se o objeto e o objeto passado como parâmetro são o mesmo
	refImmediatePackage	retorna RefPackage referente ao pacote que contém o objeto
	refOuttermostPackage	retorna RefPackage referente ao pacote mais externo que contém o objeto
	refDelete	destrói o objeto
	refVerifyConstraints	"dispara" as restrições associadas ao objeto
RefObject		
	refIsInstanceOf	verifica se o objeto é instância de uma classe passada como parâmetro
	refCreateInstance	cria uma instância do objeto
	refAllObjects	retorna todas as instâncias cujo tipo é definido pela classe deste objeto
	refSetValue	atribui um valor a um atributo ou referência deste objeto
	refInvokeOperation	executa uma operação do metamodelo definida para este objeto
RefAssociation		
	refAllLiknks	retorna todos os *links* da associação
	refLinkExists	retorna verdadeiro se o *link* faz parte do conjunto de links desta associação
	refAddLink	inclui um *link* na lista desta associação
	refRemoveLink	remove um *link* da lista desta associação
RefPackage		
	refClassRef	retorna o objeto *Class Proxy* de uma determinada classe
	refAssociationRef	retorna o objeto *Association* de uma determinada associação
	refPackageRef	retorna o objeto *Package* de um pacote aninhado ou agrupado (*clustered*)

3.5.2 Mapeamento para Interfaces Java

Java Metadata Interface (JMI) [JCP 2002] define um modelo de programação para o gerenciamento de metadados MOF na plataforma Java. A versão atual desta especificação foi elaborada a partir da especificação MOF1.4 e as principais diferenças entre elas, além da tecnologia alvo do mapeamento, são a extinção do objeto *Package Factory* no modelo de objetos no nível M1 e as interfaces reflexivas que são mostradas na Figura 3.8 . Novamente, o metamodelo **P** definido na Figura 3.6 é utilizado como exemplo.

O modelo de programação do mapeamento para Java é composto por quatro tipos de meta-objetos: *Instance*, *Class Proxy*, *Association* e *Package*. A funcionalidade de *Instance*, *Class Proxy* e *Association* é a mesma definida no mapeamento para Corba IDL (seção 3.5.1). As funcionalidades do objeto *Package_Factory* foram incorporadas pelo objeto *Package*.

Em relação as interfaces reflexivas, JMI define uma nova hierarquia reflexiva, introduzindo as interfaces *RefClass, RefStruct*, *RefEnum*, *RefException* e *RefFeatured* e definindo as interfaces *RefClass* e *RefObject* para lidar, respectivamente, com as operações dos objetos do tipo *Class Proxy* e *Instance* do mapeamento Corba-IDL.

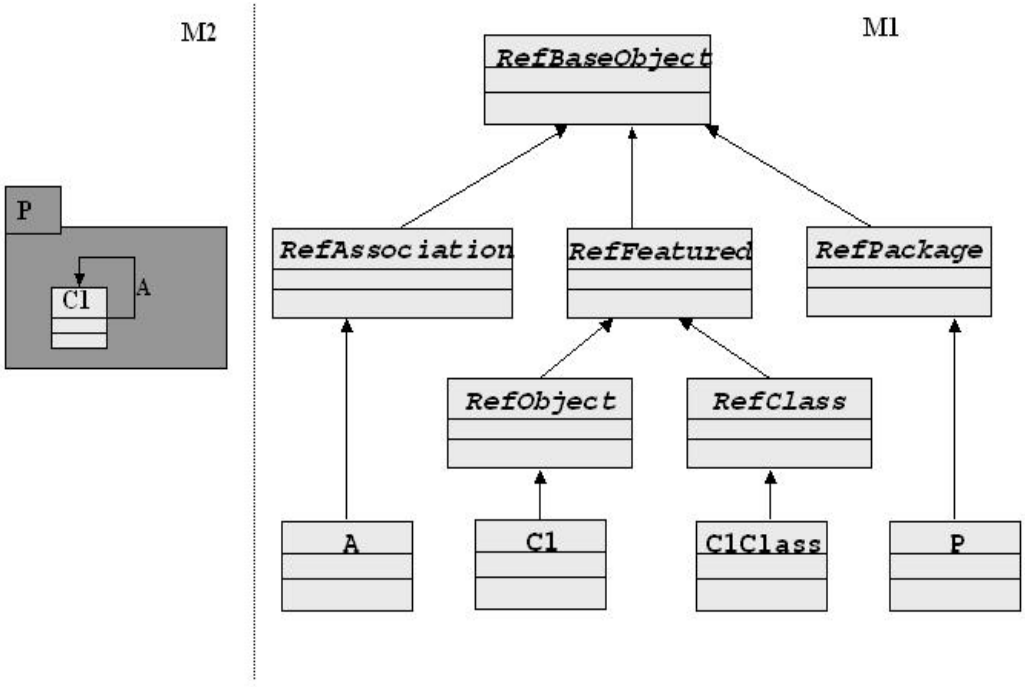

Figura 3.8 – Hierarquia de Interfaces Java

3.5.3 XML *Metadata Interchange*

A interoperabilidade definida nas especificações MOF e JMI baseia-se no uso de interfaces de programação. XML *Metadata Interchange* (XMI) [OMG 2002b, 2003c] define um mecanismo padrão para intercâmbio de metadados entre ferramentas, repositórios e *middlewares*. XMI estabelece o mapeamento MOF-XML, definindo regras e *tags* para a representação dos elementos MOF em XML. A Figura 3.9 apresenta uma classe-M1 do metamodelo UML descrita no formato XMI.

As regras do padrão XMI podem ser aplicadas a objetos e meta-objetos de qualquer nível da arquitetura de metamodelagem. É possível realizar a consistência de documentos XMI para certificar-se de que os elementos presentes no documento estão de acordo com o metamodelo que o descreve. Para esta finalidade, *Document Type Definition* (DTD) [OMG 2002b] e XML *Schema* [OMG 2003c] são utilizados para validar documentos XMI.

É importante citar que a especificação MOF contém o XML DTD correspondente ao Modelo MOF, o que torna possível o intercâmbio de metamodelos MOF de modo análogo ao intercâmbio de modelos.


```
   <UML:Model xmi.id = 'lsm:1d1bb4d:fe8c59e1c4:-7ffa' name = 'model 1' isSpecification = 'false`
isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
     <UML:Namespace.ownedElement>
       <UML:Class xmi.id = 'lsm:1d1bb4d:fe8c59e1c4:-7ff9' name = 'Empresa' visibility = 'public`
isSpecification = 'false' isRoot = 'false' isLeaf = 'false' isAbstract = 'false`  isActive = 'false'>
         <UML:Classifier.feature>
           <UML:Attribute xmi.id = 'lsm:1d1bb4d:fe8c59e1c4:-7ff8' name = 'nome' visibility = 'public`
isSpecification = 'false' ownerScope = 'instance'>
             <UML:Attribute.initialValue>
               <UML:Expression xmi.id = 'lsm:1d1bb4d:fe8c59e1c4:-7fa7' language = 'java`  body = ''/>
             </UML:Attribute.initialValue>
             <UML:StructuralFeature.type>
               <UML:Class xmi.idref = 'lsm:1d1bb4d:fe8c59e1c4:-7ff7'/>
             </UML:StructuralFeature.type>
           </UML:Attribute>
           <UML:Attribute xmi.id = 'lsm:1d1bb4d:fe8c59e1c4:-7ff6' name = 'numEmpregados`  visibility =
'public' isSpecification = 'false' ownerScope = 'instance'>
             <UML:StructuralFeature.type>
               <UML:Class xmi.idref = 'lsm:1d1bb4d:fe8c59e1c4:-7ff5'/>
 </UML:StructuralFeature.type>
           </UML:Attribute>
           <UML:Operation xmi.id = 'lsm:1d1bb4d:fe8c59e1c4:-7ff4' name = 'precoAcao`  visibility =
'public' isSpecification = 'false' ownerScope = 'instance'
             isQuery = 'false' concurrency = 'sequential' isRoot = 'false' isLeaf = 'false`  isAbstract = 'false'>
<UML:BehavioralFeature.parameter>
```

Figura 3.9 – Mapeamento de uma classe UML para XMI

3.5.4 *Human Usable Text Notation*

A especificação *Human Usable Text Notation* (HUTN) [OMG 2002c] foi adotada pelo consórcio OMG no final de 2002. É uma tecnologia similar a XMI (produtor-consumidor). Mas, ao contrário de XMI que é mais apropriada para intercâmbio de metadado entre máquinas, HUTN foi projetada tendo o usuário como objetivo. Assim como XMI realiza o intercâmbio de metadados serializando instâncias de um metamodelo usando o formato XML, HUTN define uma linguagem textual semelhante a Java para serializar e transferir modelos. HUTN foi desenvolvida com os seguintes princípios básicos para a geração da linguagem: (i) usabilidade; (ii) uso do conjunto de caracteres ASCII; e (iii) geração automática da linguagem sem intervenção humana.

Em [Steel 2001] é descrito um sistema que gera automaticamente os módulos produtor e consumidor de um modelo descrito na notação HUTN. A Figura 3.10 ilustra os principais componentes deste sistema que são descritos em seguida.

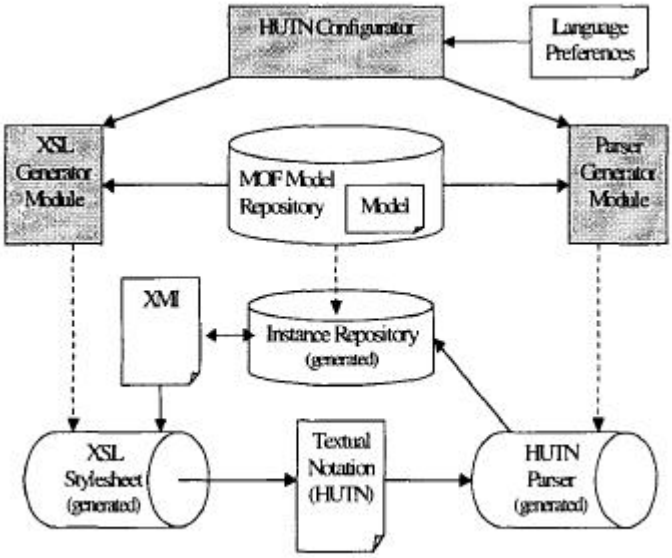

Figura 3.10 – Componentes de um sistema HUTN [Steel 2001]

Os componentes novos deste sistema são mostrados nos retângulos cinzas. *MOF Model Repository* representa o repositório de metamodelos e *Instance Repository* representa um repositório gerado automaticamente para as instâncias de um metamodelo. O *módulo HUTN*

Configurator armazena configurações definidas pelo usuário para a linguagem a ser gerada. Estas configurações são passadas para os módulos *XSL Generator* e *Parser Generator*, componentes centrais deste sistema. *XSL Generator* cria um *XSL Style Sheet* que fornece o mapeamento de um modelo descrito no formato XMI para a notação HUTN. *Parser Generator* tem duas funções principais: (i) geração de uma linguagem que reconheça um modelo descrito na notação HUTN; e (ii) criação dos meta-objetos correspondentes ao modelo no repositório de instâncias.

A Figura 3.11 mostra a classe UML da Figura 3.9 descrita na notação HUTN. Os principais benefícios obtidos com a utilização do padrão HUTN são:

- **consistência**: a linguagem HUTN gerada para cada metamodelo é diferente, mas todas elas utilizam a mesma estrutura e estilo;
- **automatização**: além da geração automática da linguagem, as transformações HUTN-MOF, HUTN-XMI e XMI-HUTN podem ser realizadas através da geração automática de *parsers*;
- **completude**: qualquer metamodelo descrito em MOF pode ter uma linguagem HUTN.

```
Class Empresa {
        name = "Empresa";
        feature:
          Attribute nome {
              name = "nome";
              type:
                    ref  DataType String;
           } ;
           Attribute numEmpregados {
                 name = "numEmpregados ";
                 type:
                     ref   DataType Integer;
           } ;
           Operation precoAcao {
                 name = "precoAcao";
                 parameter:
                   Parameter  return {
                         kind = "return";
                         type:
                            ref  DataType Real;
                  } ;
        } ;
} ;
```

Figura 3.11 – Um classe UML descrita na notação HUTN

3.6 MOF 2.0

No início de 2002, o consórcio OMG iniciou os trabalhos para a elaboração da especificação MOF 2.0 com o objetivo principal de definir um *framework* para uma nova geração de metadados independentes de plataforma. As funcionalidades acrescentadas a esta nova versão visam principalmente: (i) prover uma infra-estrutura mais adequada para a Arquitetura Orientada a Modelos (MDA); (ii) alinhar o meta-metamodelo MOF com o núcleo da nova especificação UML 2.0; e (iii) incluir funções relacionadas ao gerenciamento de metadados e que não foram contempladas nas versões anteriores de MOF.

Uma das críticas mais constante a versão corrente da especificação MOF é a falta de um alinhamento perfeito com UML. Embora compartilhem muitos conceitos e construções de modelagem, é comum às especificações do consórcio OMG (por exemplo, EDOC e SPEM), a definição do metamodelo em MOF e como um perfil UML. Nos trabalhos de elaboração de MOF2 e UML2, o consórcio OMG preocupou-se em unificar os conceitos de modelagem que se traduz no reuso, por parte de MOF2, da infra-estrutura central de UML2, conforme mostrado na Figura 3.12. Os benefícios deste alinhamento são [OMG2003e]:

- um conjunto único e simples de conceitos de modelagem de metadados;
- mapeamentos de MOF para uma tecnologia (por exemplo, JMI e XMI) também se aplicam a UML;
- ferramentas UML podem ser usadas para a modelagem de metadados;

No momento, os grupos de trabalhos de MOF 2.0 produziram versões iniciais de documentos que apresentam propostas para as seguintes funcionalidades: (i) núcleo MOF, (ii) versão e ciclo de vida do desenvolvimento, (iii) consulta (*query*) a elementos de um modelo, transformação e *view*, e (iv) gerenciamento de uma facilidade MOF e definição de operações relacionadas ao ciclo de vida de objetos. As linguagens abstratas definidas para as novas funcionalidades devem ser expressas através do meta-metamodelo MOF 2.0, ou seja, definidas como metamodelos MOF.

O princípio básico de ortogonalidade permeia as novas funcionalidades de MOF 2.0, enfatizando a necessidade de separar, sempre que possível, os modelos dos serviços (ou utilitários) que se aplicam a eles. As especificações MOF 1.x foram fortemente influenciadas pelas semânticas e ciclo de vida dos repositórios de metadados de Corba. O objetivo deste princípio é separar os conceitos de modelagem dos serviços de metadados, permitindo que soluções que utilizem a tecnologia MOF, implementem apenas as funcionalidades necessárias para as suas necessidades. As novas funcionalidades de MOF2 são descritas nas próximas seções.

3.6.1 Núcleo MOF

Núcleo MOF [OMG 2003e] é o termo utilizado pelo consórcio OMG para os pacotes que compõem o Modelo MOF 2.0, ilustrado na Figura 3.12.

Na Figura 3.12, o pacote *Core* pertence à infra-estrutura UML 2.0 e os pacotes *Basic* e *Constructs* provêem conceitos básicos e construções de modelagem para o Modelo MOF 2.0. O Modelo MOF 2.0 é composto por dois pacotes principais: *Essential MOF* (EMOF) e *Complete MO*F (CMOF). As características reflexão, extensibilidade e identidade são definidas em pacotes separados: (i) *reflection* provê mecanismos para a descoberta e manipulação de meta-objetos; (ii) *identity* define as características do conceito de identidade de um objeto, propriedade relevante para as funcionalidades de transformação de modelo, consulta, versão e outras; e (iii) *extension* provê o mecanismo de extensibilidade de um metamodelo. Ao contrário de MOF1 que definiu estas características nos mapeamentos para Corba e Java, MOF2 define reflexão, ciclo de vida e identidade como parte do núcleo MOF, separando, bem claramente, conceitos de modelagem de sua implementação.

O objetivo principal do pacote EMOF é permitir a definição de metamodelos simples através de um conjunto mínimo de construções e, ao mesmo tempo, oferecer mecanismos de extensão para que metamodelos mais complexos sejam definidos usando CMOF. A motivação deste objetivo, é diminuir a complexidade de ferramentas que suportem MDA. CMOF é o meta-metamodelo usado para descrever outros metamodelos tais como UML 2.0 e CWM2. O pacote CMOF não define novas classes, apenas utiliza as definições dos pacotes EMOF e Core de UML 2.0 para definir suas capacidades de metamodelagem.

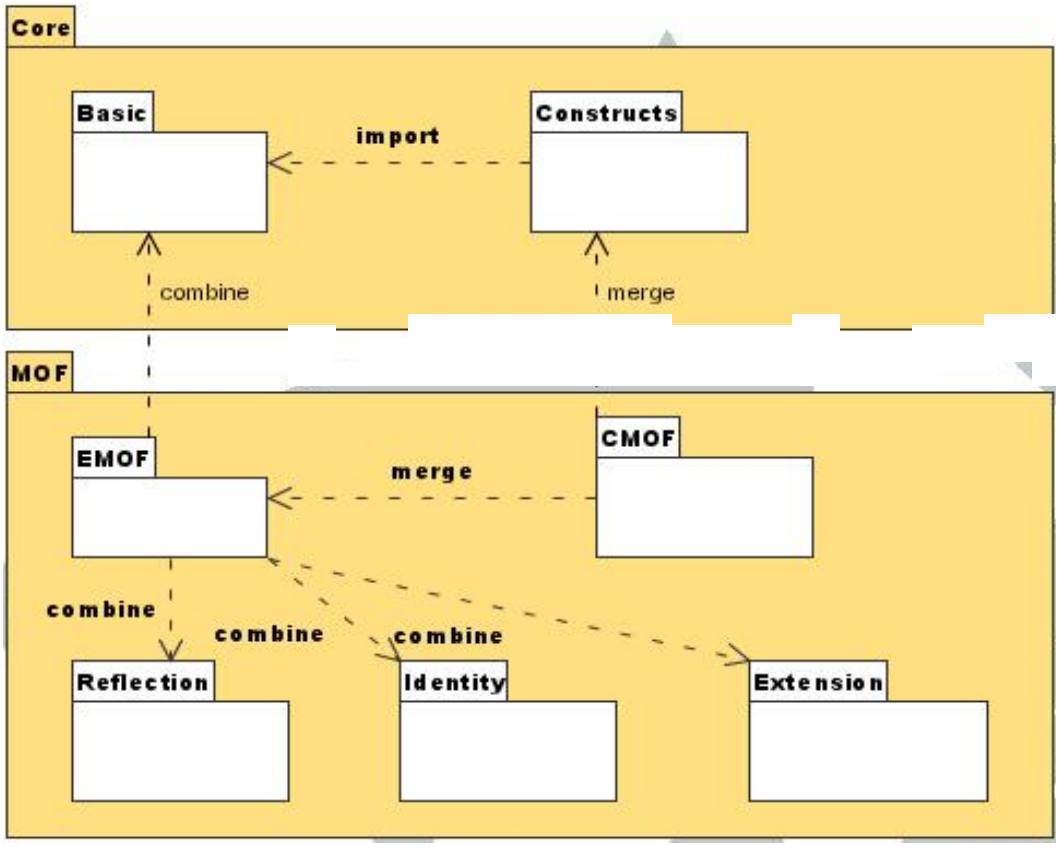

Figura 3.12 – Diagrama de Pacotes de MOF 2.0

A Figura 3.12 ilustra os mecanismos de extensão de MOF2. Eles são os seguintes:

- ***importing***: como em MOF1, os elementos do pacote importado são visíveis para o pacote que o importa;
- ***combining***: um novo pacote é definido, contendo elementos do pacote combinado e do que o combina;
- ***merging***: o pacote que realiza o *merge* (*merging package*) estende com novas característica elementos – com o mesmo nome e o mesmo tipo – do pacote externo (*merged package*). Este mecanismo é usado quando elementos com o mesmo nome representam o mesmo conceito, independentemente dos pacotes onde foram definidos, mas deseja-se evitar o acoplamento dos pacotes com o mecanismo de herança. Não é definido um novo pacote mas é criada uma dependência entre eles.

3.6.2 Mecanismos para gerenciamento de metadados

Na especificação MOF 2.0 deverão ser definidos os mecanismos para gerenciamento de versões de metadados em um repositório MOF considerando os seguintes aspectos [OMG 2004d]: *branching*, *merging*, configuração e contexto.

3.6.3 Mecanismos de consulta

Mecanismos de consulta, *view* e transformação serão definidos em MOF 2.0 [OMG 2004b]. Uma consulta em modelos MOF é necessária para selecionar elementos de um modelo através de uma consulta ad-hoc como também, para selecionar os elementos que serão utilizados em uma transformação. Uma *view* representa a derivação de um modelo em outro, revelando aspectos específicos. Uma transformação define os relacionamentos necessários para transformar um metamodelo MOF origem em um metamodelo MOF destino. Os relacionamentos devem ser expressos de maneira que a transformação possa ser automatizada.

3.6.4 Facilidade MOF e Suporte ao Ciclo de Vida de Metadados

Uma facilidade MOF oferece serviços de metadados para aplicações clientes [OMG 2004c]. Aspectos genéricos de serviços e funções tais como conexão, localização, transação, controle de acesso e operações para criação e destruição de objetos devem ser definidas em MOF 2.0.

Mecanismos consistentes devem suportar o ciclo de vida de desenvolvimento, permitindo determinar o estado de uma versão de um modelo. O ciclo de vida de desenvolvimento (por exemplo, aprovado, testado, obsoleto e outros) será importante para a especificação de contextos e para o controle de operações que poderão ser aplicadas a um modelo como por exemplo, proibir alterações em um modelo obsoleto.

3.7 Considerações Finais

Este capítulo apresentou detalhadamente a especificação MOF, tecnologia central deste trabalho. As especificações JMI, XMI e HUTN também foram apresentadas, mas maior ênfase foi dada a arquitetura de modelagem adotada pelo consórcio OMG, a linguagem (Modelo MOF)

especificada e ao mapeamento de um metamodelo para a tecnologia Corba. A descrição da especificação MOF em detalhes é uma contribuição parcial desta dissertação. Por último, apresentou-se sucintamente as principais funcionalidades a serem acrescentadas à versão 2 de MOF.

4 GRM – GERENCIADOR DE UM REPOSITÓRIO MOF

4.1 Introdução

No capítulo anterior, foi apresentada a abordagem do consórcio OMG para o gerenciamento de metadados em um ambiente distribuído. Conforme apresentado, a especificação MOF, juntamente com as especificações JMI e XMI, define um *framework* de modelagem aberto e extensível.

O propósito deste capítulo é apresentar a especificação, a arquitetura e as ferramentas que compõem o sistema GRM, um sistema de gerenciamento de metadados baseado nas tecnologias MOF, JMI e XMI.

Na seção 4.2 são apresentados aspectos do projeto do sistema. Os objetivos, casos de uso, requisitos e a arquitetura do sistema GRM são introduzidos. As seções 4.3 e 4.4 apresentam detalhes da implementação do sistema GRM. Na seção 4.3 é apresentado o software aberto que implementa parte das funcionalidades do sistema GRM. Conceitos introduzidos por este software e os principais módulos são descritos. Na seção 4.4, é descrito com detalhes o compilador MODL, desenvolvido como contribuição específica do trabalho da tese.

4.2 Especificação do Sistema GRM

4.2.1 Objetivo e Casos de Uso

O objetivo do sistema de repositório GRM é definir e manipular metamodelos MOF, oferecendo um ambiente integrado composto de várias ferramentas com o propósito de promover a prática de metamodelagem.

A Figura 4.1 apresenta um diagrama UML de casos de uso que ilustram a utilização do sistema GRM. O meta-modelador é um especialista em definir modelos de um domínio de aplicação. Ele utiliza o sistema GRM para definir, visualizar, consultar e validar metamodelos. O processo de criação de um metamodelo pode ser feito em etapas e o sistema permite que versões não validadas sejam armazenadas. Em um ambiente federativo, metamodelos definidos em outros repositórios MOF podem ser importados para o repositório de trabalho do meta-modelador e,

vice-versa, metamodelos podem ser exportados para outros repositórios MOF. Uma vez finalizado o metamodelo, o meta-modelador pode acionar o sistema GRM para que seja gerado automaticamente o código do software servidor de um repositório de modelos deste metamodelo. O código gerado é incompleto, embora implemente todas as interfaces padronizadas (Corba IDL ou Java) de acesso aos modelos. O código para as operações definidas no metamodelo e funcionalidades adicionais devem ser realizadas pelo implementador, que deve também implementar aplicações clientes que comporão o sistema de repositório do metamodelo (por exemplo, um editor de modelos).

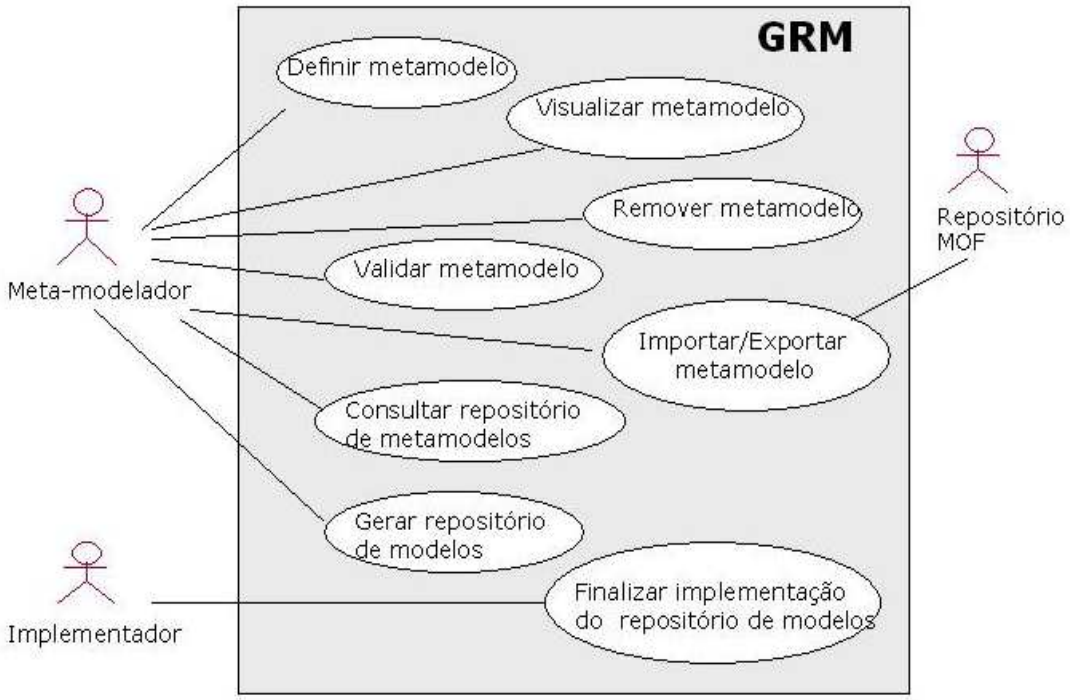

Figura 4.1 – Diagrama de casos de uso do sistema GRM

4.2.2 Requisitos

Os requisitos funcionais levantados a partir dos casos de uso e os requisitos não funcionais do sistema GRM são apresentados e analisados a seguir.

4.2.2.1 Requisitos Funcionais

- Definir metamodelos MOF:

  O meta-modelador utiliza um editor gráfico e/ou uma linguagem textual para definir metamodelos. O Modelo MOF versão 1.4 é o meta-metamodelo do sistema GRM.

- Armazenar metamodelos:

  Metamodelos podem ser armazenados em um ou mais mecanismos de persistência, conservando o estado destes entre as execuções do sistema.

- Remover um metamodelo:

  Quando um metamodelo é removido do sistema não são verificadas as dependências em relação a outros metamodelos. No caso de inconsistência, estas devem aparecer no momento de validação ou geração de código.

- Validar um metamodelo:

  Um metamodelo pode ser armazenado no repositório mesmo que não esteja em conformidade com as semânticas e as restrições do Modelo MOF. Porém, a qualquer momento, o usuário pode validá-lo e, neste caso, as semânticas e restrições definidas são verificadas. Um interpretador OCL pode ser utilizado para validar as expressões definidas nesta linguagem ou, caso não exista esta ferramenta, o sistema GRM deve implementar o código que as valide.

- Importar ou exportar um metamodelo no formato XMI:

  Para a validação do metamodelo importado, deve ser utilizado o documento XML DTD do Modelo MOF, disponibilizado juntamente com a especificação MOF.

- Consultar o repositório:

  O meta-modelador pode executar as seguintes consultas pré-definidas:

  - ✓ pesquisar um elemento do metamodelo pelo nome;
  - ✓ pesquisar um elemento do metamodelo pelo tipo;

- ✓ pesquisar os supertipos indiretos de um elemento do metamodelo;
- ✓ dado um elemento de um metamodelo, pesquisar os elementos que dependem dele (associação *depends on*).

- Visualizar um metamodelo:

  Um metamodelo pode ser visualizado na linguagem HTML e nos formatos XMI e HUTN.

- Gerar a implementação do software servidor de um repositório de modelos:

  Para um metamodelo completo e validado, pode-se gerar automaticamente o código do software servidor de um repositório para armazenar modelos deste metamodelo. O sistema GRM deve gerar código para as plataformas Corba e Java.

- Criar um ambiente integrado para as funções do sistema:

  O sistema GRM deve oferecer um ambiente integrado com facilidades para o acesso e manipulação do repositório de metamodelos.

- Suportar as funções de um sistema de repositório

  Exceto pela função "Gerência de Relacionamentos" que é inerente ao Modelo MOF, o suporte às funções típicas de um sistema de repositório, apresentadas na seção 2.2.6, são desejáveis no sistema GRM.

4.2.2.2 Requisitos não-funcionais

- Portabilidade. O sistema GRM pode ser executado em diferentes plataformas.
- Ortogonalidade. Uma funcionalidade deve ser aplicável a todos os metamodelos.

4.2.3 Arquitetura

A Figura 4.2 apresenta a arquitetura do sistema GRM, com componentes definidos nos moldes da arquitetura de repositório apresentada no capítulo 2, mas, diferentemente daquela, a

arquitetura proposta é um sistema distribuído, onde o acesso ao repositório é feito através de serviços.

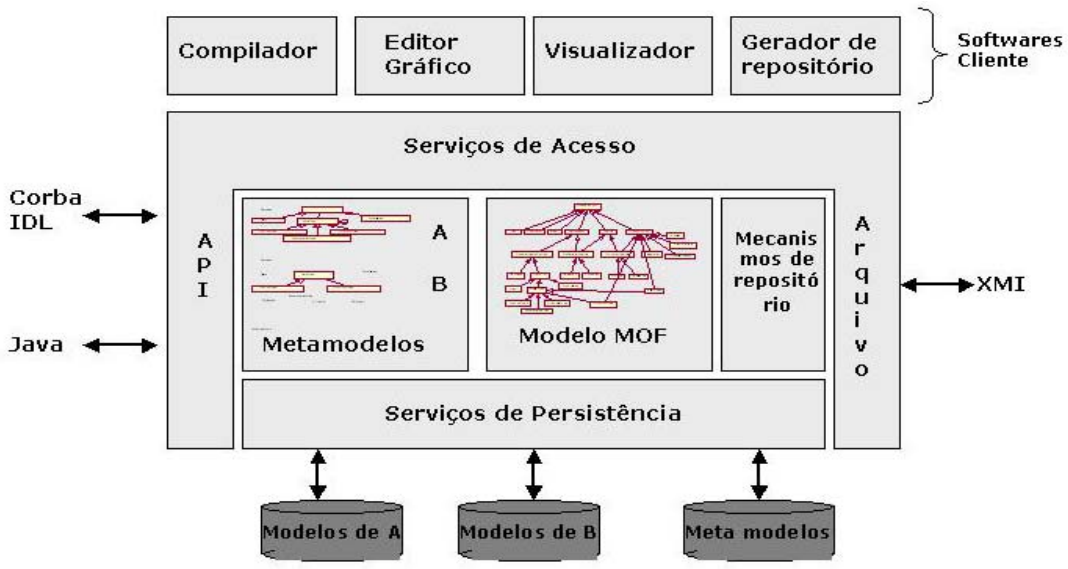

Figura 4.2 – Arquitetura do sistema GRM

Os principais componentes da arquitetura do sistema GRM são os seguintes:

- **Serviços de Persistência**: o armazenamento de metamodelos e seus respectivos modelos é gerenciado pelos serviços de persistência que provêem transparência em relação à tecnologia (base relacional, arquivo de sistema, etc) utilizada para o armazenamento e a possibilidade de distribuição dos modelos e metamodelos.
- **Modelo MOF**: é o componente central desta arquitetura e corresponde a linguagem abstrata definida na especificação MOF para a definição de metamodelos.
- **Metamodelos**: correspondem aos diferentes sistemas de gerenciamento de modelos gerados pelo sistema GRM.
- **Mecanismos de Repositório**: correspondem as funções de repositório, tais como, *check-in*/*check-out*, controle de versão, etc. Parte destas funções pode ser realizada juntamente com os serviços de persistência.
- **Serviços de Acesso**: representam (i) as interfaces padronizadas Java e Corba IDL, que permitem o acesso às construções do Modelo MOF e metamodelos e (ii) os

mecanismos para importação e exportação de metamodelos e modelos no formato XMI.

- **Software Cliente**: um conjunto de ferramentas que completa o sistema GRM. Editor gráfico, compilador, visualizadores e geradores de artefatos são ferramentas que facilitam a definição e manipulação de metamodelos.

4.3 Implementação do Sistema GRM – o Software CIM

Para o desenvolvimento do sistema GRM está sendo utilizado o software CIM — *Complex Information Management* [Unisys] — desenvolvido pela Unisys Corporation.

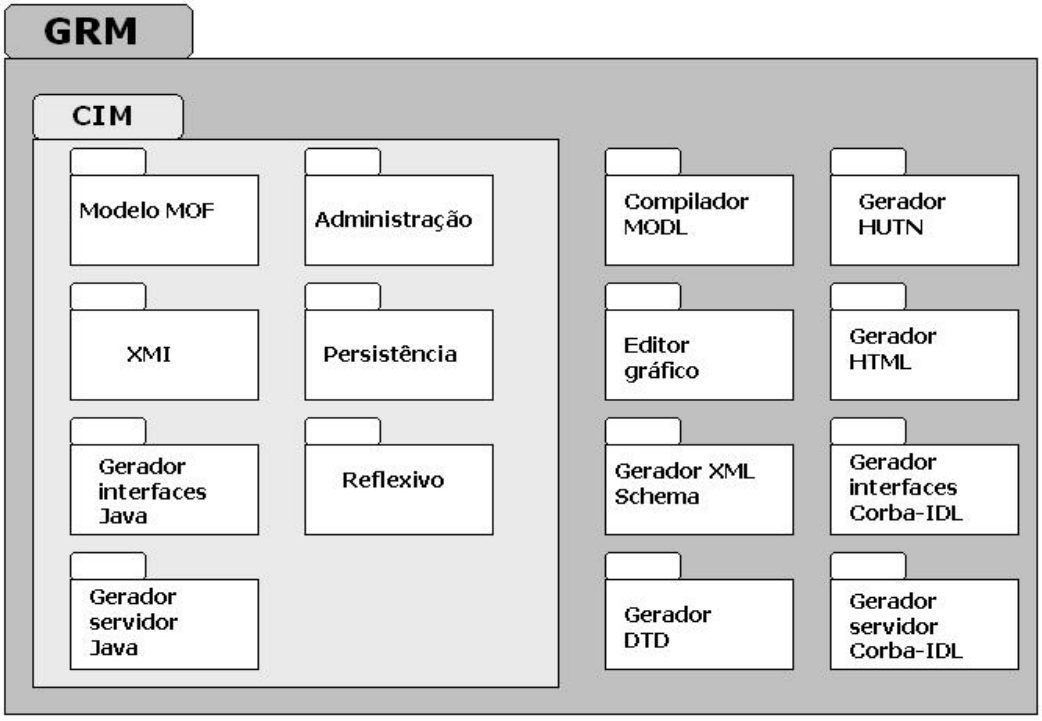

Figura 4.3 – Estruturação do sistema GRM

CIM é a implementação Java de uma infra-estrutura de metadados baseada nas especificações MOF 1.4 e JMI e tem por objetivo principal, o desenvolvimento de ferramentas e aplicações orientadas a modelos. A estruturação do sistema GRM é apresentada no diagrama de pacotes UML ilustrado na Figura 4.3, onde as funcionalidades fornecidas pelo software CIM estão agrupadas no pacote CIM e as restantes estão em desenvolvimento.

As especificações MOF e JMI definem apenas as APIs para o acesso e a manipulação de metadados. Alguns aspectos de implementação devem ser definidos pelo implementador da especificação e por isso o software CIM introduz os seguintes conceitos:

- **Facilidade**: corresponde a uma unidade de armazenamento de metadados. Uma facilidade provê o conjunto de serviços para acessar os metadados.
- **Repositório**: é a implementação do conceito MOF de *outermost package*, pacote mais externo que contém os metadados e também, o nível mais alto de interoperabilidade definido na especificação.
- **Servidor de Metadados**: é a implementação das interfaces Java de um determinado metamodelo. Uma facilidade pode ser composta por diferentes metamodelos e para cada um deles pode ser gerado um servidor de metadados.

A interface reflexiva RefBaseObject, herdada por todos os meta-objetos definidos em CIM, define os seguintes atributos:

- **Enclosure**: implementa o mecanismo de extensão descrito na seção 3.5.
- **State**: indica se um meta-obejto está carregado na memória (*active*) ou se ainda se encontra no mecanismo de persistência (*passive*);
- **MofID**: identificador único de um meta-objeto.

A Figura 4.4 mostra a interface gráfica da ferramenta administrativa do GRM. Os conceitos Facilidade, Repositório e Servidor de Metadados estão exemplificados nesta figura. Uma nova Facilidade é criada com o Modelo MOF (composto pelos pacotes *Model*, *Reflective* e *PrimitiveTypes*) e o pacote Java com.unisys.jmi.mof.ModelServer que corresponde ao software servidor com as APIs de acesso aos elementos dos pacotes Modelo MOF.

4.4 Compilador MODL

No sistema GRM decidiu-se pelo uso de uma linguagem textual para descrever um metamodelo MOF, por ser uma alternativa mais abrangente, independente da notação gráfica de

UML e, consequentemente, de um editor UML. Um editor gráfico é uma ferramenta poderosa e intuitiva para a técnica de modelagem. No entanto, para o caso de um modelo grande e complexo, a notação pode tornar-se confusa e congestionada.

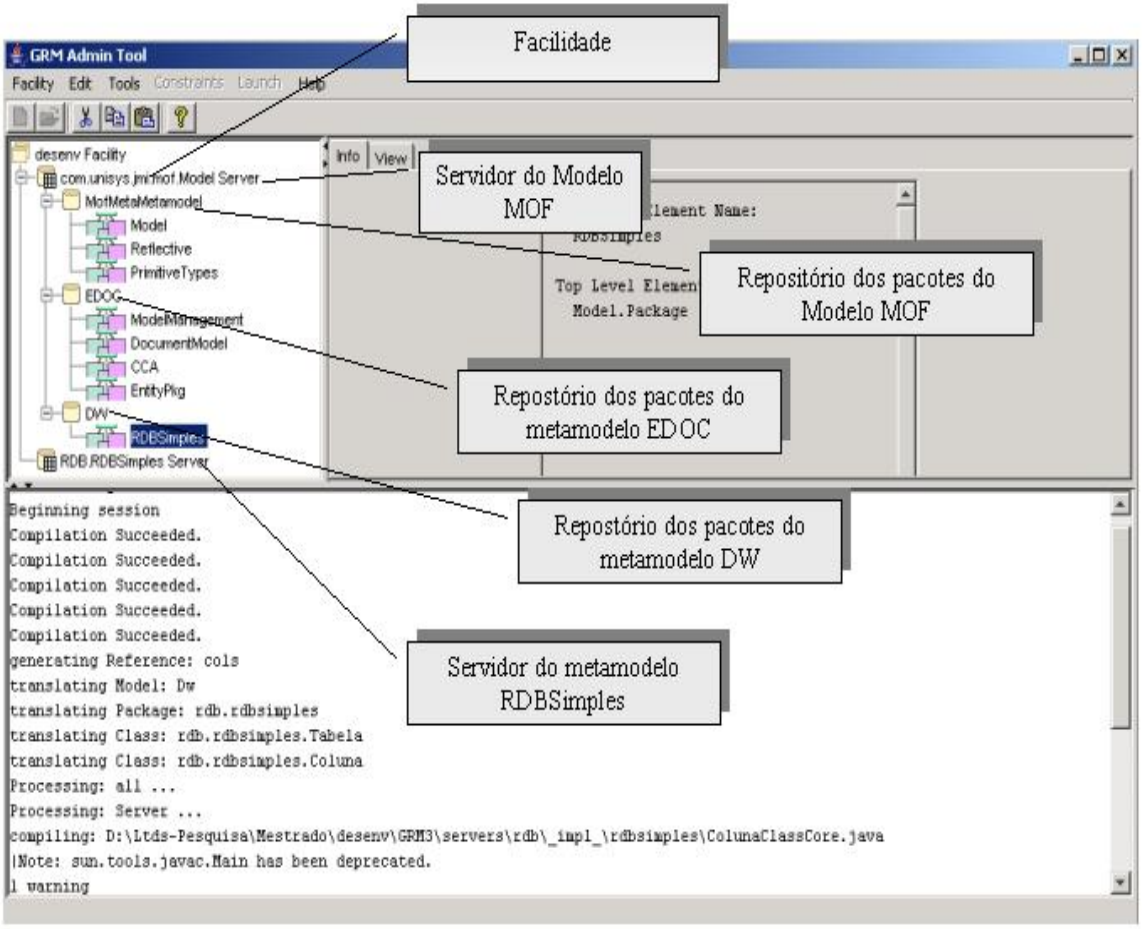

Figura 4.4 – Ferramenta administrativa do sistema GRM

O software CIM oferece um único mecanismo para a inserção de metamodelos em seu repositório: a importação no formato XMI. Esta abordagem é problemática devido à dificuldade de se editar um arquivo XMI, principalmente, quando um metamodelo é uma especialização de outro ou, através dos mecanismo de importação ou *clustering*, utiliza construções definidas em outro pacote. Nestes casos, todos os pacotes externos devem, totalmente expressos em XMI, estar presentes no mesmo arquivo de importação. A Figura 4.5 ilustra este caso, apresentando na notação UML, um exemplo de dependência entre metamodelos (metamodelo C importa

construções dos metamodelos A e B) e, como seriam os arquivos XMI e MODL com a descrição do metamodelo C.

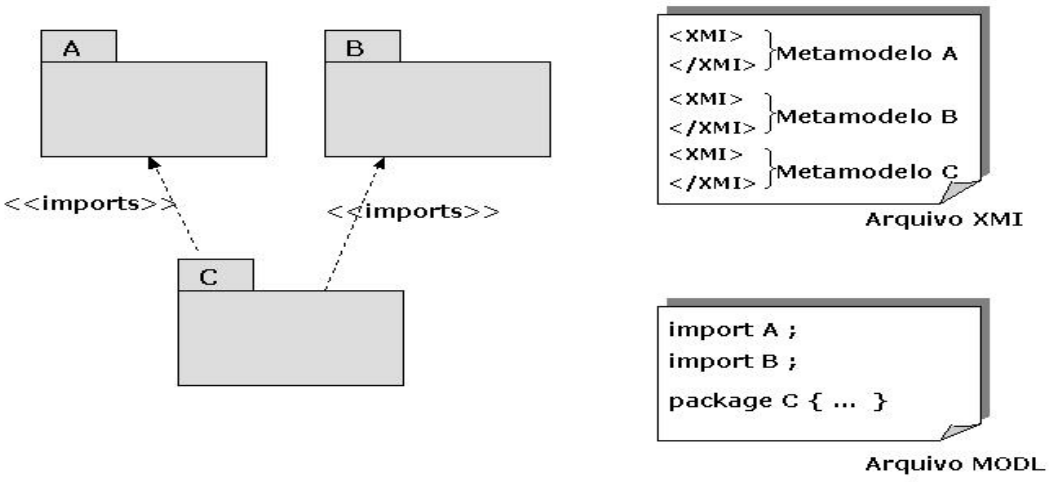

Figura 4.5 – Descrição em XMI e em MODL de um metamodelo com dependências externas

4.4.1 Meta-Object Definition Language

Meta-Object Definition Language (MODL) [DSTC 2001] é a linguagem adotada para a definição de metamodelos em GRM, embora não se trate de uma norma. Esta linguagem foi definida para a versão 1.3 de MOF e alterações foram feitas para suportar os novos tipos de dados primitivos definidos na versão MOF 1.4.

MODL é uma linguagem textual cuja sintaxe é baseada em Corba IDL. Conforme mostrado na Tabela 4.1, as construções da linguagem MODL correspondem às construções do Modelo MOF. A apresentação das principais construções de MODL é feita a seguir através da descrição do metamodelo RDBSimples apresentado na Tabela 4.1.

Tabela 4.1 – Pacote RDBSimples descrito em MODL

```
package RDBSimples {

  class Tabela  {
    attribute string nome;
    reference cols to coluna of  PossuiColunas ;
  };

  class Coluna {
     attribute string nome;
     attribute string tipo;
```

```
  };

  association PossuiColunas {
        end single Tabela tabela ;
        composite end set [0..*] of Coluna coluna ;
  };
} ;
```

O metamodelo RDBSimples contém as classes Tabela e Coluna e a associação PossuiColunas. A classe Tabela possui o atributo nome do tipo *string*. A classe Coluna tem os atributos nome e tipo, ambos do tipo *string*. A associação PossuiColunas estabelece que uma Tabela é composta por zero ou mais colunas (papel coluna) e que uma Coluna pode pertencer a apenas uma Tabela (papel tabela). A classe Tabela contém a referência cols que permite acesso direto as suas colunas. *Reference* é exemplo de uma construção do Modelo MOF que não existe na notação UML, mas é suportada pela linguagem MODL.

4.4.2 Geração do Compilador MODL

Para a implementação do compilador MODL foi utilizada a ferramenta JavaCC [JavaCC] que produz um compilador LL(k). JavaCC foi escolhida por possuir um módulo integrado (JJTree) para análise léxica, enquanto outras ferramentas requerem pacotes externos. Deste modo, JavaCC gera as regras sintáticas juntamente com as regras léxicas. A Figura 4.6 ilustra a geração dos módulos para análise léxica e sintática, utilizando-se JavaCC.

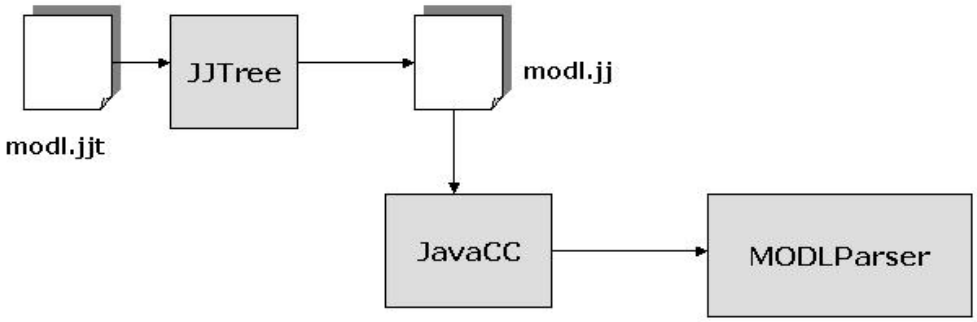

Figura 4.6 – Geração do compilador MODL

JJTree é um pré-processador que tem como entrada um arquivo com as regras de produção de uma linguagem expressa na notação EBNF. JJTree processa o arquivo de entrada (extensão

.jjt) e acrescenta informações para a geração de uma árvore sintática no compilador a ser gerado por JavaCC. O arquivo de saída de JJTree tem a extensão .jj e é o arquivo de entrada de JavaCC.

JavaCC gera um conjunto de classes Java que implementam os analisadores léxico e sintático da linguagem MODL. A Figura 4.7 ilustra as principais classes geradas. MODLParserTokenManager é a classe que realiza a análise léxica do compilador, Token implementa uma unidade léxica do metamodelo descrito no arquivo de entrada, MODLParser é o analisador sintático e a interface MODLParserConstants associa um *token* a um nome simbólico. A classe SimpleNode implementa a interface Node que define um nó da árvore sintática.

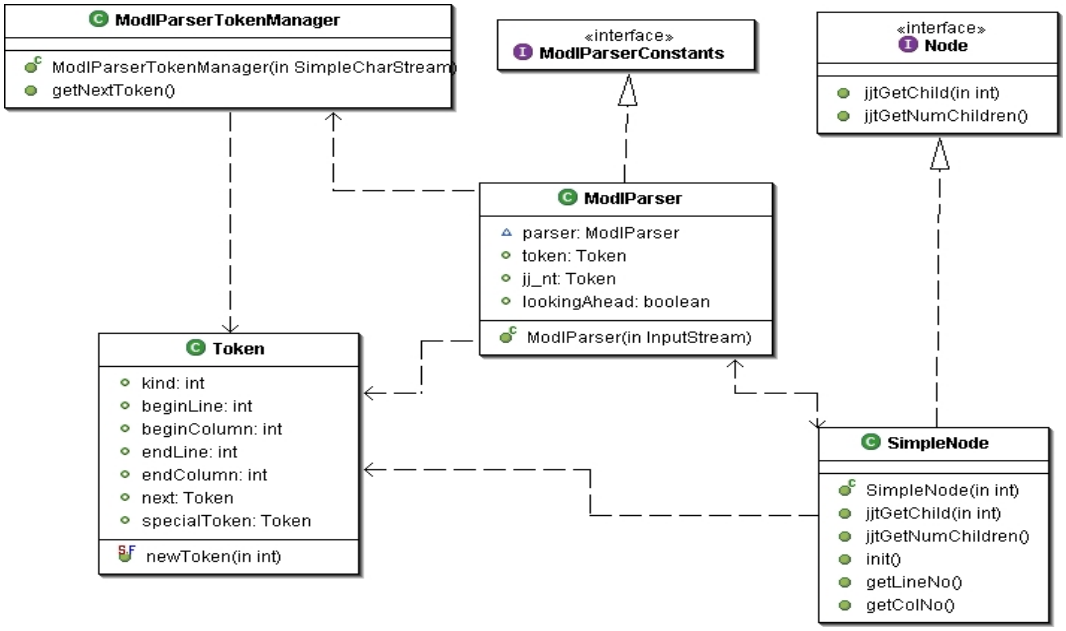

Figura 4.7 – Classes dos analisadores léxico e sintático

4.4.3 Análise Semântica e Geração de Meta-Objetos

Após as análises léxica e sintática, que correspondem ao passo 1 da Figura 4.8, os passos seguintes são a análise semântica e a geração de meta-objetos. As principais funções do analisador semântico do compilador MODL são: (i) a verificação das regras semânticas não validadas pelo analisador sintático; (ii) a identificação e a verificação das referências externas; e (iii) a geração da árvore DOM correspondente ao metamodelo. Para a geração dos meta-objetos,

a árvore DOM é percorrida e os meta-objetos correspondentes ao metamodelo compilado são instanciados.

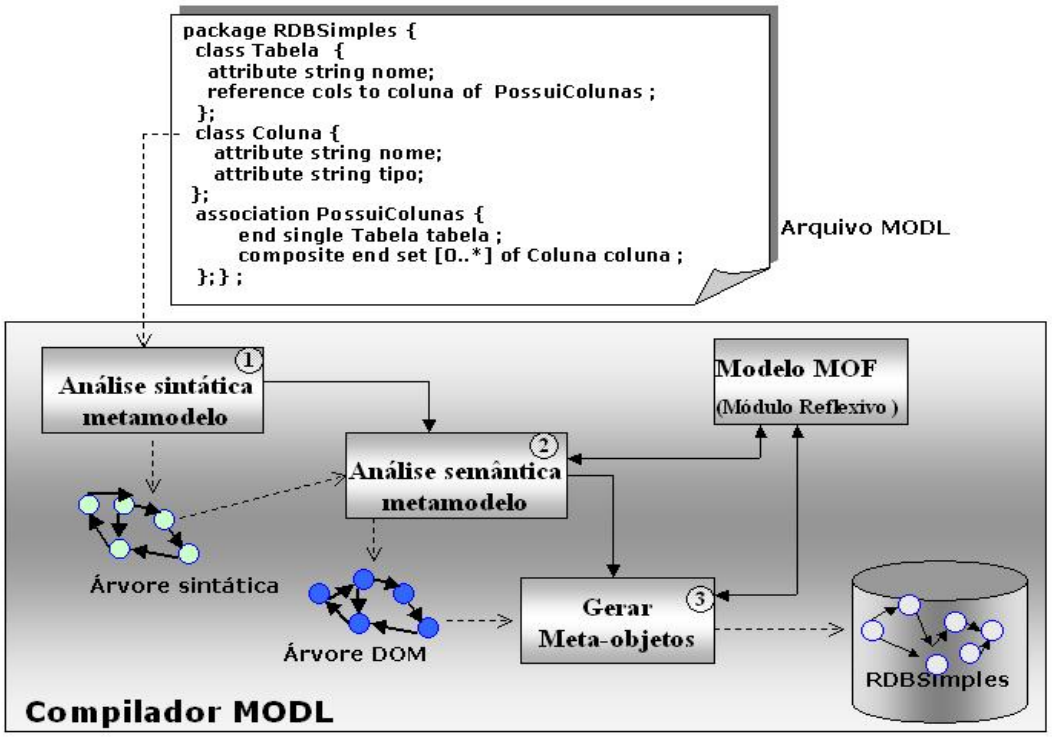

Figura 4.8 – Análise semântica e geração de meta-objetos

Grande parte das regras semânticas que o Modelo MOF define para um metamodelo é verificada pelo analisador sintático como, por exemplo, as regras da associação *Contains*, descrita na seção 3.4.2. Outras regras, como verificar se o limite superior de um intervalo não é menor que o limite inferior, são verificadas pelo analisador semântico.

4.4.3.1 Identificadores

Antes de detalhar a maneira como símbolos externos são resolvidos, é apresentada a solução adotada para a tabela de símbolos. A tabela de símbolos poderia ter sido definida junto com a árvore sintática, mas decidiu-se pelo uso e adaptação do código disponibilizado em [SymtabManager] uma vez que este implementa o conceito de escopo que define-se como uma seção lógica do metamodelo onde o nome dos elementos definidos nesta seção deve ser único. Na linguagem MODL, as construções que definem um novo escopo são as seguintes:

- package, class, association, exception e struct introduzem um escopo para os elementos definidos entre os símbolos "{" e "}";

- em uma operação um escopo é introduzido para os elementos definidos entre os símbolos "(" e ")".

O conceito de escopo estendido se aplica às construções class e package, que suportam o mecanismo de generalização. Um escopo estendido é a união dos identificadores do escopo de um tipo mais os escopos de seus supertipos.

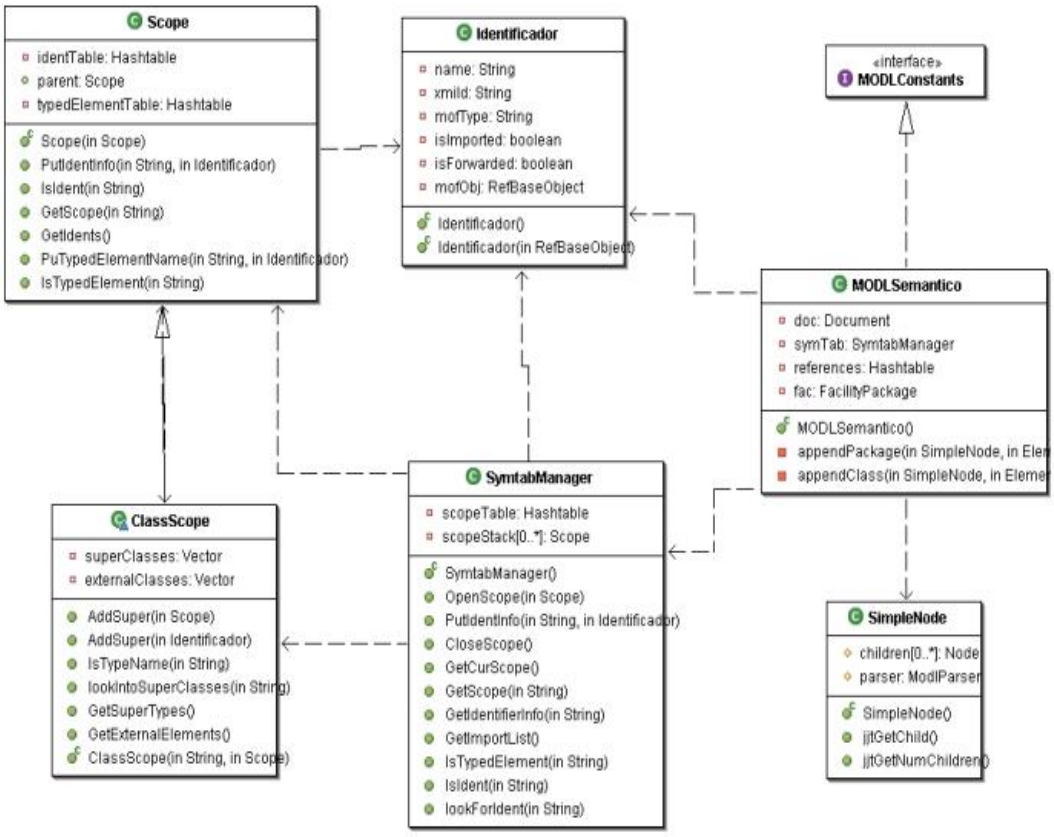

Figura 4.9 – Classes que implementam a tabela de símbolos

A Figura 4.9 apresenta um diagrama UML com as classes responsáveis pelo gerenciamento dos símbolos definidos no metamodelo. A descrição dessas classes é a seguinte:

MODLSemantico: é a classe principal do analisador semântico. Percorre a árvore sintática gerada e, a medida que um elemento do metamodelo é identificado, é completada a análise de

suas semânticas, os símbolos são devidamente identificados e por último, são gerados os nós da árvore DOM correspondente ao elemento. Seus principais atributos e métodos são os seguintes:

Tabela 4.2 – Atributos e métodos da classe MODLSemantico

doc	Árvore DOM.
mr	Repositório onde o metamodelo será inserido.
references	Lista de referências. Após percorrer a árvore sintática, para cada referência definida no metamodelo, o identificador da associação para a qual ela faz referência é atualizado.
appendPackage (simpleNode, Element)	Anexa na árvore DOM os nós correspondentes à construção Package. Existe um método "append" para cada construção do Modelo MOF.

SimpleNode: implementa um nó da árvore sintática. Os principais métodos desta classe são os seguintes:

Tabela 4.3 – Atributos e métodos da classe SimpleNode

jjtGetChild	Retorna um nó filho
jjtGetNumChildren	Retorna o número de nós filhos que o nó corrente possui

Identificador: esta classe mantém as seguintes informações sobre um símbolo:

Tabela 4.4 – Atributos e métodos da classe Identificador

name	Nome do símbolo
xmiId	Identificação única do símbolo na árvore DOM
isForwarded	Indica um símbolo que foi usado mas ainda não definido. Esta situação acontece com a construção reference.
mofType	Indica o tipo MOF do elemento (class, attribute, constraint, etc.). É usado para um identificador declarado no metamodelo que está sendo compilado. Não é necessário quando o identificador é externo porque o método reflexivo *refMetaObject* retorna seu meta-objeto.

isImported	Indica um símbolo importado
mofObj	Um valor diferente de null indica um símbolo externo (importado ou herdado)

Scope: implementa um escopo. Os principais atributos e métodos desta classe são os seguintes:

Tabela 4.5 – Atributos e métodos da classe Scope

IdentTable	Tabela com os identificadores definidos neste escopo
parent	Escopo pai
scopeName	Nome do escopo
PutIdentifier (Identifier)	Adiciona um identificador ao escopo
Identifier [] GetIdentifiers ()	Retorna a lista de identificadores do escopo
GetIdentifier (string)	Retorna um identificador cujo nome é passado como parâmetro

ClassScope: é uma especialização da classe **Scope** e implementa um escopo estendido. Os principais atributos e métodos desta classe são os seguintes:

Tabela 4.6 – Atributos e métodos da classe ClassScope

superClasses	Lista de escopos estendidos
AddSuper (Scope)	Adiciona um escopo em superClasses
lookIntoSuperTypes (string)	Verifica se o símbolo pertence a um escopo estendido
Scope [] GetSuperTypes ()	Retorna a lista de escopos estendidos

SymtabManager: gerencia a tabela de símbolos e a pilha de escopos de um metamodelo. Seus principais atributos e métodos são:

Tabela 4.7 – Atributos e métodos da classe SymtabManager

depth	Profundidade corrente do aninhamento de escopos

scopeStack	Pilha de escopos
scopeTable	Tabela de símbolos, indexada pelo nome do escopo
OpenScope (Scope, boolean)	Adiciona um escopo à pilha. Se o argumento booleano é verdadeiro, é adicionada uma instância de ClassScope
CloseScope ()	Desempilha um escopo
Identifier GetIdentifier (string)	Procura um identificador nos escopos da pilha e nos escopos estendidos destes
Scope GetCurScope ()	Retorna o escopo corrente
PutIdentifier (string, Identifier)	Adiciona um identificador ao escopo corrente
Identifier [] GetImportList ()	Retorna a lista de identificadores importados

As seguintes regras se aplicam à declaração e ao uso de identificadores em um metamodelo descrito em MODL:

1. Um identificador não pode ser declarado mais de uma vez em um mesmo escopo;

2. Se um identificador declarado externamente a um escopo é usado sem qualificação no escopo, então o identificador não pode ser declarado no escopo;

3. Um identificador deve ser declarado antes de ser usado, exceto no caso do elemento reference que é definido dentro do escopo de uma classe e referencia o nome de uma associação que deve ser declarada no mesmo escopo da classe que declarou a referência.

4. Qualquer identificador declarado no escopo estendido de um supertipo não pode ser declarado no escopo da classe ou do pacote;

5. Class e package não podem ter mais de uma declaração do mesmo identificador no seu escopo estendido.

As Tabelas 4.8, 4.9, 4.10 e 4.11 apresentam exemplos que ilustram a aplicação destas regras.

Tabela 4.8 – Aplicação da regra nº 1

```
package P1 {
   class C1 {
      attribute integer id;   // ok
      attribute long id  ;    // erro, quebra regra 1
   };
} ;
```

Tabela 4.9 – Aplicação da regra nº 2

```
package P2 {
  typedef int T1 ;
  class C2 {
     typedef T1 T2;   // ok, usa T1 do escopo externo
     typedef string T1;  // erro, quebra regra 2
  };
} ;
```

Tabela 4.10 – Aplicação da regra nº 3

```
package P3 {
  class C1 {
      reference R to E1 of A ;   // ok, regra 3
  };
  association A {
     composite end optional C1 E1;
     end set [0..*] of C1 E2;
   };
};
```

Tabela 4.11 – Aplicação das regras nº 4 e 5

```
package P1 {
     class C1 {...};
     class C2 {...};
};
package P2 : P1 {
     class C1 {...};    // erro, quebra regra 4
};
package P3 : P2 {
     class C2 {...};    // erro, quebra regra 5
};
package P4 {
     class C1 {...};    // ok
};
package P5 : P1, P4 {
        // erro, quebra regra 5 (C1 está declarado em P1 e P4)
};
```

4.4.3.2 Referências Externas

Referências externas ocorrem em um metamodelo descrito em MODL nas seguintes situações: (i) especialização de uma classe ou pacote; e (ii) importação (ou *clustering*) de um pacote. A Figura 4.10 ilustra estas situações.

Figura 4.10 – Exemplos de especialização e importação em MODL

O pacote P3 é uma especialização de P2 e P1. P5 importa o pacote P1 e a classe C4 do pacote P4. De um pacote importado pode-se re-usar as classes, os tipos de dados (*typedef*) e as exceções. Uma referência externa deve sempre ter o nome completamente qualificado quando usada.

O diagrama UML apresentado na Figura 4.11 ilustra o uso de operações das interfaces reflexivas para obter informações de elementos externos ao pacote sendo compilado.

As interfaces FacilityPackage e ModelRepository definem, respectivamente, os conceitos de Facilidade e de Repositório. FacilityPackage é implementada pela classe Java FacilityPackageImpl que possui o método getRepositories () para retornar todos os repositórios da facilidade. A classe Java ModelRepositoryImpl implementa a interface ModelRepository que define o método getMofPackage () para retornar uma lista com todos os pacotes existentes em um determinado repositório. O pacote retornado é uma instância da classe MofPackageCore e é

denominado, neste contexto, *outermost package* (pacote mais externo). A partir desta instância é obtido o escopo (método getContents ()) de qualquer elemento que seja subtipo da classe *Namespace* e os supertipos (método getSuperTypes ()) de qualquer elemento que seja subtipo da classe *GeneralizableElement*.

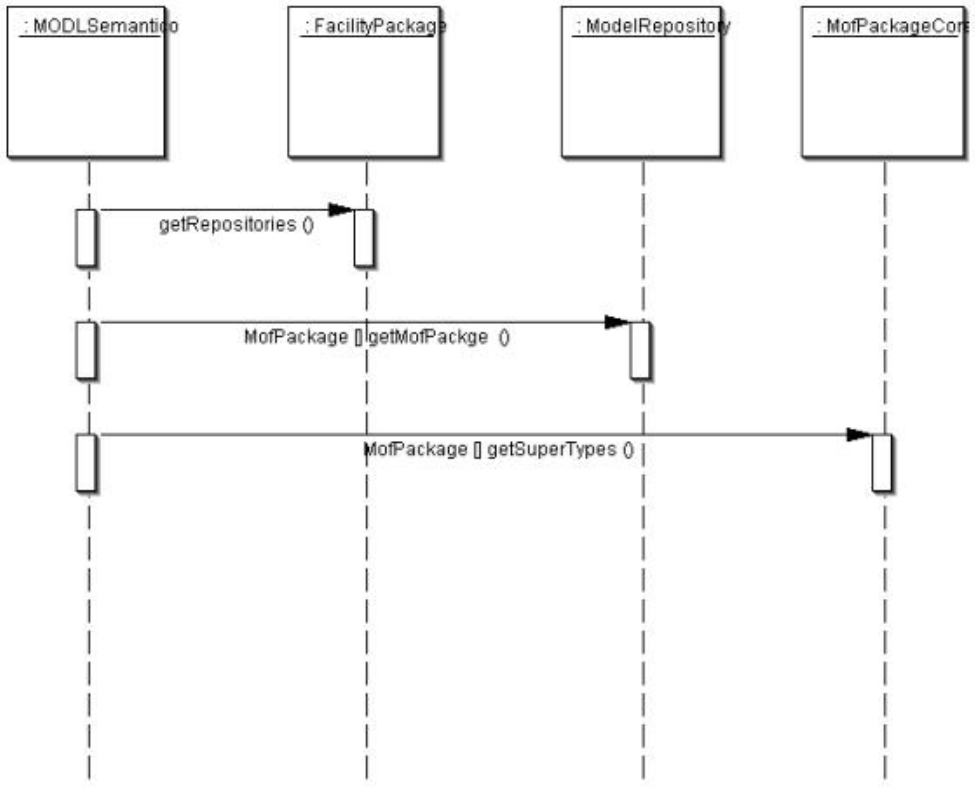

Figura 4.11 – Busca dos repositórios e pacotes de uma facilidade

Utilizando-se os métodos getContents () e getSuperTypes (), as seguintes semânticas são verificadas em relação as referências externas:

1. um pacote não pode importar a si mesmo;
2. um pacote importado não pode ser herdado;
3. não pode haver importação circular. Se um pacote P1 importa o pacote P2, então P2 não pode importar P1 ou, se um pacote herda de P2 então P2 não pode importar P1.
4. um pacote aninhado (*nested*) não pode importar e nem ser importado.

Para a resolução de um identificador, o analisador semântico utiliza o seguinte algoritmo (suponha que deseja-se resolver o uso do símbolo "i"):

1. *esc é igual ao escopo corrente;*
2. *Se esc contém um elemento e cujo nome é "i",*
3. *Então e identifica "i";*
4. *Senão*
5. *Se o escopo esc é estendido*
6. *Então*
7. ***Se " escopo estendido contém um elemento e cujo nome é "i""***
8. *Então e identifica "i" ;*
9. *fim se;*
10. *fim se;*
11. *Se esc está contido em outro esco*po
12. *então (esc = "escopo que contém esc") e repita o passo 2;*
13. *Senão"i" não foi declarado;*
14. *fim se ;*
15. *fim se ;*

O diagrama UML mostrado na Figura 4.12 ilustra a seqüência realizada para pesquisar um símbolo em um escopo estendido (linha 7 do algoritmo).

Primeiro é verificado se o símbolo foi declarado no escopo estendido. Caso não tenha sido, busca-se todos os seus escopos estendidos através do método getSuperTypes () da classe ClassScope. Quando o escopo estendido é externo, executa-se os métodos lookupElement () e lookupElementExtended () para pesquisar, respectivamente, no escopo da referência e nos seus escopos estendidos. Por último, caso o símbolo não tenha sido declarado em nenhum dos escopos pesquisados, a pesquisa é feita na lista de elementos importados, utilizando-se o método lookupElement ().

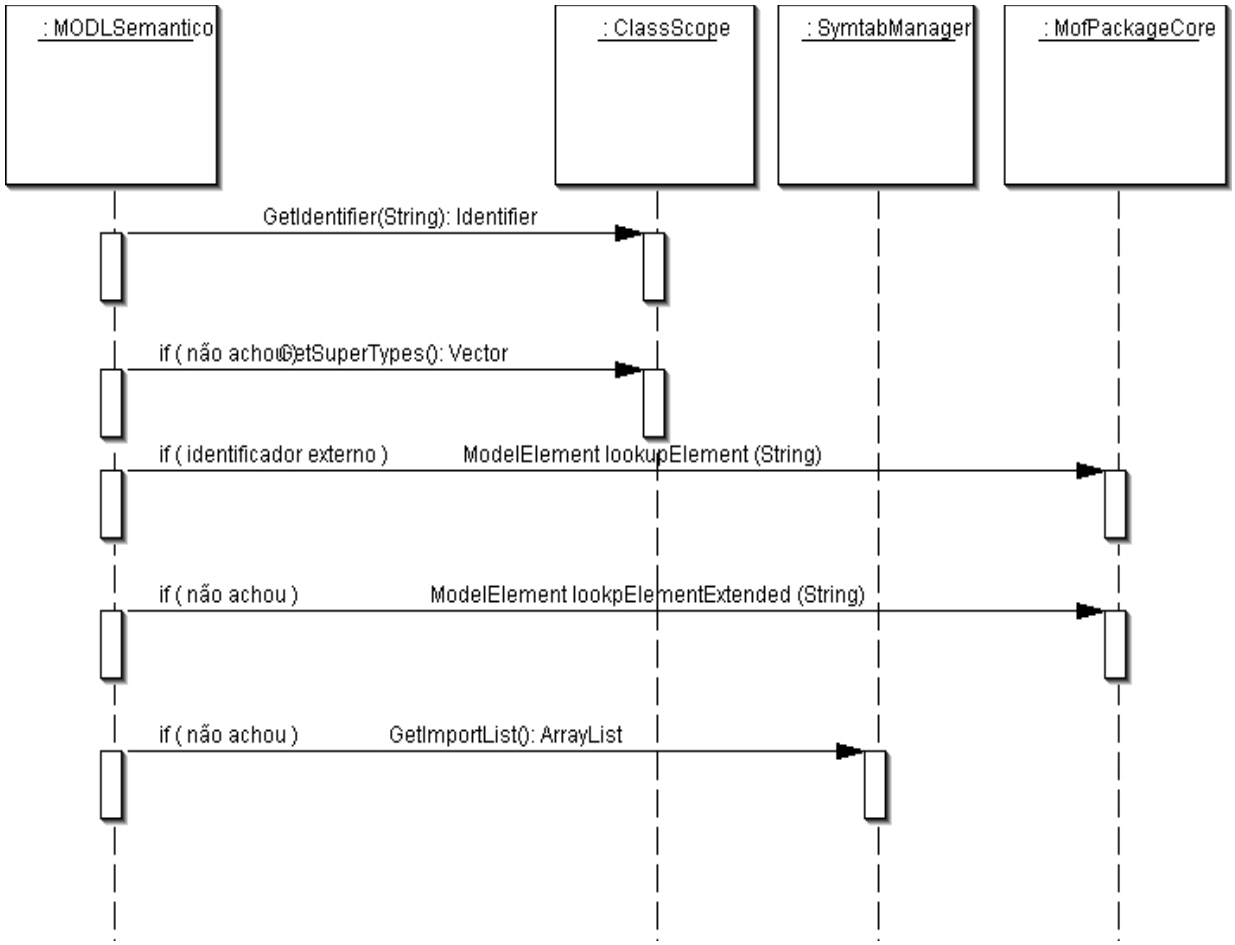

Figura 4.12 – Resolução de um identificador externo

4.4.3.3 Geração de meta-objetos

A geração de meta-objetos é o último passo do compilador MODL. Neste ponto são criadas as instâncias das classes do Modelo MOF que descrevem o metamodelo. Na Figura 4.13 são mostrados alguns dos meta-objetos criados ao final da compilação do metamodelo RDBSimples, apresentado na Figura 4.8. As principais associações (*Contains*, *RefersTo* e *IsOfType*) também são apresentadas.

Para a geração dos meta-objetos é utilizado o mesmo mecanismo usado para importar metamodelos definidos em XMI. Ou seja, a árvore DOM gerada no passo anterior é percorrida e as APIs do Modelo MOF são acessadas para que sejam criadas os meta-objetos correspondentes ao metamodelo compilado.

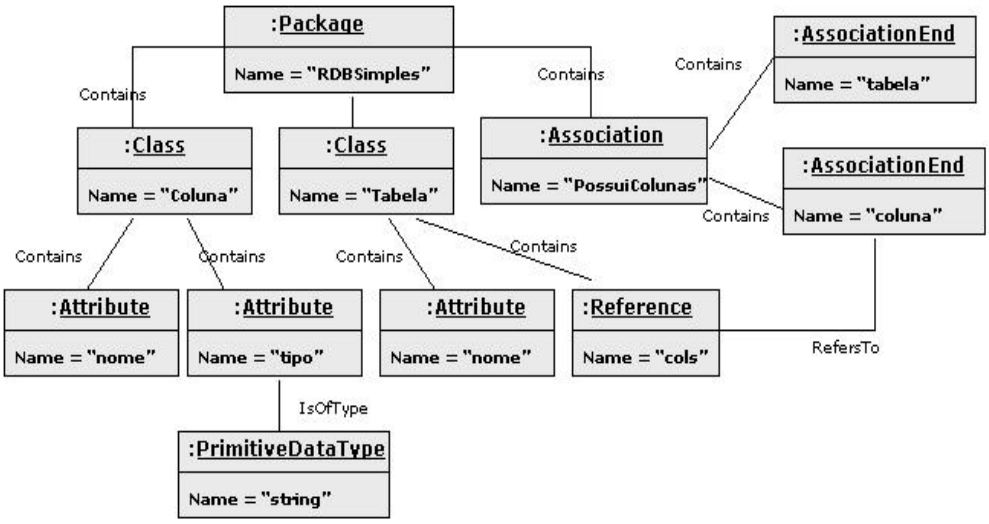

Figura 4.13 – Objetos M3 representando o metamodelo RDBSimples

4.5 Considerações Finais

O software dMOF [dMOF] é um trabalho pioneiro no uso da tecnologia MOF. Assim como o sistema GRM, um compilador transforma um metamodelo descrito em MODL em meta-objetos MOF. Ferramentas para gerar interfaces Corba IDL e XML DTD também estão presentes neste software. Ao contrário do sistema GRM, dMOF suporta a versão MOF 1.3 e não gera interfaces Java. Outra diferença é que dMOF é oferecido com uma licença de uso por tempo limitado.

Na seção 3.5.4 desta dissertação foi apresentado um sistema que, automaticamente, a partir de um metamodelo, gera um compilador capaz de ler instâncias deste metamodelo descritas na notação HUTN e gerar os meta-objetos MOF correspondentes. De modo análogo, pode-se gerar o compilador para o Modelo MOF para ler metamodelos expressos em HUTN. A principal vantagem desta abordagem no contexto desta dissertação é a geração automática do compilador. Assim sendo, ferramentas semelhantes às descritas na seção 3.5.4 são relevantes no sistema GRM para a geração automática de compiladores específicos ao Modelo MOF e dos diferentes metamodelos que venham a ser definidos.

5 EXEMPLO DE APLICAÇÃO: METAMODELOS EM UMA PLATAFORMA DE GOVERNO ELETRÔNICO

5.1 Introdução

São muitas as definições para o termo governo eletrônico (ou e-governo) e nesta dissertação adota-se a seguinte: governo eletrônico é o uso das tecnologias de comunicação e informação (TCI) para promover um governo mais transparente e eficiente para os seus cidadãos, com serviços públicos mais acessíveis [Wimmer 2002]. O processo de transição dos governos em direção ao governo eletrônico passa, geralmente, por quatro fases [Fernandes 2001]:

1. Informacional: caracteriza-se pelo estabelecimento do governo na Internet para a difusão de informações sobre os diversos órgãos de departamentos dos diferentes níveis de governo. Inicialmente, cada órgão ou departamento é referenciado por uma URL diferente. Eventualmente, as URLs podem ser reunidas em um portal que, nesta fase, consiste apenas de uma espécie de catálogo de endereços dos vários órgãos do governo.

2. Interativa: nesta fase, as soluções continuam departamentalizadas mas as aplicações na Internet são capazes de receber informações e dados por parte dos cidadãos, empresas ou outros órgãos.

3. Transacional: caracteriza-se pela realização de operações mais complexas tais como, declaração de imposto de renda, pagamento de contas e compras. Inicia-se adaptações nos processos de trabalho uma vez que serviços anteriormente prestados por funcionários atrás do balcão são realizados por meio da Internet.

4. Avançada: esta fase implica em uma re-invenção do governo. O serviço é disponibilizado por linhas de negócio, e não segundo a divisão real do governo em secretarias, departamentos, etc. Ao lidar com o governo, o usuário não precisa dirigir-se a diferentes órgãos porque é desenvolvido um portal de convergência de todos os serviços prestados pelo governo.

Outros aspectos caracterizam a fase avançada de um governo eletrônico [Almeida 2002]. Dentre eles, Almeida cita:

- uma reengenharia profunda na forma como o governo trabalha a fim de que as informações e as transações pela Internet sejam integradas com os processos internos de trabalho. Muitos dos serviços a serem realizados exigirão uma intensa colaboração entre os diversos órgãos e departamentos, por meio de uma Intranet governamental segura, que integre todos eles. O desenvolvimento da interoperabilidade entre as várias unidades administrativas torna-se premente.

- a implantação de aplicações e de tecnologia como infra-estrutura principal para o fornecimento de serviços 7 dias por semana, 24 horas por dia, através de diversos canais de interação, permitindo que informações e serviços estejam acessíveis a qualquer momento e local.

- a formação de uma infra-estrutura de governo baseada em uma única fonte criando uma organização virtual estruturada em torno dos clientes e suas necessidades.

A Divisão de Software para Sistemas Distribuídos (DSSD) do Centro de Pesquisas Renato Archer (CenPRA) tem em andamento um projeto para criar uma Plataforma de Governo Eletrônico (PGovE) para auxiliar iniciativas governamentais a atingirem a fase avançada [Figueiredo 2003, Figueiredo2004a, Figueiredo2004b].

O objetivo deste capítulo é mostrar a importância de metamodelos em uma solução de governo eletrônico. A linguagem MODL é utilizada para descrever alguns dos metamodelos que serão usados neste projeto.

Este capítulo está organizado da seguinte maneira: na seção 5.2 é feita uma breve descrição da plataforma de governo eletrônico – PGovE e do uso de metamodelos em tempo de projeto e execução. Para ilustrar a aplicação do compilador MODL, são descritos na linguagem MODL três metamodelos. Por último, na seção 5.3 é apresentado um diagrama que ilustra os passos necessários para a geração de um repositório de metadados através do sistema GRM.

5.2 Metadados em uma plataforma de governo eletrônico

Um governo é uma entidade multi-organizacional e seu ambiente computacional caracteriza-se pela forte presença de soluções departamentalizadas e individualizadas,

combinando vários sistemas operacionais, linguagens de programação, tecnologias de rede e bancos de dados. O grande desafio do governo eletrônico é preservar o caráter descentralizado do governo, respeitando a autonomia dos diferentes órgãos que o compõem, mas, ao mesmo tempo, assegurar que soluções de conectividade e interoperabilidade sejam encontradas para que todo cidadão acesse um serviço de qualidade e de forma transparente, ou seja, sem que precise saber qual órgão executará o serviço.

O objetivo principal da DSSD é desenvolver uma plataforma de serviços aberta, flexível e evolutiva que suporte a migração gradual do governo tradicional para o governo virtual, proporcionando suporte: (i) ao reuso de processos de negócio; (ii) ao acesso e integração de aplicações e dados legados; (iii) à autonomia das diversas administrações que compõem o governo.

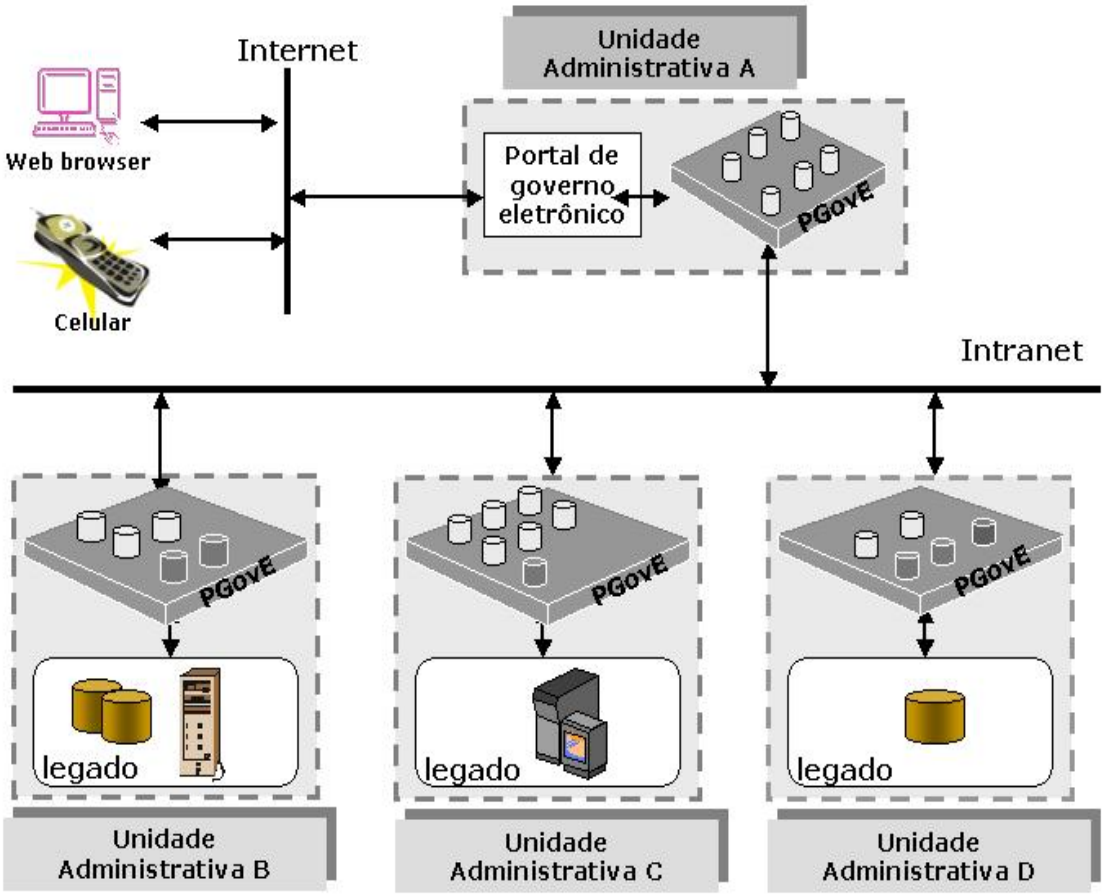

Figura 5.1 – Uma utilização da plataforma PGovE

A Figura 5.1 apresenta uma solução de governo eletrônico utilizando-se a plataforma PGovE. Na solução ilustrada, o Portal do Governo Eletrônico é o ponto único de acesso às informações e aos serviços disponibilizados. O legado é composto por dados operacionais, analíticos e aplicações legadas. PGovE é a plataforma de serviços que promove a integração de serviços eletrônicos fornecidos por diferentes unidades administrativas através de composição de serviços e o mecanismo de federação.

Metadados é uma tecnologia fundamental na plataforma PGovE para lidar, principalmente, com os seguintes aspectos: (i) a complexidade do desenvolvimento de software; (ii) a integração de aplicações e dados; e (iii) o gerenciamento de recursos.

O uso de metamodelos padronizados e formais pela plataforma garante a transformação automática (ou semi-automática) de modelos para uma tecnologia específica e permite o intercâmbio de artefatos de software e a interoperabilidade entre ferramentas, aplicações, *middlewares* e bases de dados.

O uso de metamodelos na plataforma PGovE acontece na fase de projeto, quando serviços eletrônicos são concebidos, e na fase de execução como descrito a seguir.

Fase de projeto: os metamodelos dão suporte ao desenvolvimento de software. Por exemplo:

- **CCA** (*Component Collaboration Architecture*) da especificação EDOC para modelar a estrutura e o comportamento de componentes de uma aplicação (serviços de e-governo);
- ***Entity*** para modelar entidades que representam conceitos do domínio do governo;
- **MMU** (Metamodelo Unificado) [Mejía 2002] para modelar processos de *workflow*.

Fase de execução: os metamodelos são usados para integrar aplicações e descrever recursos. Por exemplo:

- **WSDL** para descrever a interface de um serviço;
- **CWM** para descrever dados oriundos de diferentes bases legadas;

- **CAM** (*Common Application Metamodel*) [OMG 2004e] para modelar diferentes tipos de acesso às aplicações legadas.

Metamodelos não-padronizados, específicos do domínio de governo, serão definidos para suportar regras de negócio, componentes básicos (pagamento de taxas, gerenciamento de conteúdo, controle de acesso, agendamento de atendimento presencial, gerenciamento de documentos) e outros.

Nas seções a seguir, são descritos em MODL alguns dos metamodelos previstos para a plataforma PGovE. Em seguida é apresentado um diagrama ilustrando a definição de um sistema de gerenciamento de modelos *Entity* para uso na plataforma PGovE.

5.2.1 Metamodelo *Entity*

A Tabela 5.1 apresenta uma listagem do metamodelo *Entity*, definido na especificação EDOC [OMG 2004a] e utilizado para modelar entidades que representam conceitos do domínio do governo (por exemplo, cidadão, veículo, imposto, etc).

Tabela 5.1 – Metamodelo *Entity* descrito em MODL

```
package EntityPkg {

   import CCA;
   import DocumentModel;

   tag org.omg.mof.idl_prefix "org.omg.mof" on EntityPkg  ;
   tag javax.jmi.packagePrefix "EntityPkg " on EntityPkg   ;
   tag org.omg.xmi.namespace "Model" on EntityPkg  ;

   class DataProbe: CCA::MultiPort {
     attribute boolean extentProbe;
     reference probes to probes of ProbesAssoc;
   };

   class DataManager: CCA::ProcessComponent {
     attribute boolean networkAccess;
     attribute boolean sharable;
     reference manages to manages of ManagesAssoc;
   };

   class Entity: DataManager {
     attribute boolean managed;
     reference probedBy to probedBy of ProbesAssoc;
     reference parentOf to parentOf of EntityRoleParents;
     reference contextFor to contextFor of EntityRoleContexts;
   };
```

```
class EntityRole {
  attribute boolean virtualEntity;
  reference parent to parent of EntityRoleParents;
  reference roleContext to roleContext of EntityRoleContexts;
};

class Key : DocumentModel::CompositeData {
  attribute boolean primeKey;
  reference keyElements to keyElements of KeyContainsKeyElement;
  reference owner to owner of EntityDataContainsKey;
};

class KeyElement {
  reference containingKey to containingKey of KeyContainsKeyElement;
};

class KeyAttribute: KeyElement {
  reference attributeName to attributeName of KeyAttribute2Attribute;
};

class ForeignKey: KeyElement {
  reference owner to owner of Relationship;
};

class EntityData: DocumentModel::CompositeData {
  reference foreignKeys to foreignKeys of Relationship;
  reference keys to keys of EntityDataContainsKey;
};

association ProbesAssoc {
  end set[1..*] of DataProbe probedBy;
  end set[0..*] of Entity probes;
};

association EntityRoleParents {
  end set[0..*] of EntityRole parentOf;
  composite end single Entity parent;
};

association EntityRoleContexts {
  end set[0..*] of EntityRole contextFor;
  composite end optional Entity roleContext;
};

association KeyContainsKeyElement {
  composite end optional Key containingKey;
  end set[0..*] of KeyElement keyElements;
};

association ManagesAssoc {
  end set[0..*] of DataManager managedBy;
  end single DocumentModel::CompositeData manages;
};

association KeyAttribute2Attribute {
  end set[0..*] of KeyAttribute keyAttribute;
  end single QUOTE "DocumentModel::ECAAttribute" attributeName;
```

```
  };

  association EntityDataContainsKey {
    composite end single EntityData owner;
    end set[0..*] of Key keys;
  };

  association Relationship {
    composite end single EntityData owner;
    end set[0..*] of ForeignKey foreignKeys;
  };

  constraint EntityDataConstraint1 on EntityData, Key, EntityDataContainsKey:
    "EntityData must have a prime Key that is unique within the extent of the EntityData
type (i.e. the type and all sub-types)";

  constraint EntityDataConstraint2 on EntityData, ManagesAssoc, Entity:
    "An EntityData is managedBy an Entity";

  constraint KeyConstraint1 on Key, EntityData, EntityDataContainsKey:
    "If Key is Prime Key = true, then the value must be unique within the extent of the
associated EntityData type and its subtypes.";

  constraint KeyConstraint2 on Key, KeyAttribute2Attribute,
    KeyContainsKeyElement:
  "The attributes that are incorporated into the key must be immutable.";

  constraint KeyConstraint3 on Key, KeyElement, KeyContainsKeyElement:
    "The KeyElements that comprise the key have an immutable sequence.";

  constraint ForeignKeyConstraint1 on ForeignKey, Key, KeyContainsKeyElement,
    Relationship:
  "If the associated Key has PrimeKey = true, then the relationship used to obtain the
Foreign Key value must be immutable.";

  constraint KeyAttributeConstraint1 on KeyAttribute, Key,
    KeyContainsKeyElement, KeyAttribute2Attribute:
    "If the containing Key is designated PrimeKey = true, then the Attribute values that are
incorporated into the key must be immutable.";

  constraint EntityConstraint1 on Entity, EntityData, ManagesAssoc:
    "An Entity manages EntityData, which may have a key and relationships.";

  constraint EntityConstraint2 on Entity, EntityData, ManagesAssoc,
    EntityDataContainsKey:
  "A managed Entity must have a Primary Key.";

  constraint EntityConstraint3 on Entity, EntityData, EntityDataContainsKey,
    ManagesAssoc:
  "A network accessible Entity must have a prime Key.";

  constraint EntityConstraint4 on Entity:
    "An Entity that is Sharable will serialize concurrent transactions that attempt to access
its data.";

  constraint EntityRoleConstraint1 on EntityRole, Entity, EntityRoleParents:
    "The parent Entity of an EntityRole cannot be dynamically changed.";
```

```
  constraint DataProbeConstraint1 on DataProbe:
    "DataProbes only emit messages (i.e. output only)";

  constraint DataProbeConstraint2 on DataProbe, ProbesAssoc, Entity:
    "DataProbe can only attach to an Entity with Managed = true.";

}; // EntityPkg
```

5.2.2 Metamodelo WSDL

A Tabela 5.2 apresenta um fragmento do metamodelo WSDL proposto por [Bézivin 2004].

Tabela 5.2 – Fragmento do metamodelo WSDL descrito em MODL

```
package WSDL {

    tag org.omg.mof.idl_prefix "org.omg.mof" on WSDL ;

    tag javax.jmi.packagePrefix " WSDL " on WSDL;

    tag org.omg.xmi.namespace "Model" on WSDL;

    class WSDLElement  {  };

    class ExtensibleElement:WSDLElement  {  };

    class Definition:ExtensibleElement  {
          attribute string name;
          attribute string targetNameSpace;
          reference services to services of HasAssoc1 ;
    };

    class Service:ExtensibleElement  {
          attribute string name;
          reference ports to ports of HasAssoc2 ;
          reference definition to definition of HasAssoc1 ;
          reference doc to documents of DocumentAssoc1 ;
    };

    class Documentation:WSDLElement  {
          reference services to documented of DocumentAssoc1 ;
          reference operations to documented of DocumentAssoc2 ;
          reference ports to documented of DocumentAssoc3 ;
    };

    class Port:ExtensibleElement  {
         reference doc to documents of DocumentAssoc3 ;
    };

    class Operation  {
         attribute string name;
```

```
        reference inputs to inputs of HasInput ;
        reference outputs to outputs of HasOutput ;
        reference doc to documents of DocumentAssoc2 ;
};

class Input  {
        attribute string name;
        attribute string message;
        reference doc to documents of DocumentAssoc4 ;
};

class Output  {
        attribute string name;
        attribute string message;
        reference doc to documents of DocumentAssoc5 ;
};

association DocumentAssoc1 {
     end optional Documentation documents;
     end single Service documented ;
};

association DocumentAssoc2 {
    end optional Documentation documents;
    end single Operation documented ;
};

association DocumentAssoc3 {
    end optional Documentation documents;
    end single Port documented ;
};

association DocumentAssoc4 {
    end optional Documentation documents;
    end single Input documented ;
};

association DocumentAssoc5 {
    end optional Documentation documents;
    end single Output documented ;
};

 association HasAssoc1 {
     composite end single Definition definition;
     end set [0..*] of Service services;
 };

 association HasAssoc2 {
     composite end single Service service ;
     end set [0..*] of Port ports ;
 };

 association HasInput {
     composite end single Operation operation ;
     end set [0..*] of Input inputs ;
 };
```

```
    association HasOutput {
       composite end single Operation operation ;
       end set [0..*] of Output outputs ;
    };

};   // WSDL
```

5.2.3 Metamodelo para Gerenciamento de Conteúdo

A Tabela 5.3 apresenta a listagem de um metamodelo para gerenciamento de conteúdo de páginas Web proposto por [Wienberg 2003]. Um metamodelo como o descrito a seguir é importante na construção de um Portal de Governo Eletrônico.

Tabela 5.3 – Metamodelo *ContentMngmt* descrito em MODL

```
package ContentMngmt {

   tag org.omg.mof.idl_prefix "org.omg.mof" on ContentMngmt;

   tag javax.jmi.packagePrefix " ContentMngmt " on ContentMngmt;

   tag org.omg.xmi.namespace "Model" on ContentMngmt;

   class  Type {
         attribute string interface_id;
         reference supers to supertype of Inherits ;
         reference subs to subtype of Inherits ;
         reference props to properties of Has ;
         reference target to target of TargetAssoc ;
   };

   class  Property {
         attribute string interface_id;
   };

   class  LinkCollectionProperty:Property {
         attribute integer minCard ;
         attribute integer maxCard ;
         attribute boolean duplicates ;
   };

   class  MediaProperty:Property {
         attribute string mineType ;
   };
   class  StringProperty:Property {
         attribute boolean nullable ;
   };

   class  LinkProperty: LinkCollectionProperty {   };
```

```
  class  LinkSetProperty: LinkCollectionProperty {   };

  class  LinkMapProperty: LinkCollectionProperty {  };

  class  LinkListProperty: LinkCollectionProperty {   };

  association Inherits {
          end optional Type supertype;
          end set [0..*] of Type subtype;
   };
   association Has {
      end single Type type ;
      end set [0..*]  of Property properties ;
   };

   association TargetAssoc {
      end single Type link ;
      end single LinkCollectionProperty target  ;
   };

}; //  ContentMngmt
```

5.3 Geração de um Repositório de Metadados

A Figura 5.2 ilustra os passos necessários para a definição de um repositório de metadados utilizando-se o sistema GRM.

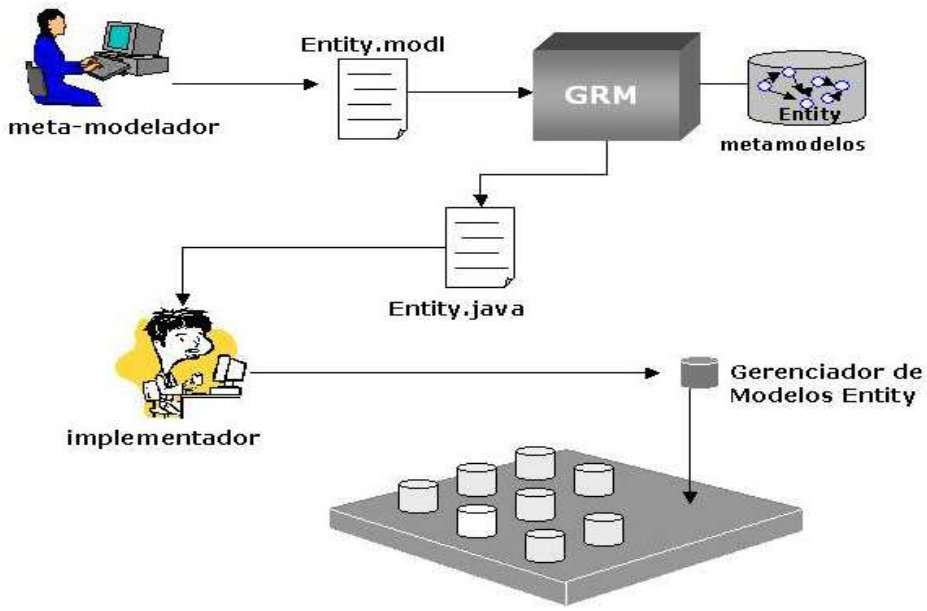

Figura 5.2 – Geração de um repositório de metadados

Inicialmente, o meta-modelador, um especialista do domínio, cria um metamodelo e utiliza o compilador MODL para armazená-lo no sistema GRM. Neste exemplo é utilizado o metamodelo *Entity* para definir conceitos do domínio de governo (por exemplo: cidadão, veículo, imposto, etc).

Uma vez finalizada a fase de meta-modelagem, é gerado automaticamente pelo sistema GRM o código Java do software servidor de um sistema de repositório de modelos de entidades. Em seguida, o implementador trabalha com este código, acrescentando semânticas específicas do domínio de governo e desenvolvendo ferramentas clientes que complementam o sistema com facilidades tais como, um editor para popular o repositório com as entidades e uma ferramenta de visualização.

Considerando que uma solução de governo eletrônica deva preservar o caráter descentralizado do governo, respeitando a autonomia dos diferentes órgãos que o compõem, cada unidade administrativa do governo deve definir e manter as suas próprias entidades. Por exemplo, o Instituto de Identificação Civil mantém no seu repositório as entidades “carteira de identidade” e “ficha criminal” e o departamento de trânsito mantém as entidades “veículo” e “multa”. Assim sendo, no ambiente federativo ilustrado na Figura 5.1, em tempo de execução, um serviço deve ser capaz de consultar o repositório de entidades de outra unidade administrativa. O sistema GRM permite esta interoperabilidade através de interfaces padronizadas e intercâmbio nos formatos XMI e HUTN.

6 CONCLUSÃO

6.1 Introdução

O ponto de partida para este trabalho foi MOF. O estudo desta especificação e outras relacionadas, especialmente os metamodelos de EDOC e CWM, mostrou que um sistema baseado na infra-estrutura de modelagem proposta pelo consórcio OMG traria muitos benefícios para a plataforma de serviços colaborativos flexível, extensível e evolucionária, almejada pela Divisão de Software para Sistemas Distribuídos.

MOF é uma tecnologia fundamental para a plataforma em desenvolvimento. O seu ponto forte é a possibilidade de definir metamodelos de diferentes domínios e permitir o seu reuso através de mecanismos de extensão. Os mapeamentos para as tecnologias Corba IDL e Java oferecem interfaces padronizadas para acesso e manipulação dos metadados. Adicionalmente, os mapeamentos para XML e HUTN provêem mecanismos de intercâmbio de metadados entre ferramentas, *middlewares* e repositórios.

Como ponto fraco de MOF, aponta-se a ausência de uma separação clara entre alguns conceitos de modelagem e a implementação destes em uma tecnologia. Tanto na especificação MOF quanto em JMI, conceitos tais como ciclo de vida e identidade de metadados não foram abordados no Modelo MOF, mas nos mapeamentos concretos (Corba e Java). Um dos objetivos de MOF2 é corrigir este erro.

O sistema GRM foi concebido visando a criação de um ambiente de metamodelagem que facilitasse o desenvolvimento de componentes de software que necessitem de gerenciamento de metadados. Um conjunto de ferramentas para descrever, acessar e manipular metamodelos foi estabelecido. O desenvolvimento do compilador MODL permitiu que, além de importar metamodelos descritos no formato XMI, o sistema GRM oferecesse a alternativa de descrever estes modelos em uma linguagem textual.

Em setembro de 2003, com o início do projeto eGOIA [eGOIA 2004], a DSSD direcionou seus esforços para a construção de uma plataforma aberta de serviços colaborativos no contexto de governo eletrônico, onde metamodelos serão utilizados para suportar a integração do *front-office* e dos sistemas legados. A partir de metamodelos descritos em MODL, serão gerados componentes para gerenciamento de conteúdo, gerenciamento de perfil de usuário e os

metamodelos das especificações EDOC e CWM serão utilizados na integração de dados oriundos de sistemas legados.

6.2 Principais Contribuições do Trabalho

As principais contribuições deste trabalho são comentadas a seguir:

- Construção de um compilador da linguagem MODL

  Como principal contribuição deste trabalho, cita-se o desenvolvimento de um compilador da linguagem MODL para o sistema GRM. Esta linguagem textual é utilizada para a definição de metamodelos MOF. O módulo de análise semântica, juntamente com o de análise sintática, assegura que o metamodelo compilado esteja em conformidade com todas as semânticas e restrições definidas no Modelo MOF. Meta-objetos MOF são gerados para as construções definidas no metamodelo e persistidos em arquivos do sistema no formato XMI.

- Estudo das especificação MOF e JMI

  O capítulo 3 desta dissertação é uma importante contribuição deste trabalho, pois apresenta detalhadamente as especificações MOF e JMI. O estudo destas especificações é fundamental para o entendimento da arquitetura de metamodelagem proposta pelo consórcio OMG e todas as outras especificações relacionadas: XMI, CWM, EDOC e UML.

- Especificação do sistema GRM

  O sistema GRM foi especificado visando principalmente a definição de metamodelos MOF, o intercâmbio de metadados e a geração automática de diferentes artefatos a partir de um metamodelo MOF. O sistema GRM é flexível no sentido que permite a definição de metamodelos oriundos de diferentes domínios. É extensível, permitindo que um metamodelo seja especializado de modo a atender as necessidades específicas de determinado domínio. Por último, baseia-se em padrões abertos.

6.3 Trabalhos Futuros

O desenvolvimento do sistema GRM conforme especificado na seção 4.2, ainda não foi finalizado. O sistema é operacional, oferecendo o compilador MODL e ferramentas para visualização, geração de repositórios JMI, geração de XML DTD e intercâmbio de metadados no formato XMI. A inclusão de ferramentas para gerar XML Schema e a notação HUTN são funcionalidades importantes para a sua disseminação.

A migração para MOF2 é um caminho natural uma vez que esta nova especificação deve refletir o amadurecimento da arquitetura de metamodelagem do consórcio OMG, corrigindo problemas das versões anteriores, como a falta de alinhamento entre MOF e UML, e acrescentando ao seu núcleo funcionalidades importantes tais como controle de versão, *query*, *view*, transformação e gerenciamento do ciclo de vida de metadados.

7 REFERÊNCIAS BIBLIOGRÁFICAS

7.1 Artigos e Livros

[Almeida 2002] Almeida, Marcos O., **Governo Eletrônico no Brasil**, VII Congresso Internacional del CLAD sobre la Reforma del Estado y de la Administración Pública, Lisboa, Portugal, outubro de 2002

http://www.clad.org.ve/fulltext/0043815.pdf acessado em 24/8/2004

[Auth 2002] Auth G., Maur E., Helfert M., **A Model-based Software Architecture for Metadata Management in Data Warehouse Systems,** anais da V International Conference on Business Information Systems, Poznan, Polônia, abril 2002

http://verdi.unisg.ch/org/iwi/iwi_pub.nsf/wwwPublRecentEng/7D83210E1E92D8FBC1256BD7005A972D/$file/BIS02AuthHelfertvonMaur.pdf

acessado em 24/8/2004

[Atkinson 1997] Atkinson, C., **Meta-Modeling for Distributed Object Environments**, 1st International Enterprise Distributed Object Computing Conference (EDOC '97)**,** Gold Coast, Austrália, 1997

http://www-agce.informatik.uni-kl.de/publications/edcoc.pdf acessado em 24/8/2004

[Barreto 1999] Barreto, Cássia M., **Modelo de Metadados para a Descrição de Documentos Eletrônicos na WEB**, dissertação apresentada no Instituto Militar de Engenharia em agosto de 1999, Rio de Janeiro (RJ).

http://genesis.nce.ufrj.br/dataware/teses.htm#Metadados acessado em 24/08/2004

[Bernstein 1994] Bernstein P. A. Dayal U., **An Overview of Repository Technology**, anais 20ª Conferência Very Large Data Base, páginas 705-713, Santiago Chile

[Bernstein 1998] Bernstein P. A., **Repositories and Object Oriented Databases,** ACM/SIGMOD vol. 27, páginas 88-96, 1998

[Bézivin 2004] Jean Bézivin J., Hammoudi S., Lopes D., Jouault F., **An Experiment in Mapping Web Services to Implementation Platforms**, Relatório de pesquisa realizado por l'Institut de Recherche en Informatique de Nantes, Université de Nantes, Nantes – França, março de 2004

http://www.sciences.univ-nantes.fr/info/recherche/Vie/RR/RR-IRIN2004-01.pdf acessado em 24/08/2004

[Costa 2001] Costa, Fábio M., **Combining Meta-Information Management and Reflection in an Architecture for Configurable and Reconfigurable Middleware**, Lancaster University, tese de doutorado apresentada em agosto de 2001.

http://www.comp.lancs.ac.uk/computing/users/fmc/papers/thesis.pdf acessado em 24/08/2004

[eGOIA 2004] **eGOIA - Electronic Government Innovation and Access**

http://www.egoia.info/ acessado em 24/8/2004

[Fernandes 2001] Fernandes A., **e-Governo no Brasil**, XIII Seminário Regional de Política Fiscal organizado pela CEPAL, BIRD, BID e FMI, Santiago-Chile, 22-26 de janeiro de 2001

http://federativo.bndes.gov.br/destaques/egov/egov_estudos.htm acessado em 24/8/2004

[Figueiredo 2003] Figueiredo A., Mendes M., Kamada A., Rodrigues M., Damasceno L., **Using metamodels to promote data integration in an e-Government scenario**, Digital Communities in a Networked Society e-Commerce, e-Business and e-Government, IFIP – Kluwer Accademin Publisher, Capítulo 23, páginas 293-303

[Figueiredo 2004a] Figueiredo A., Mendes M., Kamada A., Rodrigues M., Damasceno L., **Metadata Repository Support for Legacy Knowledge Discovery in Public Administrations**, Lecture Notes in Computer Science, IFIP International Working Conference, KMGov 2004, Krems, Austria, páginas 157-165, Maio 17-19, 2004

[Figueiredo 2004b] Figueiredo A., Mendes M., Kamada A., Borelli J., Rodrigues M., Damasceno L., Souza G., Tizzo N., **Applying MDA Concepts in an e-Government Platform**, artigo aceito na DEXA 2004 - 3rd International Conference on Electronic Government, agosto de 2004, Zaragoza - Espanha

[Frankel 2003] Frankel D., **Model Driven Architecture – Applying MDA to Enterprise Computing**, Wiley Publishing Inc., 2003 páginas 110-113

[Kerhervé 1997] Kerhervé B., Gerbé O., **Models for Metadata or Metamodels for Data ?**, Anais da 2ª IEEE Metadata Conference, 1997.

http://www.computer.org/proceedings/meta97/papers/bkerherve/bkerherve.html acessado em 24/08/2004

[Kobryn 1999] Kobryn C., **UML 2001: A Standardization Odyssey**, Communications of the ACM, Outubro de 1999, Vol. 42, No. 10, páginas 29-37

[Lagoze 1996] Lagoze C., Lynch C. A., Daniel R., **The Warwick Framework: A Container Architecture for Aggregating Sets of Metadata**, Relatório Técnico, junho de 1996

http://techreports.library.cornell.edu:8081/Dienst/UI/1.0/Display/cul.cs/TR96-1593 acessado em 26/11/2004

[Mejía 2002] Mejía J. A. S., **Uma Abordagem Unificada para Modelar Processos de Workflow e seu Software de Suporte**, tese de doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas, Brasil, junho de 2002, página 72

[Poole 2001] Poole J., **Model-Driven Architecture: Vision, Standards And Emerging Technologies**, *position paper* submetido ao ECOOP 2001 - Workshop on Metamodeling and Adaptive Object Models, Junho de 2001, Budapeste, Hungria

[Staudt 1999] Staudt M., Vaduva A., Vetterli T., **Metadata Management and Data Warehousing**, Relatório Técnico, Department of Information Technology, Universidade de Zurique, abril de 1999

ftp://ftp.ifi.unizh.ch/pub/techreports/TR-99/ifi-99.06.ps.gz acessado em 24/08/2004

[Steel 2001] Steel J., Raymond K., **Generating Human-Usable Textual Notations for Information Models**, Anais da 5ª IEEE International Conference on Enterprise Distributed Object Computing (EDOC'01), páginas 250-261,

2001.

[Tan 2002] Tan J., Zaslavsky A., Ewald C., Bond A., **Domain-Specific Metamodels for Heterogeneous Information Systems**, anais da XXXV Hawaii International Conference on System Sciences (HICSS'02), IEEE Computer Society, páginas 3649-3650, 2002

[Wienberg 2003] Wienberg A., **Merging Collection Data Structures in a Content Management System**, anais 11th International Workshop on Software Configuration Management (SCM-11), maio de 2003, Portland, Oregon, EUA

http://www.sts.tu-harburg.de/papers/2003/Wien03.pdf acessado em 24/08/2004

[Wimmer 2002] Wimmer M., Tambouris E., **Online One-Stop Government – A working framework and requirements**, anais do XVII IFIP World Computer Congress, Kluwer Academic Publishers, Boston et al, páginas 117-130, Agosto de 2002, Montreal, Canada

7.2 Especificações

[DSTC 2001] Distributed Systems Technology Center, **dMOF User Guide**, versão 1.1, junho de 2001

http://www.dstc.edu.au/Downloads/CORBA/MOF/dMOF1_1.UserGuide.pdf acessado em 18/08/2004

[EIA 1994] Electronic Industries Association**, CDIF - CASE Data Interchange Format – Overview**, janeiro de 1994

http://www.eigroup.org/cdif/electronic-extracts/OV-extract.pdf acessado em 16/08/2004

[JCP 2002] Java Community Process **, The Java Metadata Interface (JMI) Specification**, versão 1.0, junho de 2002

http://jcp.org/aboutJava/communityprocess/final/jsr040/index.html acessado em 16/08/2004

[MDC 1999] Meta Data Coalition, **Open Information Model**, versão 1.0, abril de 1999

http://www.omg.org/docs/ad/99-04-06.pdf acessado em 16/08/2004

[OMG 2002a] Object Management Group, **Meta Object Facility (MOF) Specification**, versão 1.4, abril de 2002

http://www.omg.org/docs/formal/02-04-03.pdf acessado em 16/08/2004

[OMG 2002b] Object Management Group, **XML Metadata Interchange (XMI) Specification**, versão 1.2, janeiro de 2002

http://www.omg.org/docs/formal/02-01-01.pdfacessado em 16/08/2004

[OMG 2002c] Object Management Group, **Human-Usable Text Notation (HUTN) Specification**, versão 1.0, dezembro de 2002

http://www.omg.org/docs/ptc/04-01-10.pdf acessado em 16/08/2004

[OMG 2003a] Object Management Group, **OMG Unified Modeling Language Specification**, versão 1.5, março de 2003

http://www.omg.org/docs/formal/03-03-01.pdf acessado em 16/08/2004

[OMG 2003b] Object Management Group, **Common Warehouse Metamodel (CWM) Specification**, versão 1.1, março de 2003

http://www.omg.org/docs/formal/03-03-02.pdf acessado em 16/08/2004

[OMG 2003c] Object Management Group, **XML Metadata Interchange (XMI) Specification**, versão 2.0, maio de 2003

http://www.omg.org/docs/formal/03-05-02.pdf acessado em 16/08/2004

[OMG 2003d] Object Management Group, **MDA Guide Version 1.0.1**, junho de 2003

http://www.omg.org/docs/omg/03-06-01 acessado em 16/08/2004

[OMG 2003e] Object Management Group, **Meta Object Facility (MOF) 2.0 Core Proposal**, outubro de 2003

ftp://ftp.omg.org/pub/docs/ptc/03-10-04 acessado em 24/08/2004

[OMG 2004a] Object Management Group, **Enterprise Collaboration Architecture (ECA) Specification**, versão 1.0, fevereiro de 2004

http://www.omg.org/docs/formal/04-02-01.pdf acessado em 16/08/2004

[OMG 2004b] Object Management Group, **Revised Submission for MOF 2.0 Query/View/Transformation RFP**, versão 1.0, abril de 2004

ftp://ftp.omg.org/pub/docs/ad/04-04-01 acessado em 24/08/2004

[OMG 2004c] Object Management Group, **MOF 2.0 Facility and Object Lifecycle Specification**, junho de 2004

ftp://ftp.omg.org/pub/docs/ad/04-04-02 acessado em 24/08/2004

[OMG 2004d] Object Management Group, **MOF 2.0 Versioning and Development Lifecycle Specification**, abril de 2004

ftp://ftp.omg.org/pub/docs/ad/04-04-03 acessado em 24/08/2004

[OMG 2004e] Object Management Group, **UML Profile and Interchange Models for Enterprise Application Integration (EAI) Specification** , março de 2004

http://www.omg.org/docs/formal/04-03-26.pdf acessado em 24/08/2004

7.3 Software

[Apache] **Java API for XML Processing (JAXP)**, The Apache XML Project.

http://xml.apache.org/crimson/ acessado em 17/8/2004

[dMOF] **dMOF**, versão 1.1, Distributed Systems Technology Center

http://www.dstc.edu.au/Downloads/demo_download.html acessado em 26/11/2004

[JavaCC] **Java Compiler Compiler (JavaCC) - The Java Parser Generator**, java.net

https://javacc.dev.java.net/ acessado em 18/08/2004

[SymtabManager] **SymtabManager classes**

http://java.freehep.org/lib/freehep/api/org/freehep/jaco/rtti/cpp/SymtabManager.html acessado em 24/8/2004

[Unisys] Unisys eCommunity , **JMI-RI Documentation CIM**, Versão 1.3, junho/2002

https://ecommunity.unisys.com acessado em 17/08/2004.