UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA ELÉTRICA E DE COMPUTAÇÃO

Interoperabilidade de metadados em aplicações distribuídas: desenvolvimento de ferramentas para validação de metamodelos

Autor: Luciano Lança Damasceno

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. Manuel de Jesus Mendes	DCA/FEEC/UNICAMP
Prof. Dr. Marco Aurélio Amaral Henriques	DCA/FEEC/UNICAMP
Prof. Dr. Ivan Luiz Marques Ricarte	DCA/FEEC/UNICAMP
Prof. Dr. Paulo Sérgio da Silva	DEE/FEB/UNESP

Campinas, SP

Fevereiro/2005

FICHA CATALOGRÁFICA ELABORADA PELA

BIBLIOTECA DA ÁREA DE ENGENHARIA - BAE - UNICAMP

D118i Damasceno, Luciano Lança

Interoperabilidade de metadados em aplicações distribuídas: desenvolvimento de ferramentas para validação de metamodelos / Luciano Lança Damasceno. --Campinas, SP: [s.n.], 2005.

Orientador: Manuel de Jesus Mendes

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

1. Metadados. 2. Engenharia de Software. 3. Software de validação. I. Mendes, Manuel de Jesus. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Resumo

A popularização da Internet criou expectativas e promessas quanto ao uso mais inteligente dos dados disponíveis, culminando na necessidade de interoperabilidade entre aplicações. O uso de padrões de metadados comuns, descrevendo a semântica dos sistemas e suas capacidades possibilitaram a interoperabilidade entre sistemas de informações distribuídos. No entanto, a incompatibilidade entre metamodelos exige uma arquitetura de metadados que suporte esta diversidade. Meta-Object Facility (MOF) é um padrão aberto, suportando e definindo diferentes tipos de metamodelos. Um formato comum de intercâmbio de metadados independente de *middleware* era necessário, XML Metadata Interchange (XMI) foi especificado como resposta a esta necessidade.

Esta dissertação apresenta um sistema de repositório de metadados e metamodelos baseados em MOF, possuindo ferramentas de suporte para validação dos metadados intercambiados através de XMI e visualização de metadados e metamodelos MOF.

Com o objetivo de validar o sistema, o resultado deste trabalho foi aplicado em uma iniciativa de governo eletrônico. Ambientes de tecnologia de informação governamentais se destacam pela heterogeneidade, complexidade e pela forte presença de soluções adaptadas para as diferentes unidades administrativas.

Palavras Chave: metadado, metamodelo, intercâmbio de metadados, XML Metadata Interchange, *Meta-Object Facility.*

Abstract

The increase of Internet popularity creates hopes and promises as the use of available data becomes more intelligent, culminating in the need of interoperability among applications. The use of common metadata standards, describing the semantics and capabilities of the systems, enables interoperability of distributed information systems. However, metamodels incompatibility demands a metadata architecture that supports this diversity. Meta Object Facility (MOF) is an open standard, supporting and defining different kinds of metamodels. Furthermore, it was necessary a common metadata interchange format independent of middleware. XMI was specified as an answer to this need.

This dissertation presents a MOF metadata repository system, having support tools to validate the interchanged metadata and to visualize the stored metamodels and metadata.

In order to validate the system; the result of this work was applied in an electronic government initiative. Information technology of governmental environments is cleared by the heterogeneity, complexity, and strong presence of the adjusted solutions to the different administrative units.

Key Words: metadata, metamodel, metadata interchange, XML Metadata Interchange, Meta-Object Facility

...where ignorance is bliss,

'Tis folly to be wise.

Thomas Gray

Aos meus pais Lucas e Ivani.

AGRADECIMENTOS

Em primeiro lugar, sou grato ao meu orientador: Professor Dr. Manuel de Jesus Mendes. Sem a sua orientação, cumplicidade e amizade o trabalho não teria se concretizado.

Aos amigos do CenPRA, pela paciência, apoio e dedicação. Não poderia deixar de citar as especiais atenções da Adriana Figueiredo e do amigo Neil. Obrigado.

Ao meu irmão Lucas, à turma e a todos da república em Barão Geraldo que durante todos esses anos não puderam contar com a minha presença ou, no último caso, tiveram que me "aturar" durante anos. Em especial ao Lucio, pela força e palavras de apoio nos momentos de estudos diários.

À minha eterna namorada pelo amor, incentivo, compreensão e carinho durante todos esses anos. Sabendo sempre contornar a distância e o tempo gasto com os estudos, brotando ainda mais forte a relação de respeito e amor entre nós. A você dedico esta tese.

ÍNDICE

1 INTRODUÇÃO 1

1.1 CARACTERIZAÇÃO DO PROBLEMA 1

1.2 MOTIVAÇÃO 2

1.3 OBJETIVOS DO TRABALHO 3

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO 3

2 EXTENSIBLE MARKUP LANGUAGE - XML 5

2.1 LINGUAGENS DE MARCAÇÃO 5

2.2 HISTÓRIA DA XML 5

2.3 A LINGUAGEM XML 6

2.3.1 Objetivos e Vantagens da linguagem XML 6

2.3.2 As características da linguagem XML 8

2.3.3 Tipos de marcação XML 10

2.4 VALIDAÇÃO DE CLASSES DE DOCUMENTOS XML 12

2.4.1 Document Type Definition (DTD) 13

2.4.2 XML Schema 17

2.5 XML NAMESPACE 19

2.6 CONSIDERAÇÕES FINAIS 20

3 METADADOS E PADRÕES DE INTEROPERABILIDADE DE MODELOS. 23

3.1 INTRODUÇÃO 23

3.2 METADADOS 25

3.3 ARQUITETURA DE METADADOS DE 4 CAMADAS 26

3.4 PADRÕES DE INTERCÂMBIO DE MODELOS 29

3.4.1 Universal Object Language 31

3.4.2 CASE Data Interchange Format (CDIF) 33

3.4.3 Resource Description Framework (RDF) 38

3.4.4 O Open Information Model e o formato XML Encoding 42

3.5 CONSIDERAÇÕES FINAIS 45

4 META OBJECT FACILITY E XML METADATA INTERCHANGE 47

4.1 META OBJECT FACILITY (MOF) 47

4.1.1 A arquitetura de Meta-modelagem MOF 48

4.1.2 Os níveis da arquitetura MOF 48

4.1.3 O Modelo MOF (nível M3) 51

4.1.4 Principais construções de metamodelagem do Modelo MOF 52

4.1.5 Interfaces Reflexivas MOF 55

4.1.6 Considerações finais sobre o MOF 57

4.2 XML METADATA INTERCHANGE (XMI) 58

4.2.1 Fundamentos 60

4.2.2 O arquivo XMI 63

4.2.3 Problemas encontrados no padrão XMI 66

4.2.4 Usabilidade do padrão XMI 67

4.3 MODEL DRIVEN ARCHITECTURE (MDA) 68

4.4 CONSIDERAÇÕES FINAIS 69

5 MÓDULO GERADOR DTD XMI 71

5.1 GERENCIADOR DE REPOSITÓRIOS DE METADADOS (GRM) 71

5.2 COMPLEX INFORMATION MANAGER (CIM) 72

5.3 ARQUITETURA DO GRM 74

5.4 MÓDULO GERADOR DTD-XMI 75

5.4.1 Principais requisitos adotados *76*

5.4.2 Detalhes de implementação *76*

5.4.3 Funcionamento do Módulo DTD-XMI *77*

5.4.4 Representação DTD XMI para classes de metamodelo *83*

5.4.5 Testes *84*

5.5 GERADOR HTML 86

5.5.1 Detalhes de Implementação *87*

6 CENÁRIO DE USO 91

6.1 INTEROPERABILIDADE DE METADADOS EM PLATAFORMAS DE GOV. ELETRÔNICO 92

6.1.1 Integração de serviços *94*

6.1.2 Integração de dados legados *97*

6.2 CONSIDERAÇÕES FINAIS 100

7 CONCLUSÃO 101

7.1 PRINCIPAIS CONTRIBUIÇÕES DO TRABALHO 102

7.2 TRABALHOS FUTUROS 103

8 REFERÊNCIAS BIBLIOGRÁFICAS 105

8.1 ARTIGOS E LIVROS 105

8.2 ESPECIFICAÇÕES 108

8.3 SOFTWARE 112

LISTA DE FIGURAS

Figura 3.1 – Níveis de Abstração /Refinamento.....27

Figura 3.2 – Arquitetura de metadados de quatro camadas.....29

Figura 3.3 – A família de padrões CDIF.....36

Figura 3.4 – Arquitetura de padrões W3C XML Semântica.....39

Figura 4.1 – Construções de metamodelo M2 como instância das construções M3.....50

Figura 4.2 – Um modelo e o seu Metamodelo.....51

Figura 4.3 – Diagrama de classes do Modelo MOF.....52

Figura 4.4 – Exemplo da arquitetura MOF utilizando o metamodelo UML.....53

Figura 4.5 – Arquitetura MOF de 4 camadas.....58

Figura 4.6 – Mapeamentos Padrões de um Metamodelo MOF.....59

Figura 4.7 – Obtendo artefatos XML de modelos MOF.....62

Figura 4.8 – Arquitetura Orientada a Modelos.....69

Figura 5.1 – Pacotes do Gerenciador de Repositórios de Metadados (GRM).....72

Figura 5.2 – Arquitetura do GRM.....74

Figura 5.3 – Tela da ferramenta administrativa do sistema GRM.....78

Figura 5.4 – Arquitetura do Módulo Gerador DTD-XMI.....79

Figura 5.5 – Diagrama de Seqüência simplificado da geração do DTD.....82

Figura 5.6 – Fragmento de um documento XMI e o seu respectivo DTD.....82

Figura 5.7 – Página HTML gerada pelo Visualizador de Metadados.....87

Figura 6.1 – Uma utilização da plataforma PGovE [FIGUEIREDO04].....93

Figura 6.2 – Interoperabilidade entre repositórios através de XMI.....95

Figura 6.3 – Utilização do módulo DTD-XMI dentro da pGovE.....96

Figura 6.4 – Seqüência de atividades de um serviço de governo de emissão de segunda via de carteira de identidade. 97

Figura 6.5 – Metamodelo Type Descriptor 98

LISTA DE TABELAS

Tabela 2.1 – Exemplo de um documento XML. 9

Tabela 2.2 – Expressão para a criação de elementos. 10

Tabela 2.3 – Expressão para criação de doctype 10

Tabela 2.4 – Exemplo de DTD 16

Tabela 2.5 – Declaração de elementos simples 18

Tabela 3.1 – Exemplo da instrução de metadados RDF 42

Tabela 4.1 – Interfaces reflexivas MOF 56

Tabela 4.2 – Interfaces reflexivas adicionais JMI 56

Tabela 4.3 – Exemplo de um documento XMI 65

Tabela 5.1 – Principais classes do módulo DTD-XMI 77

Tabela 5.2 – Representação DTD de classes de um metamodelo 83

Tabela 5.3 – Regra de produção para DTD em EBNF 84

Tabela 5.4 – Representação DTD de atributos de metamodelos do tipo boolean e enumeration 84

Tabela 5.5 – Principais classes do VM 88

Tabela 6.1 – Metamodelos CAM de representação física 98

Tabela 6.2 – Exemplo de parte do DTD-XMI gerado do metamodelo Type Descriptor 99

LISTA DE SIGLAS

API	Application Programming Interface
BPEL4WS	Business Process Execution Language for Web Services
BPP	Business Process Profile
CAM	Common Application Metamodel
CDIF	Case Data Interchange Format
CenPRA	Centro de Pesquisa Renato Archer
CORBA	Common Object Request Broker Architecture
CWM	Common Warehouse Metamodel
DOM	Document Object Model
DTD	Document Type Definition
DSSD	Divisão de Software de Sistema Distribuído
EAI	Enterprise Application Integration
ECA	Enterprise Collaboration Architecture
EDOC	Enterprise Distributed Object Computing
eGOIA	Electronic Government Innovation and Access
GRM	Gerenciador de um Repositório MOF
HTML	HyperText Markup Language
HTTP	HyperText Transport Protocol

HTTPS	HyperText Transport Protocol Secure
HUTN	Human-Usable Textual Notation
IDL	Interface Definition Language
JMI	Java Metadata Interface
MathML	Mathematic Markup Language
MDA	Model-Driven Architecture
MDC	Meta Data Coalition
MODL	Meta-Object Definition Language
MOF	Meta-Object Facility
OCL	Object Constraint Language
OIM	Open Information Model
OMG	Object Management Group
PGovE	Plataforma de Governo Eletrônico
PIM	Platform Independent Model
PSM	Platform Specific Model
RDF	Resource Description Framework
RDFS	RDF Schema
RTIM	Repository Type Information Model
SAX	Simple API for XML
SGML	Standard Generalized Markup Language

SMIF	Stream-based Model Interchange Format
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
UML	Unified Modeling Language
UOL	Universal Object Language
XIF	XML Interchange Format
XMI	XML Metadata Interchange
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
WSCI	Web Service Choreography Interface
WSDL	Web Service Description Language
W3C	World Wide Web Consortium

1 INTRODUÇÃO

1.1 Caracterização do Problema

Wegner [WEGNER96] definiu interoperabilidade como sendo a capacidade de dois ou mais componentes de software cooperarem entre si, apesar de diferenças em linguagens, interfaces e plataformas de execução. Para isto, é necessário que as aplicações consigam ler e entender dados representados em diferentes padrões de representação, ou seja, metadados. Hoje, a Internet e a *World Wide Web* (Web) aceleraram as expectativas de se obter interoperabilidade entre aplicações.

O uso de metadados possibilitou a integração e a interoperabilidade de sistemas de informação distribuídos. Metadados definem a estrutura e o significado de objetos de dados, permitindo que aplicações saibam como processar solicitações e tarefas. No entanto, um grande limitador de interoperabilidade atualmente está justamente na incompatibilidade de metadados.

Em ambientes de sistemas de computação distribuídos, os diferentes padrões de metadados existentes dificultam a integração e a interoperabilidade de aplicações que utilizam diferentes padrões de tecnologias de middleware.

A adoção de um padrão único de metadado capaz de cobrir todos os domínios foi uma solução proposta no passado para o problema de incompatibilidade dos diferentes metadados existentes [ECKERSON99]. Esta tentativa fracassou, deixando o setor de gerenciamento de metadados sem uma linha comum.

Arquiteturas de metadados foram introduzidas para lidar com o problema de heterogeneidade entre os diversos metadados, a fim de se obter interoperabilidade. Essas arquiteturas permitem que um mesmo metadado seja utilizado em múltiplas plataformas através de padrões de mapeamentos específicos, melhorando a portabilidade das aplicações.

O consórcio *Object Management Group* (OMG) ratificou em 1997, a especificação Meta-Object Facility (MOF). Fundado em 1989, OMG é uma organização internacional

que visa promover a teoria e a prática de tecnologias orientadas a objeto no desenvolvimento de software.

A especificação MOF define uma linguagem abstrata e um *framework* para especificar, construir e gerenciar metadados [OMG02a]. MOF é baseado em uma arquitetura de metadados em quatro camadas, permitindo sua manipulação e gerenciamento. O padrão procede da premissa básica de que haverá mais de um tipo de metadado, portanto, existirá mais de uma linguagem de modelagem de metadado [FRANKEL03].

MOF permite mapeamentos para diferentes tecnologias como Java, CORBA e XML. O mapeamento MOF-Java de metadados em interfaces é definido através da especificação *Java Metadata Interchange* (JMI) [JCP02]. O mapeamento MOF-CORBA é especificado dentro da própria especificação MOF.

O mapeamento MOF-XML é um mecanismo utilizado por ferramentas, repositórios e *middleware* para o intercâmbio de modelos e metamodelos serializado em documentos XMI e *Document Type Definition* (DTD)/*XML Schemas*, respectivamente.

O mapeamento MOF-XML é definido através das regras de produção existentes na especificação XML Metadata Interchange (XMI), permitindo a geração de documentos e DTD XMI. Por documento ou DTD XMI entende-se um documento XML ou um DTD XML gerado através de regras de mapeamento MOF-XMI definidos na especificação XML Metadata Interchange (XMI) do consórcio OMG.

1.2 Motivação

O Centro de Pesquisa Renato Archer (CenPRA) está buscando a criação de uma plataforma aberta de serviços de governo eletrônico tendo como base metamodelos padronizados como mecanismo de integração e interoperabilidade. Visando dar suporte a definição e o gerenciamento de diferentes metamodelos e metadados o sistema Gerenciador de Repositórios de Metadados (GRM) está sendo desenvolvido. O GRM facilita o desenvolvimento de componentes de software que requeiram o gerenciamento e intercâmbio de metadados, sendo baseado nos padrões MOF, JMI e XMI.

1.3 Objetivos do Trabalho

O propósito desta dissertação é a utilização de tecnologias abertas para o desenvolvimento de um sistema de gerenciamento de metadados. Partindo-se deste princípio, optou-se pela utilização das tecnologias MOF, JMI e XMI. As principais contribuições deste trabalho são:

1. Desenvolvimento de um módulo para geração de metamodelos MOF em DTDs XMI.
2. Desenvolvimento de um gerador de HTML facilitando a leitura dos metadados armazenados pelo sistema.
3. Demonstrar a importância de interoperabilidade de metamodelos e modelos MOF, respectivamente, em iniciativas de governo eletrônico.

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

1.4 Organização da Dissertação

Esta dissertação encontra-se organizada de acordo com a seguinte estrutura de capítulos:

- Capítulo 2: apresenta os conceitos básicos da sintaxe da linguagem XML e algumas características de sua utilização.
- Capítulo 3: apresenta os conceitos relacionados a metadados e os principais padrões de interoperabilidade de metadados relacionados ao consórcio OMG para a solicitação de proposta do XMI.
- Capítulo 4: introduz os principais conceitos e fundamentos dos padrões *Meta-Object Facility* e *XML Metadata Interchange*.
- Capítulo 5: descreve a implementação do módulo gerador de DTD-XMI, assim como do gerador de HTML pertencentes ao sistema GRM.

- Capítulo 6: expõe um cenário de utilização das implementações realizadas nesta dissertação no contexto de governo eletrônico.
- Capítulo 7: apresenta as considerações finais desta dissertação, juntamente com sua contribuição e trabalhos futuros.

2 EXTENSIBLE MARKUP LANGUAGE - XML

Neste capítulo, os conceitos básicos da linguagem XML serão descritos brevemente para a compreensão e a definição dos termos usados neste e nos demais capítulos desta dissertação.

2.1 Linguagens de Marcação

O termo "linguagem de marcação" é derivado da prática de publicação tradicional de adicionar instruções de impressão nas margens dos manuscritos. Esta prática era chamada de *marking up*.

Uma linguagem de marcação é um conjunto de símbolos que podem ser inseridas no texto de um documento para demarcar e rotular as suas partes, possibilitando desta maneira, representar tanto o conteúdo (texto) quanto as instruções sobre sua estrutura.

Aspectos de apresentação podem ser codificados dentro do próprio documento, juntamente com a sua estrutura e o seu conteúdo ou definidos em documentos separados. A separação do conteúdo e da estrutura de apresentação tornou-se uma importante característica dos documentos eletrônicos. Essa característica facilitou a utilização e o processamento desses documentos pelos programas de computador. É creditado a William W. Tunnicliffe, em 1967, a idéia de separar do conteúdo as instruções de apresentação em documentos eletrônicos [GOLDFARB96].

2.2 História da XML

No final do ano de 1997, o *World Wide Web Consortium* (W3C) publicou a versão 1.0 da linguagem XML. O termo XML é o acrônimo para *eXtensible Markup Language*. Essencialmente, XML é um subconjunto ou dialeto refinado e simplificado da *Standard Generalized Markup Language* (SGML), que possibilita a transmissão, recebimento e processamento de documentos SGML genéricos através da Web [W3C04b].

SGML é um padrão internacional de meta-linguagem, não proprietário e de código aberto, sendo um dos seus objetivos a garantia de que documentos codificados de acordo com suas regras possam ser transmitidos sem perda de informação [ISO86]. Tanto a

linguagem HTML quanto XML são derivadas do SGML, sendo HTML uma aplicação e a XML um perfil (*profile*), i.e, um subconjunto específico do padrão. XML conseguiu manter todos os principais conceitos e estruturas do SGML e, ao mesmo tempo, abandonar toda a sua complexidade. Além disto, XML introduziu uma família de linguagens que oferecem um melhor gerenciamento semântico da informação do que HTML [BERGHOLZ00]. Uma comparação detalhada entre SGML e XML pode ser consultada em [CLARCK97].

Inicialmente, XML foi vista errôneamente, como sendo uma linguagem de editoração de páginas Web, substituta da HTML. Ultimamente, o padrão XHTML, uma linguagem derivada da XML, está tentando se tornar o substituto natural da HTML na editoração de páginas Web [SIMPSON00].

A especificação XML, atualmente na versão 1.1, é mantida pelo grupo de trabalho *XML Core* da W3C, originalmente conhecido como o *SGML Editorial Review Board*. Este grupo também é responsável por outras especificações como, por exemplo, XML *Namespace* e XML *Information Set* (XML Infoset).

2.3 A linguagem XML

2.3.1 Objetivos e Vantagens da linguagem XML

Os principais objetivos planejados pela W3C para a linguagem são [W3C04b]:

- ser diretamente utilizável na Internet;
- suportar uma grande variedade de aplicações;
- ser compatível com SGML;
- e permitir a fácil criação de programas, nos quais se processem seus documentos.

Listam-se, a seguir, algumas das principais vantagens da linguagem:

- Extensibilidade, isto é, a capacidade de criação de novos elementos em um documento XML, conforme a necessidade, adaptando sua estrutura a qualquer domínio específico. A duplicidade de nomes entre dois elementos diferentes é tratada através da definição de *namespace*, abordado na seção 2.5.

- Autodescrição, isto é, a característica autodescritiva dos documentos que os torna relativamente fáceis de serem interpretados, manipulados e interrogados. Isto confere à linguagem "habilidades semânticas" que possibilitam melhorias significativas em processos de recuperação e disseminação da informação.
- A possibilidade de representar estruturas complexas, tais como árvores ou grafos de profundidade arbitrária, apesar da sua simplicidade.
- A sua flexibilidade, que possibilitando a representação, tanto de dados estruturados quanto de dados semi-estruturados.
- Validação de documentos XML através da definição de esquemas que permitem a verificação estrutural de dados nos documentos.
- Manipulação do conteúdo de um documento XML através de APIs padronizadas, por exemplo, DOM[1] e SAX[2], atingindo um alto nível de automação.
- Utilização para o desenvolvimento de novas linguagens baseadas em XML devido a sua natureza metalingüística. Pode-se citar, como exemplo, a *Mathematic Markup Language* (MathML) para descrever notações matemáticas.
- Independência de plataforma, não sofrendo variações em função da plataforma que processa XML. Esta característica possibilita a sua utilização na integração de sistemas heterogêneos.
- Compatibilidade com os protocolos *HyperText Transport Protocol* (HTTP), *HyperText Transport Protocol Secure* (HTTPS) e *Secure Socket Layer* (SSL), torna XML um padrão natural para transmissão de documentos via Web.

---

[1] *Document Object Model* (DOM) API é uma proposta oficial da W3C. Consiste em uma interface independente de plataforma e linguagem que permite que programas e scripts acessem e atualizem dinamicamente o conteúdo, estrutura e o estilo do documento [W3C00].

[2] *Simple API for XML* (SAX) originalmente disponível somente para a linguagem Java, consiste em API para a manipulação de documentos XML. Atualmente possui versões para várias linguagens de programação [SAX02].

- Separação entre conteúdo e apresentação, que possibilita obter múltiplas perspectivas sobre um mesmo documento, utilizando linguagens de estilo como, por exemplo, *eXtensible Stylesheet Language* (XSL)[3]. Pode-se criar um único documento XML e esse ser processado ou apresentado automaticamente em inúmeros dispositivos com características de entrada e saída variadas.

As vantagens citadas preenchem algumas das demandas da Web: independência de plataforma, separação entre conteúdo e apresentação, independência de domínio de aplicação e simplicidade. Essas características tornaram a XML parte integral de inúmeras tecnologias chave, tais como: portais Web, sistemas de gerenciamento de conteúdo, transações e trocas B2B.

Outra vantagem que deve ser destacada é capacidade de definir outras linguagens, trazendo benefícios como: baixa curva de aprendizagem para todas as linguagens derivadas; utilização das mesmas ferramentas básicas para a manipulação e processamento dessas linguagens e realização de transformações entre as linguagens através de folhas de estilo.

As linguagens derivadas de XML podem ser criadas orientadas a domínios (verticais) ou a tarefas (horizontais). As primeiras definem formatos de troca dentro de um setor ou comunidade, por exemplo, ebXML e Dublin Core. Já as horizontais atuam como linguagens globais para tarefas técnicas especiais ou orientadas a aplicações, provendo conjuntos de marcações úteis numa grande diversidade de contextos, como por exemplo, para a descrição de serviços Web (*Web Service Description Language* - WSDL).

2.3.2 As características da linguagem XML

A linguagem XML descreve uma classe de objetos de dados [W3C04b], i.e., documentos que estão de acordo com a recomendação W3C XML. O fato de documentos

[3] Especificação da W3C. O XSL é o modo de se definir folhas de estilo em XML. Possui tags predefinidas em XML que definem o formato da informação e um documento de formato padrão (*template*) para XML. Trabalha em conjunto com o XML e define atributos de estilo para o conteúdo a ser apresentado.

poderem ser representados como um objeto Java ou, simplesmente, como um registro em um banco de dados relacional, fez com que o termo “objetos de dados” fosse adotado para descrever diferentes representações.

O modelo de dados ou conjunto de informações XML é independente do formato real de um documento. Arquivos texto XML e árvores DOM são exemplos de formatos reais de documentos XML. A estrutura dos documentos é vista como uma árvore de nós rotulados com referências, onde a maioria da estrutura léxica é dedicada para definir esta árvore [BOS97].

Segundo a recomendação W3C XML [W3C04b], “cada documento XML possui tanto uma estrutura lógica quanto uma física”. A estrutura lógica é composta de declarações, elementos, comentários, referências de caracteres e instruções de processamento. Já a estrutura física é composta de entidades, sendo iniciada por uma raiz ou entidade documento [MSDN04b].

Documento XML é composto basicamente por um cabeçalho, chamado prólogo, e do corpo do documento, chamado instância. A Tabela 2.1 demonstra o prólogo, declarado sempre na primeira linha do documento, começando com <?xml. No prólogo são declaradas informações que identificam o documento como sendo XML, assim como a versão da linguagem utilizada e a declaração do tipo de documento, utilizado para a validação do documento. A instância do documento contém os dados do documento, organizados como uma hierarquia de elementos.

Tabela 2.1 – Exemplo de um documento XML.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE CD SYSTEM “cd.dtd”>
<!—comentário -->
<![CDATA[valor > R$10,00]]>
<CATALOGO>
    <CD preco=”R$18.90” ano=”1985”>
        <TITULO>Empire Burlesque</TITULO>
        <ARTISTA>Bob Dylan</ARTISTA>
    </CD>
  </CATALOGO>
```

2.3.3 Tipos de marcação XML

Existem seis tipos diferentes de marcação possíveis de serem utilizadas em um documento: elementos, atributos, referências de entidades, instruções de processamento, comentários e seções CDATA. Esses diferentes tipos de marcação serão brevemente descritos a seguir.

Elementos

Os elementos são os blocos de montagem da linguagem, sendo as partes de um documento. Há um nó elemento para todo elemento em um documento [W3C99a]. O conteúdo dos nós elementos pode ser composto de outros nós elementos aninhados, nós comentários, nós de instruções de processamento e de nós texto. A sintaxe de um nó elemento está representada na tabela abaixo:

Tabela 2.2 – Expressão para a criação de elementos.

<nome atrib$_1$="val$_1$" ...atrib$_n$="val$_n$"> conteúdo </nome> ou para um nó elemento vazio:

A sintaxe de um nó elemento é definida por um nome ou tipo do elemento, delimitada pelos marcadores de início e fim, "<" e ">" respectivamente. O conteúdo dos elementos, quando contém outros aninhados, forma uma lista ordenada de nós filhos.

Um tipo especializado de elemento, o *doctype*, possui um *nome* e uma URL opcional (*url*) para a utilização de uma definição de documento externo. A intenção da URL é especificar um esquema especializado para a estrutura do documento. A Tabela 2.3 demonstra como um *doctype* é definido.

Tabela 2.3 – Expressão para criação de doctype

<!DOCTYPE nome do nó raiz [DTD]> ou externo privado e público, respectivamente <!DOCTYPE nome do nó raiz SYSTEM "url">

A declaração de DTD (Subseção 2.4.1) para um documento XML pode ser realizada dentro do *doctype*, conforme a figura acima. Outro tipo de declaração consiste em referenciar um DTD externo indicando a sua URL na *url*. Declarações de DTDs externos podem ser de dois tipos: privadas ou públicas. DTDs externos privados são identificados pela palavra chave SYSTEM e os públicos pela PUBLIC. Também é possível declarar uma combinação DTDs internos e externos.

Atributos

Os elementos podem conter atributos, os quais são conjuntos de pares de nomes e valores (Tabela 2.2), formando uma lista não ordenada de nós folhas. Através de sua utilização é possível transmitir mais informações sobre um elemento, conseguindo descrever suas propriedades mais claramente. Os valores dos atributos podem ser restringidos a certos tipos de dados através de DTD ou XML Schemas, os quais serão abordados na Seção 2.4.

O grupo de trabalho responsável pela XML definiu alguns nomes de atributos reservados para finalidades específicas. Exemplos de restrições aos nomes dos atributos são: o atributo *id*, reservado para o identificador do elemento, e nomes de atributos iniciados com o prefixo *xml*, declarado tanto em maiúscula quanto em minúscula.

Instruções de processamento

Uma instrução de processamento é um contêiner de informações específicas para um dado processador XML. A sua sintaxe é formada por uma instrução delimitada pelos marcadores "<?" inicial e "?>" final, ficando: <?*instrução*?>.

A *instrução* da sintaxe acima consiste de um par: objetivo/valor. Objetivos são as palavras-chave utilizadas pelos *parsers* XML, as quais determinam as instruções que devem ser processadas. Se não houver um valor, a própria palavra-chave de destino pode funcionar como dado, por exemplo, uma instrução para uma quebra de página <?lb?>.

Instruções de processamento iniciadas com *xml* são reservadas pela recomendação, podendo ser utilizadas em outros padrões derivados de XML. Um exemplo é a instrução

de processamento utilizada na especificação de folhas de estilo para documentos XML: *<?xml-stylesheet href="mystyle.css" type="text/css"?>*.

Comentários

Comentários são similares as instruções de processamento, mas não são interpretados pelos parsers XML. Enquanto as instruções de processamento são usadas para incluir notas explanatórias para alguma aplicação ou *parser* XML, os comentários são usados para incluir notas explanatórias para o entendimento humano [BOS97].

A sintaxe do nó comentário possui somente um conteúdo *c*, ignorado pelas aplicações, entre os delimitadores "<!--" e "-->", ficando: <!--*c*-->

Seções CDATA

Estas seções indicam aos parsers XML que devem ignorar os caracteres de marcação contidos dentro delas. São úteis na utilização de caracteres reservados como < e > dentro de um documento. A sua sintaxe é: <! [CDATA[*conteúdo_ignorado*]]>.

2.4 Validação de classes de documentos XML

A adoção da linguagem XML cresce rapidamente. Vocabulários estão sendo definidos para representarem aparentemente todos os tipos de domínios dentro do mundo XML. Com isto, a capacidade de validar documentos XML está se tornando um aspecto importante para troca de informações entre aplicações. A estrutura de um documento XML é validada através de mecanismos de esquemas XML.

Existem atualmente dois mecanismos de validação recomendados pela W3C, são eles: *Document Type Definition* (DTD) e XML Schema. A W3C não é a única fonte de linguagens de esquemas XML. Outras propostas foram desenvolvidas, como a XML-Data da Microsoft. A XML-Data foi utilizada na iniciativa BizTalk® para descrever documentos de mensagem de negócios. Este padrão foi substituído pela recomendação W3C XML Schema [MSDN04a].

Apesar da existência destas outras linguagens, este trabalho restringe-se abordar apenas as linguagens da W3C: DTD e XML Schema.

A recomendação W3C XML 1.0 [W3C04b] define os conceitos: validade e bem formado. Um documento está bem formado quando a sua sintaxe estiver de acordo com a sintaxe estabelecida pela recomendação. E, um documento só é válido, se estiver de acordo com as restrições expressas em um esquema XML a ele associado.

Um esquema é um modelo para descrever a estrutura da informação [WALSH99], descrevendo os arranjos possíveis entre marcações e texto em documentos XML. Pode também ser visto como uma definição de um vocabulário comum para troca de documentos entre aplicações específicas.

Os esquemas são descritos em termos de restrições. As restrições definem o que pode aparecer em qualquer contexto, podendo ser de dois tipos:

1. Restrições de modelo de conteúdo: descrevem a ordem e a seqüência em que os elementos aparecem em documento. Os testes de validade são realizados verificando-se a ordem e o aninhamento dos elementos dentro de um documento.
2. Restrições de tipos de dados: descrevem as unidades de dados válidas nos documentos. A validação desse tipo de restrição é realizada verificando se unidades específicas de informações são do tipo de dados correto e se seus valores estão dentro da faixa legal especificada [WALSH99].

Embora DTDs e XML Schemas usem diferentes sintaxes e possuam diferentes abordagens para a tarefa de descrever estruturas de documentos, ambas visam a mesma solução [LAURENT99]. XML Schema é vista como substituta natural do DTD, mas essa transição ainda permanece indefinida, não se sabendo se ocorrerá rapidamente e de maneira efetiva. DTDs estão fortemente presentes em ferramentas comerciais atuais, enquanto que XML Schemas ainda está sendo implementada ou permanece como planos futuros em várias outras.

2.4.1 Document Type Definition (DTD)

Na realidade, DTDs são mais antigas que XML, sendo um prosseguimento da SGML, com a sua sintaxe quase completamente intacta [RAY01]. DTD XML fornece uma maneira de validar a sintaxe e alguma semântica de um documento XML [OMG02b]. Um documento XML possui tanto uma estrutura lógica quanto uma estrutura

física; através de um DTD XML torna-se possível definir restrições em ambas as estruturas. Uma vez definida uma DTD, documentos XML que a referenciam devem obedecer ao vocabulário comum definido e as restrições impostas por ela.

Um DTD pode definir as seguintes restrições:

- elementos obrigatórios e proibidos;
- os elementos aninhados, definindo a estrutura hierárquica do documento;
- atributos obrigatórios e opcionais de um determinado elemento;
- valores de atributos válidos;
- identificadores de elementos dentro de um documento (atributo de identificação);
- referências em uma parte de um documento a um elemento definido em uma outra parte (incorporação por referência);
- segmentos compartilhados de especificação DTD internos ou externos incorporados ao DTD (entidades).

DTD externo pode ser referenciado dentro de documentos, incorporando-o a partir de uma origem comum. A incorporação de um DTD através de uma origem comum a vários documentos fornece a base para o compartilhamento de documentos com uma única definição ou vocabulário. Os DTDs são usados para definirem vocabulários XML. Um vocabulário é uma especificação compartilhada da sintaxe e da estrutura dos documentos de uma determinada área de interesse.

A sintaxe de uma declaração DTD deve sempre iniciar com o delimitador "<!", seguido de um marcador básico, obrigatoriamente escrito com letra maiúscula, como, por exemplo, ENTITY. A declaração deve ser encerrada com o sinal ">". A descrição dos objetos pertencentes ao documento XML é definida num DTD. Os elementos utilizados em um documento XML necessitam ser declarados num DTD externo ou interno.

Dois tipos de declarações serão destacados devido a sua importância em DTD. O primeiro tipo de declaração é a especificação de um elemento de um documento XML em DTD. A sintaxe de um elemento em DTD começa com a string "<!ELEMENT", seguida do nome do elemento no documento, um modelo de conteúdo e o delimitador de

fechamento. No modelo de conteúdo encontra-se a sintaxe que expressa as informações sobre o conteúdo que o elemento suporta. O conteúdo pode ser composto de dados e elementos filhos, podendo este ser definido em cinco tipos diferentes de modelos de conteúdo de elementos: vazios, sem restrição de conteúdo, contendo apenas dados de caracteres, contendo apenas outros elementos e com conteúdo misto.

Elementos podem ser definidos como grupos de sub-elementos ou sub-grupos. A ordem em que os sub-elementos ou subgrupos são apresentados em um documento XML é balizada através de dois delimitadores disponibilizados pela sintaxe DTD. O delimitador de grupo "," descreve uma seqüência obrigatória de elementos, funcionando como um operador *AND* nas linguagens de programação. Já o delimitador "|" assemelha-se ao operador *OR*, podendo os elementos aparecerem independente da ordem em que foram declarados. Operadores de cardinalidade podem ser adotados na declaração de elementos. Os operadores são: zero ou um (?), um ou mais (+) e zero ou mais (*).

O texto encontrado entre a tag de início e de final de um elemento é citado como dados de caracteres em XML. PCDATA são os dados de caracteres que necessitam ser analisados por um analisador XML. CDATA também significa os dados de caracteres, porém esses não serão tratados por um analisador.

O segundo tipo destacado nesta seção é o atributo. Os atributos, quando definidos por uma DTD, oferecem meios de associar propriedades simples a elementos, conseguindo completá-los e modificá-los. A sintaxe de um atributo consiste na string <!ATTLIST seguida do nome do elemento que receberá o atributo. Logo após, é declarado o nome do atributo, juntamente com o seu tipo e declarações padrões. As declarações padrões são utilizadas para definir valores fixos (#FIXED), quando os valores não precisam ser incluidos (#IMPLIED) ou quando o valor é requerido (#REQUIRED). Os valores fixos são definidos juntamente com os seus valores. Quando o valor de um atributo necessita ser um dos valores pertencentes a um conjunto definido pode-se utilizar uma lista, por exemplo, (valor1, valor2), contendo todos os valores válidos e, além disso, pode-se definir um valor padrão inicial para o atributo declarando-o após a lista. Uma declaração de atributo é exemplificada na linha três da Tabela 2.4.

Um exemplo de DTD é mostrado na Tabela 2.4.

Tabela 2.4 – Exemplo de DTD

```
1   <!ELEMENT notafiscal (cliente,produto+)>

2   <!ELEMENT cliente (nome,endereco*)>

3   <!ATTLIST cliente
            cpf_cnpj CDATA #REQUIRED
            preferencial (sim|não) #IMPLIED
            origem CDATA #FIXED “Brasil”
            tipo (fisica|juridica) “fisica”
    >

4   <!ELEMENT nome (#PCDATA)>

5   <!ELEMENT endereco (#PCDATA)>

6   <!ELEMENT produto (marca, fabricante?)>

7   <!ELEMENT marca (#PCDATA)>

8   <!ELEMENT fabricante (#PCDATA)>
```

Infelizmente, a especificação DTD não obedece à sintaxe XML, ou seja, os DTDs não são documentos XML. Além disso, não é capaz de definir tipos de dados, possui extensibilidade limitada e é marginalmente compatível com *namespaces*, dentre outras deficiências contidas no DTD. Estas deficiências foram corrigidas na recomendação W3C XML Schema.

2.4.2 XML Schema

XML Schema é uma recomendação W3C desde maio de 2001, sendo composta de três documentos: Parte 0: *Primer* – documento não normativo que fornece uma fácil descrição da XML Schema; Parte 1: *Structures* – especifica a definição da linguagem XML Schema; e Parte 2: *Datatypes* – determina meios para definir tipos de dados usados na XML Schema, assim como em outras especificações XML.

A finalidade de um documento XML Schema é definir uma classe de documentos XML. [W3C01a]. A linguagem trouxe inúmeras vantagens e avanços em relação ao DTD, entre elas pode-se citar:

- Obediência à sintaxe XML; sendo desta forma extensível e permitindo declarações de estruturas internas mais ricas e complexas.
- Tipos de dados orientados a dados em comparação aos dos DTDs orientados a documentos [W3C01b].
- Fornecimento de um controle aprimorado sobre o conteúdo dos elementos.
- Capacidade de definição de tipos de dados complexos, por exemplo, estruturas de entidades reutilizáveis.
- Capacidade de definição de relacionamentos entre documentos utilizando chaves, semelhante aos modelos relacionais convencionais.

O documento XML Schema, assim como DTD, pode ser referenciado dentro de um documento XML. Assim, torna-se possível definir um único esquema que será compartilhado por muitos documentos na validação.

Existem dois tipos básicos de elementos encontrados na linguagem: elementos simples e elementos complexos. Um elemento simples pode ser definido com a utilização da marcação *simpleType*, utilizado normalmente quando se deseja impor alguma restrição ao elemento. As declarações de tipos complexos exigem a utilização de *complexType*. A Tabela 2.5 exemplifica a declaração de um elemento simples e um composto.

Tabela 2.5 – Declaração de elementos simples

```
1   <xsd:simpleType name= “notafiscal”>
2   <xsd:restriction base= “xsd:positiveInteger”>
3   <xsd:maxExclusive value= “100”/>
4   </xsd:restriction>
5   </xsd:simpleType>
6   <xsd:complexType name=”tipoEndereco” >
7   <xsd:sequence>
8   <xsd:element name= “rua” type= “xsd:string”>
9   <xsd:element name= “numero” type= “xsd:decimal”>
10  </xsd:sequence>
11  </xsd:complexType>
```

Dois atributos são disponibilizados para a definição de cardinalidade na declaração de elementos. O atributo *minOccurs* é responsável por indicar o número mínimo de vezes que um elemento deve aparecer. Quando esse atributo é omitido é assumido que a quantidade mínima de vezes que o elemento pode aparecer é um. O atributo responsável pela indicação do número máximo de vezes que um determinado elemento pode aparecer é o *maxOccurs*. Quando não está definido, a quantidade máxima o valor *unbounded* deve ser atribuído. Se esse atributo for omitido da declaração entende-se que esse elemento deve aparecer no máximo uma vez no documento.

A XML Schema possui três delimitadores de grupos para a definição da ordem dos elementos dentro de documentos XML. O delimitador *sequence* informa que os elementos devem aparecer na seqüência apresentada no documento XML Schema. Já o

choice informa que apenas um dos elementos declarados deve aparecer. Por fim tem-se o delimitador *all,* informando que todos os elementos declarados, podem aparecer nenhuma ou uma vez no documento.

A capacidade de especificar tipos de dados tanto para elementos quanto para atributos consiste em outra vantagem da XML Schema, permitindo, além disso, a união, criação de uma lista e derivação por extensão e por restrição de tipos. A linguagem também permite um maior controle sobre os tipos enumerados.

2.5 XML Namespace

Um único documento XML pode conter elementos e atributos definidos e usados por múltiplos módulos de software [W3C04a], podendo ocorrer colisões de nomes e problemas de reconhecimento pelos módulos responsáveis pelo processamento. Este exemplo evidencia a necessidade de um mecanismo de nomes, estendendo o escopo além do documento ao quais os elementos e atributos pertençam. Este mecanismo, dentro da W3C, é o *namespace*.

Em janeiro de 1999, uma recomendação chamada *Namespace in XML* [W3C04a] adicionou *namespace* a linguagem XML, possibilitando a nomeação exclusiva de elementos e atributos. A recomendação encontra-se na versão 1.1 desde fevereiro de 2004.

Namespace é uma coleção de nomes, identificados por uma referência URI, utilizada nos documentos XML em tipos de elementos e de atributos [W3C04a] em determinado domínio ou finalidade.

Cada nome contém um significado específico dentro de um contexto ou domínio. Nomes iguais em elementos ou atributos podem aparecer dentro de uma organização possuidora de vários documentos, sem que estes tenham o mesmo significado. Elementos ou atributos com nomes iguais, mas com diferentes significados, podem ser tratados através de *namespace*. A sua utilização evita que as organizações usem o mesmo nome a elementos de diferentes propósitos.

Os *URIs* utilizados na declaração *namespace* proporcionam exclusividade baseada na atribuição global de nomes de domínio da Internet. O gerenciamento dos nomes utilizados para que não haja conflito fica a cargo dos proprietários do domínio específico. Para garantir consistência, organizações devem decidir sobre estratégias de *namespace* e convenção de nomes, quando estabelecerem um *namespace* organizacional [GLACE03].

Dentro de um documento XML, o URI do *namespace* é associado a um prefixo, por exemplo: <Ordem xmlns:cenpra=http://www.cenpra.gov.br/exemplo-ns>. Nesse caso, o URI está associado ao prefixo *cenpra* do elemento *Ordem*. Os nomes de elementos dentro do elemento *Ordem* podem ser qualificados usando o prefixo de *namespace cenpra*, por exemplo: <cenpra:OrdemItem />.

Na XML, o URI não precisa fazer referência a um recurso Web real, servindo apenas como um identificador exclusivo para fazer a diferenciação em homônimos. No entanto, é útil que o URI identifique uma página Web que descreva o *namespace* e forneça um DTD ou XML Schema para as estruturas associadas.

2.6 Considerações Finais

Assim como a meta-linguagem SGML, a XML não possui nenhum mecanismo formal para suportar declaração de restrições de integridade semântica. *Parsers* XML não têm maneiras de validar a semântica dos objetos, mesmo que estas estejam declaradas informalmente em um DTD XML [COVER98]. XML pode ajudar os humanos predizerem qual informação deve estar situada entre as marcações, como no caso de *<Pedido>*, porém para um *parser*, *<Pedido>*, *<i>* e *<Capítulo>* são totalmente iguais e totalmente sem significado semântico.

Apesar das marcações XML possuírem um propósito semântico, se comparadas ao propósito de *layout* da linguagem HTML, XML continua sendo uma linguagem de marcação principalmente focada na sintaxe. A semântica das marcações está tipicamente codificada nas aplicações de envio e recebimento que manipulam estes documentos XML [ERDMANN00].

Várias atividades recentes buscam adicionar mais capacidade semântica aos documentos XML. Exemplo de trabalho nesta área é o modelo padrão para a descrição de

semântica dentro da W3C é o Resource Description Framework (RDF) [PATEL-SCHNEIDER02].

XML não é rica o suficiente para capturar informações semânticas de alto nível, por não fornecer um modelo de objetos verdadeiros, não prover múltiplos níveis de abstração e de não trazer sozinha a semântica dos seus elementos. Isto não a torna uma solução ótima para modelar metadados através de vários domínios diferentes. Para este tipo de solução, é necessário um conjunto de construções de alto nível para a construção de modelos de qualquer tipo de metadados [MOSHER02].

3 METADADOS E PADRÕES DE INTEROPERABILIDADE DE MODELOS.

Este capítulo oferece um embasamento teórico, elucidando os principais conceitos e padrões de interoperabilidade de modelos. Serão abordados conceitos sobre metadados e arquitetura de modelos de quatro camadas, fundamentais para a compreensão da proposta deste trabalho. O padrão de interoperabilidade *XML Metadata Interchange* (XMI) da OMG será elucidado nos capítulos subsequentes.

3.1 Introdução

A partir do momento em que dois computadores foram conectados e conseguiram se comunicar, estes deixaram de ser apenas máquinas de computar ou calcular e tornaram-se dispositivos de comunicação e colaboração. O desenvolvimento e aprimoramento de novos padrões de tecnologia na área de telecomunicações e de redes de computadores nas últimas décadas tornaram possíveis construções de redes confiáveis de alcance geográfico mundial com altas taxas de transmissão de dados, seja por meio físico ou por redes sem fio. Avanços alcançados em hardware e em redes de computadores trouxeram como conseqüência o aprimoramento da conectividade dos sistemas de software que, juntamente com a tecnologia de microcomputadores, possibilitaram chegar ao nível de conectividade encontrada hoje, culminando no atual fenômeno da popularização da Internet e das redes locais corporativas.

A Internet e a Web romperam o clássico cenário cliente-servidor, onde existiam vários clientes acessando apenas um servidor dentro do mesmo ambiente da empresa. Esse cenário deixa de existir, transformando-se em milhões de servidores distribuídos geograficamente em milhões de empresas. Futuramente, todo dispositivo eletrônico estará conectado na Internet e disponibilizará os seus dados armazenados, tornando-se também um servidor com inúmeros serviços.

A Internet acelerou as expectativas dos usuários em se obter acesso sem limites a recursos de informações e realizar troca de dados de forma transparente entre aplicações [ECKERSON99]. Entre as empresas, criou-se a necessidade de que estas explorassem a

Web, não apenas para a difusão das informações, mas também para aumentar a interação e a integração com os seus clientes, parceiros comerciais, fornecedores e distribuidores através de sistemas de software.

Raramente os sistemas de software vivem isolados, necessitando geralmente de se comunicarem com outros sistemas já existentes. Diferentes sistemas geralmente possuem diferentes tecnologias e padrões de representação de dados.

Wegner [WEGNER96] definiu interoperabilidade como sendo a capacidade de dois ou mais componentes de software cooperarem entre si, apesar de diferenças em linguagens, interfaces e plataformas de execução. Interoperabilidade também foi definida [IEEEa], como "a habilidade de dois ou mais sistemas ou componentes trocarem informações e as usarem." Nota-se que interoperabilidade, de acordo com essa última definição, não é simplesmente a troca de informações entre dois sistemas, mas também a utilização da informação que está sendo trocada. Para isto, é necessário que as aplicações consigam ler e entender dados representados em diferentes padrões de representação. A quantidade de informação que pode ser compartilhada entre duas aplicações é limitada pela quantidade que pode ser entendida por ambas as aplicações simultaneamente.

Atualmente encontra-se um ambiente heterogêneo formado por diferentes tipos de plataformas e aplicações conectadas que sofrem restrições impostas pelos diferentes formatos de dados proprietários. Estas restrições limitam o acesso irrestrito às informações: é evidente a necessidade de se conseguir ler e entender dados de outras aplicações, definidos com diferentes semânticas, estruturas e sintaxes. Esta capacidade permitiria atingir um alto grau de integração e interação entre aplicações diferentes, viabilizando uma livre troca de informações através das aplicações.

Para isto, a padronização de um modelo de representação de dados comum é um ingrediente crucial para o compartilhamento de informações, sendo de grande utilidade em áreas tão distintas quanto engenharia de software, troca de informações entre ferramentas ou aplicações, portais corporativos e *data warehousing*.

3.2 Metadados

Metadados são definidos como informação sobre dados ou dados sobre dados. Se procurarmos o significado do prefixo *meta*, utilizado com freqüência neste trabalho, encontra-se, entre outros, o seguinte: "conceito que se auto-explica" [BABYLON02]. Uma das primeiras utilizações do termo metadado foi nos esquemas de banco de dados. Esquemas ou modelos de banco de dados são informações que descrevem o formato de um registro, como, por exemplo, tipo, nome e tamanho de cada uma das colunas de uma tabela num banco de dados relacional.

O uso de metadados tornou-se, nos últimos anos, o centro de atenções em diversas áreas, com a concretização de sua importância no uso efetivo da Web. Os metadados podem ser aplicados, por exemplo, na busca de informações mais precisas, na integração de dados ou informações de fontes heterogêneas e no refinamento de consultas a base de dados, conseguindo com que estas retornem realmente o que o usuário deseja. Metadados são anexados aos dados, ajudando na sua interpretação.

Metadados são utilizados para descrever as propriedades e o significado dos dados para melhor entendê-los, utilizá-los e gerenciá-los, em outras palavras, os metadados definem a sintaxe e a semântica dos objetos de dados das aplicações, permitindo que estas saibam como processar solicitações e tarefas. Através da sua definição, a troca de informações entre diferentes aplicações pode ser realizada.

Como cada aplicação utiliza a sua própria solução, o problema de heterogeneidade de metadados continua sendo motivo de preocupação. Isto realça o fato do problema estar na incompatibilidade e não na carência destes. A carência de um padrão comum para representar e compartilhar metadados conduziu a grandes dificuldades em compartilhar até mesmo de dados simples, prejudicando a integração e interoperabilidade de aplicações complexas, componentes e sistemas.

Metadados incompatíveis continuarão causando grandes problemas de interoperabilidade. O que é necessário para interoperabilidade irrestrita é um tipo de linguagem universal, capaz de descrever todos os tipos de metadados [BAKER97]. Esta solução esbarra na dificuldade em se criar uma linguagem capaz de descrever metadados

de todas as áreas existentes. Visto que cada aplicação necessita e possui metadados de propósitos próprios e específicos, vários foram criados, sendo a maioria destes, embutidos nos sistemas, tornando o seu gerenciamento extremamente difícil de ser realizado.

Inicialmente, sistemas distribuídos de todos os tipos foram tratados em uma base adaptada ou proprietária, usando sistemas de mensagens privados para cominicação entre processadores sobre protocolos de redes de baixo nível. Estes sistemas, que utilizavam tecnologias privadas e específicas, foram substituídos pelos *middlewares*. Os *middlewares* são sistemas de propósitos gerais que provêem serviços independentes de sistema operacional ou linguagem de programação. Em sistemas distribuídos, a representação de metadados é utilizada com freqüência devido ao aumento de heterogeneidade e de abertura (*openness*), i.e., oferecimento de serviços de acordo com um conjunto de regras que descrevem a sintaxe e a semântica dos serviços [CRAWLEY97]. O fato da informação dos metadados ser completa o bastante, possibilitou o seu uso em integração e interoperabilidade de sistemas de informação distribuídos [JEFFERY98].

3.3 Arquitetura de metadados de 4 camadas

Uma técnica desenvolvida pelos seres humanos para lidar com problemas complexos é a abstração. Seres humanos, quando incapacitados de dominar a totalidade de objetos complexos, escolhem simplesmente ignorar os detalhes não essenciais, lidando assim, apenas com o modelo generalizado e idealizado do objeto [SHAW81].

Baseado neste conceito, para cada objeto complexo encontrado, pode-se criar modelos descritos com diferentes níveis de detalhamento de informação, ou seja, modelos pertencentes a níveis de abstração diferentes.

Como se mostra na Figura 3.1, a partir de um modelo num nível mais refinado e, com isto, rico em detalhes, pode-se ignorar os detalhes não essenciais, derivá-lo a um nível mais alto de abstração.

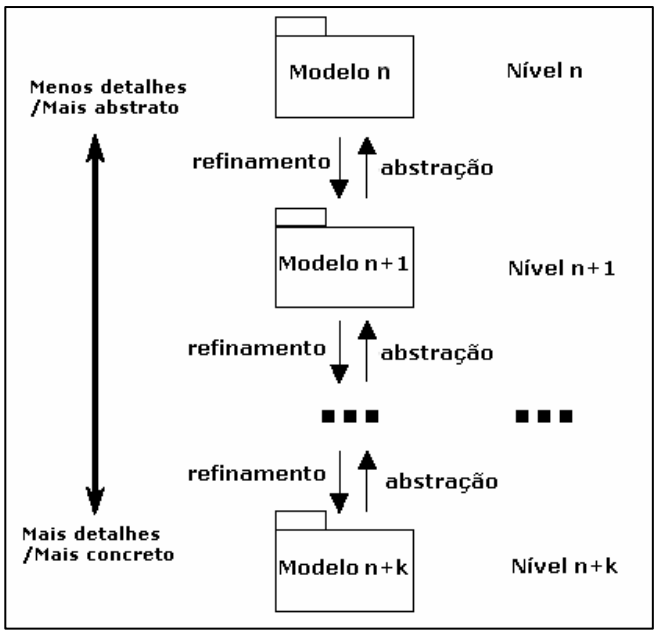

Figura 3.1 – Níveis de Abstração /Refinamento

O termo *meta* é usado para descrever o modelo de um certo nível e os elementos que o compõem [ATKINSON97]. Desta forma, um modelo de objetos pertencente a um nível de abstração *n+1* é definido através do uso das construções ou elementos pertencentes ao modelo de nível *n*, chamado de metamodelo. Os elementos pertencentes ao modelo mais abstrato são prefixados pelo termo *meta*. Estes elementos podem ser vistos como a sintaxe abstrata capaz de descrever os elementos pertencentes ao modelo de nível de abstração inferior.

Como exemplo, um modelo B descreve um outro modelo, A, sendo assim, o metamodelo do modelo A. O metamodelo B está em um nível maior de abstração do que o modelo A, podendo o modelo A também ser visto como uma instância do metamodelo B. Um outro modelo C poderia ser usado para definir a sintaxe abstrata do metamodelo B. Este modelo C seria então chamado meta-metamodelo.

Arquiteturas de metadados estão sendo introduzidas para lidar com o problema de heterogeneidade entre os diversos metadados de diferentes sistemas e plataformas, a fim de se obter interoperabilidade e portabilidade. Estas arquiteturas permitem que um mesmo modelo seja realizado em múltiplas plataformas através padrões de mapeamentos específicos melhorando portabilidade das aplicações.

Arquiteturas de metadados permitem descrever metadados em diferentes níveis de abstração. O número de meta-níveis ou níveis de abstração é arbitrário, sendo um *framework* clássico de meta-modelagem baseado em uma arquitetura de quatro meta-camadas [CDIF94]. A hierarquia de níveis de abstração é interrompido tornando a última camada auto-descritiva.

Na figura 3.2, as camadas são descritas da seguinte maneira:

- A camada de informação é formada pelos dados e objetos reais, instâncias da camada de modelo.
- A camada de modelo é constituída de modelos que descrevem os dados que estão na camada de informação. Estes modelos são também chamados de metadados.
- A camada de metamodelo é composta de modelos de informações sobre os metadados (modelos) da camada inferior; estes modelos são por isso designados de meta-metadados ou metamodelos. Os metamodelos descrevem os metadados da camada de modelo, sendo a linguagem abstrata capaz de descrever os diferentes tipos de metadados.
- A camada de meta-metamodelos é populada pelos modelos de informações sobre os metamodelos. Em outras palavras, é a linguagem abstrata capaz de definir diferentes tipos de metamodelos. Geralmente, os meta-metamodelos são reflexivos, i.e., eles se auto-descrevem, não havendo necessidade de um nível a mais de abstração.

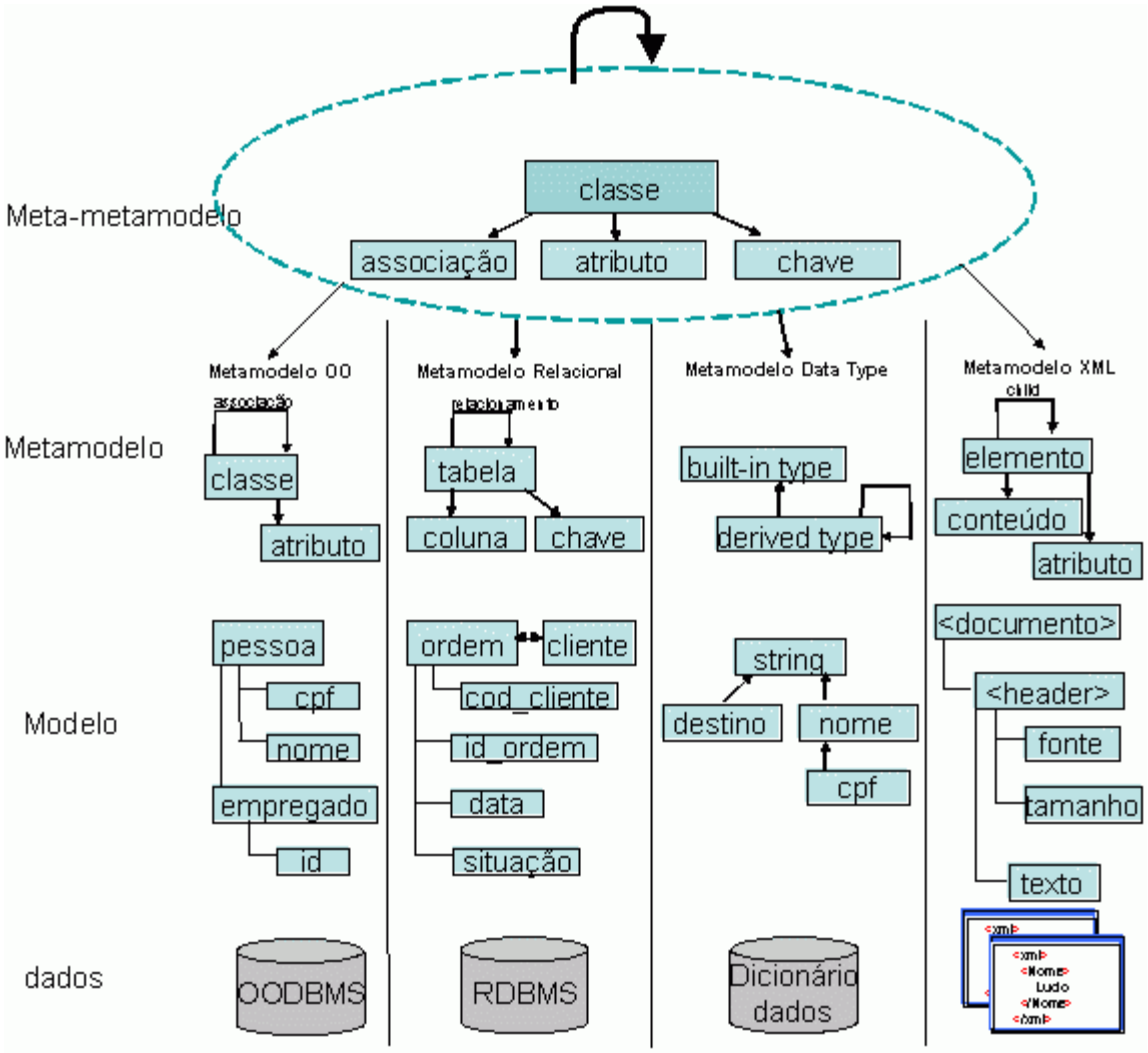

Figura 3.2 – Arquitetura de metadados de quatro camadas

Esta arquitetura proporciona várias vantagens como: suportar diferentes tipos de metamodelos e permitir que novos tipos sejam adicionados, permitir que diferentes tipos de modelos sejam relacionados e suportar o intercâmbio de modelos e metamodelos arbitrários entre partes que usam o mesmo meta-metamodelo. O padrão *Meta Object Facility* (MOF) [OMG02a] adotado pela OMG é baseado na arquitetura de quatro camadas. O MOF é um *framework* genérico de metadados capaz de descrevê-los e gerenciá-los.

3.4 Padrões de intercâmbio de modelos

Dada a importância da utilização de metadados para descrever a estrutura (sintaxe) e o significado (semântica) das informações, as tentativas de prover a sua troca e um

acesso irrestrito culminaram na criação de diferentes metadados para cada domínio específico. Dificuldades no gerenciamento de metadados limitaram a criação de aplicações integradas e interoperáveis. O gerenciamento de metadados pode ser solucionado, definindo-se um modelo padrão, capaz de representar diferentes tipos de metadados, ou seja, uma forma padrão de modelar metadados em diferentes níveis de abstração.

Este modelo padrão precisa ter definições precisas do significado das características e dos atributos das instâncias dos metamodelos, permitindo mapeamentos destas características para linguagens e formatos de intercâmbios específicos.

Formatos de intercâmbio precisam ser independentes de plataforma e da forma como são trocados, podendo ser lidos por diferentes ferramentas, implementados com tecnologias distintas. MOF provê uma linguagem capaz de descrever metadados pertencentes a diferentes níveis de abstração, além de fornecer mapeamentos para tecnologias específicas permitindo o gerenciamento e o intercâmbio de metamodelos e modelos.

Em dezembro de 1997, a OMG publicou uma solicitação de propostas (RFP) sobre um formato para intercâmbio de modelos, chamada *Stream-based Model Interchange* (SMIF). A RFP solicitava propostas para uma especificação completa da sintaxe e da codificação necessárias para exportar ou importar modelos e metamodelos compatíveis com o padrão de metamodelagem MOF.

A proposta deveria usar o MOF como seu meta-metamodelo e prover um modo para a identificação única de qualquer conceito especificado em um metamodelo referenciado no *stream* de transferência [OMG97].

Três submissões iniciais relativas à RFP foram recebidas: 1) *Daimler-Benz Research and Technology* e *Recerca Informàtica* 2) *Fujitsu/Softeam*; e 3) um grupo industrial liderado por DSTC, IBM, *Oracle*, *Platinum Technology* e *Unisys*. A proposta enviada pelo terceiro grupo foi aceita como padrão pela OMG devido ao uso central do XML como a sintaxe padrão SMIF, fato não encontrado nas outras duas propostas.

Nesta seção serão apresentadas as propostas enviadas a OMG, relacionadas com a RFP SMIF, chamadas *Universal Object Language* (UOL) e *CASE Data Interchange Format* (CDIF), ficando a *XML Metadata Interchange* (XMI) para o capítulo seguinte.

Apesar de não estarem relacionadas com a RFP SMIF, também serão apresentados os padrões de arquiteturas de metadados *Resource Description Framework* (RDF) e *Open Information Model* (OIM). A apresentação desses padrões visa mostrar diferentes abordagens de interoperabilidade de metadados fora do contexto da OMG.

3.4.1 Universal Object Language

As empresas *Recerca Informàtica* e *Daimler-Benz Research and Technology* juntas enviaram uma proposta à OMG, respondendo a solicitação SMIF. A proposta visava estabelecer um formato padrão para o intercâmbio de modelos baseados em MOF através da especificação da linguagem *Universal Object Language* (UOL), dos seus XML-DTDs e dos mapeamentos: *UOL-MOF, UOL-CDIF e UOL-STEP/EXPRESS*.

UOL é uma linguagem de ciclo de vida completo orientada a objetos. Linguagem de ciclo de vida completa orientada a objetos é uma linguagem de engenharia de objetos, capaz de descrever todas as construções e conceitos da Análise e Projeto Orientados a Objetos (OOAD[4]), sendo conceitualmente executável [OMG98].

A linguagem UOL é uma transição da linguagem Eiffel+. Essa transição consiste na expansão do suporte à elementos UML [OMG03a] e na adição de várias palavras reservadas, buscando trazer facilidades na leitura dos documentos.

Eiffel+ foi finalizada em 1995, na tese de doutorado de Allen Peralta, um dos co-autores da proposta UOL. Eiffel+ consiste na idéia de uma linguagem textual capaz de descrever todos os elementos de OOAD e, com isto, obter diferentes produtos em cada fase de desenvolvimento. A escolha por uma linguagem textual é defendida pelo fato das linguagens gráficas não serem adequadas para todos os tipos de tarefas e pessoas, tornando necessária uma linguagem textual para esses casos.

---

[4] *OOAD - Object-oriented Analysis & Design: é uma metadologia de desenvolvimento de software orientados a objetos. [PRESSMAN]*

Os argumentos para o desenvolvimento de uma nova linguagem textual baseada na linguagem Eiffel[5] foram, segundo os autores da proposta UOL, a dificuldade de se introduzir uma nova linguagem orientada a objetos (OO) à comunidade e as características existentes na Eiffel.

Apesar das vantagens da XML serem reconhecidas pelos desenvolvedores da proposta, a linguagem não foi indicada como solução direta para a sua sintaxe. Os motivos descritos para isso foram: 1) XML não foi desenvolvida para ser usada intensivamente por humanos sem a ajuda de ferramentas específicas, desta forma, não é adequada para comunicações genéricas entre humanos e ferramentas ou entre humanos; 2) a não aceitação da linguagem XML como uma *round-trip engineering language* pela dificuldade em usá-la sem editores sintáticos.

Apesar de não adotarem XML como a sintaxe padrão, a proposta utiliza-a na publicação de partes do repositório e no desenvolvimento dos XML-DTDs necessários. A sua utilização possibilita o acesso ao repositório via XML e também o seu uso por ferramentas e aplicações que o suportam.

UOL é um formato legível a humanos que permite representar modelos UML e metamodelos MOF de maneira compacta, suportando a transferência entre qualquer par de ferramentas em *batch* ou em tempo real [OMG98]. A transferência de informações, na realidade texto, entre as ferramentas é facilmente realizada, necessitando apenas de um *parser* para traduzir a informação em seu formato particular para o formato do padrão. UOL pode também ser facilmente gerada para as informações contidas no repositório.

A sua semântica rica permite criar representações compactas, porém ainda legíveis. Ser legível para humanos é uns dos principais objetivos da proposta por vários motivos [OMG98]: 1) a saída de uma ferramenta em várias situações precisa ser analisada por

---

[5] *Eiffel pode ser visto como um framework para o desenvolvimento e implementação de software orientado a objetos. Não podendo ser considerada apenas uma linguagem de programação no sentido restrito de implementação, devido ao fato de cobrir completamente a área de desenvolvimento de software. Eiffel é uma marca registrada de NICE (The Nonprofit International Consortium for Eiffel).*

humanos, antes de ser inserida em outra ferramenta; 2) o uso de outras ferramentas além das *Computer Aided Software and Systems Engineering* (CASE), por exemplo, processadores de texto ou compiladores, necessitam também de um formato de transferência válido. Sendo o modo texto o único formato legível pelo homem e aceito universalmente por qualquer ferramenta, o UOL assume este requisito como objetivo.

Um dos objetivos finais da proposta é a simplificação e otimização da transferência de modelos UML obtidos através da extensão da UOL básica, com construções adicionais da UML. Esta extensão é proposta como compatível, mas não mandatória. Com a extensão, uma representação textual UML é criada, sendo uma alternativa válida para a visualização de modelos UML em ferramentas não gráficas.

3.4.2 CASE Data Interchange Format (CDIF)

Em julho de 1998, a submissão CDIF foi publicada pela OMG como uma das respostas à solicitação RFP SMIF. Enviada pelas empresas *Fujitsu* e a *Softeam*, juntamente com a colaboração de outros parceiros comerciais.

Foram incluídas tecnologias adicionais ao padrão CDIF já existente, satisfazendo os requisitos solicitados pela OMG. Tais tecnologias são: atribuição de meta-identificadores permanentes a elementos do metamodelo que não estão no Metamodelo Integrado CDIF; suporte a estereótipos, restrições e mecanismos de extensão de *tagged values* em UML e regras para transformação isomórfica do kernel estático[6] do MOF para a arquitetura CDIF. Um mapeamento direto de vários elementos do kernel estático do meta-metamodelo MOF foi realizado para o meta-metamodelo CDIF, como, por exemplo, meta-meta-classes MOF em meta-entidades CDIF [ISO99].

Sendo CDIF um padrão já existente, demonstrado e implementado em algumas ferramentas de grandes fabricantes de software, a submissão destaca este fato a favor da sua adoção como formato padrão para intercâmbio de modelos da OMG. CDIF foi criado para modelos estruturados, necessitando adaptar-se para realizar intercâmbios de modelos

[6] O kernel estático MOF é um subconjunto próprio do meta-metamodelo MOF especificado em [OMG02a].

orientados a objetos. Esta adaptação traria economia de tempo no desenvolvimento de aplicações comerciais que utilizam esta tecnologia.

Introdução ao padrão CDIF

Iniciada pela *Electronic Industries Association* (EAI) em outubro de 1987, a Família de Padrões CDIF pretendia criar um padrão de transferência de modelos e métodos entre ferramentas CASE. Os principais desenvolvedores destas ferramentas e as organizações membros contribuíram para o desenvolvimento de uma família de padrões que após uma década contava com dez padrões em continuo progresso. A padronização em nível internacional, possuía a cooperação formal e informal da ANSI (X3L8, X3H4), ECMA (TC33:PCTE), IEEE (P1175), ISO (ISO IRDS, ISO/IEC JTC1/SC7/WG11) e OMG [FLATSCHER96].

A necessidade de interoperabilidade entre ferramentas CASE pode ser exemplificada por vários tipos de situações, como, o fato destas não conseguirem cobrir todo o ciclo de vida de desenvolvimento de software, criando a necessidade de várias ferramentas ou de se utilizar ferramentas com característica específica para o desenvolvimento de algum tipo de componente do sistema.

O fato da representação das informações contidas em uma ferramenta nunca serem totalmente iguais as exigidas pela outra, afeta a eficiência do desenvolvimento de um sistema. A Família de Padrões CDIF visava solucionar o problema de interoperabilidade entre estas ferramentas.

Então, com a ajuda do CDIF na realização de uma transferência, ferramentas CASE trocam modelos em diferentes fases de um ciclo de vida de desenvolvimento de software, bastando que estas possuam um comum acordo de utilização do padrão. O padrão CDIF facilita o sucesso da transferência de arquivos entre ferramentas CASE quando estas, mesmo não possuindo nada em comum, tenham um acordo comum de utilização do padrão [CDIF94].

CDIF é uma Família de Padrões que esboça uma arquitetura única para troca de informações entre ferramentas de modelagem e entre repositórios de modelos, definindo as interfaces de componentes necessárias para implementar esta arquitetura [ERNST97].

A Família de Padrões CDIF

CDIF fornece um conjunto de definições, independente de fornecedores ou métodos, para conceitos de metadados em geral, modelagem de dados e conceitos relacionados [ERNST97]. Além disto, define também um formato para representar modelos de maneira que possam ser transferidos.

A Família de Padrões CDIF pode ser dividida em três grupos distintos de documentos: 1) um definindo a arquitetura base; 2) outro definindo o formato de transferência, fornecendo as regras gerais, a sintaxe e a codificação para intercâmbio; 3) e um outro definindo o Metamodelo Integrado CDIF, determinando assim a informação que pode ser expressa e a sua semântica.

Algumas das especificações da família podem ser identificadas pelos respectivos documentos ISO [ISO98]

- **ISO/IEC 15474-1:199x, Information Technology — CDIF Framework — Parte 1: Overview:** explica a arquitetura CDIF completa e fornece uma visão geral de todos os padrões correntes que formam a família CDIF.
- **ISO/IEC 15474-2:199x, Information Technology — CDIF Framework — Parte 2: Modeling and Extensibility:** explica todo o escopo e a abordagem de modelagem em CDIF. Também define o meta-metamodelo e os mecanismos de extensibilidade do CDIF.
- **ISO/IEC 15474-1:199x, Information Technology — CDIF Transfer Format — Parte 1: General Rules for Syntaxes and Encodings:** define como CDIF suporta múltiplas sintaxes e codificações de troca. Descreve também como o meta-metamodelo é concretamente representado durante uma transferência.
- **ISO/IEC 15475-1:199x, Information Technology — CDIF Transfer Format — Parte 2: Syntax:** define uma sintaxe CDIF específica.
- **ISO/IEC 15475-1:199x, Information Technology — CDIF Transfer Format — Parte 3: Encoding:** define uma codificação CDIF específica.

A Figura 3.3 representa a família de padrões, incluindo o meta-metamodelo CDIF e o conjunto de regras associadas de Modelagem e Extensibilidade. Este conjunto de regras

define o *framework* do Metamodelo Integrado CDIF e do Formato de Transferência CDIF.

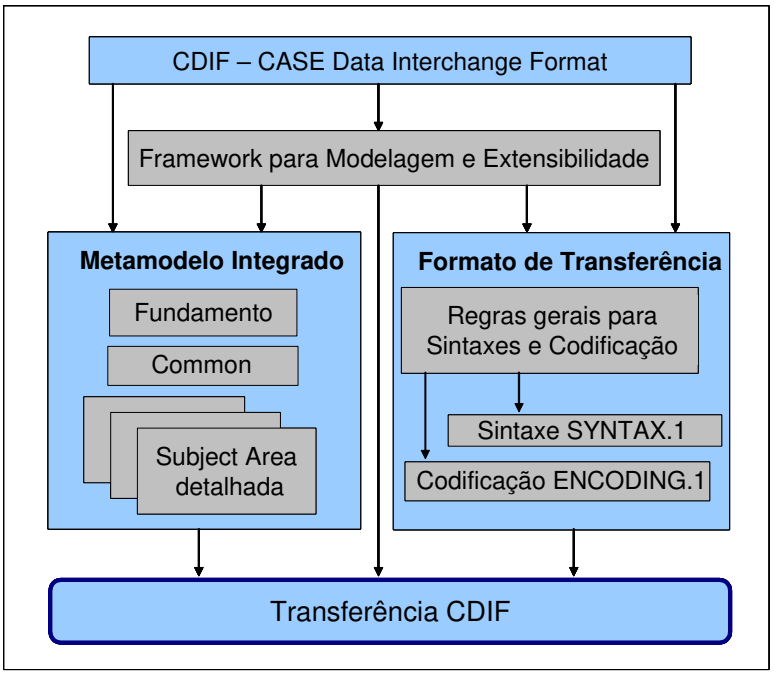

Figura 3.3 – A família de padrões CDIF

Arquitetura CDIF

CDIF possui uma arquitetura formal usada para definir o relacionamento entre as informações representadas em uma transferência, a semântica desta informação expressa pelo Metamodelo Integrado CDIF e qualquer extensão que esta possua [ERNST97].

O CDIF é baseado na arquitetura de modelos em quatro camadas. Quatro camadas são utilizadas, sendo a segunda uma abstração da primeira, a terceira da segunda e assim por diante. Cada abstração, ou meta-camada, como é chamada, define as regras para o nível do qual é abstraída. Esta arquitetura utiliza modelos para descrever informações contidas em uma camada com menor nível de abstração.

Transferência CDIF

Os conceitos básicos da arquitetura são expressos na separação das definições: qual informação será transferida e como. O conteúdo da informação de uma transferência é definido pelo Metamodelo Integrado CDIF e, a definição de como a informação é colocada em um formato padronizado, é definida pelo Formato de Transferência. O meta-

metamodelo CDIF estabelece as regras e os blocos de construções para as definições do Metamodelo Integrado CDIF e do Formato de Transferência, unindo-os.

Metamodelo Integrado CDIF

O Metamodelo Integrado CDIF é um modelo integrado único de Entidade-Relacionamento-Atributo, sendo uma descrição do conjunto de construções e notações usadas na definição de modelos de informação em desenvolvimento de sistemas [LEMESLE98]. Esse modelo é particionado pelas visões de construções do metamodelo base em seções gerenciáveis, chamadas *Subject Areas*.

Subject Areas são conjunto de meta-objetos coletáveis (meta-entidades, meta-atributos e meta-relationamentos) que cobrem técnicas específicas encontradas em ferramentas CASE, tais como, Data Flow Modeling e OOAD. Geralmente, esses metas-objetos aparecem em várias S*ubject Areas*. As meta-entidades, os meta-atributos e os meta-relacionamentos incluídos em cada *Subject Area* suportam os conceitos comuns requeridos pelos métodos usando a técnica coberta. Quando informações adicionais são requeridas entre ferramentas similares ou diferentes, cria-se a necessidade de um mecanismo de extensibilidade para que os construtores de ferramentas estendam a definição da informação. Não há um mecanismo de extensão para *Subject Areas*, acontecendo a criação de uma nova, a qual referencia todas as entidades definidas na primeira [LEMESLE98].

Formato de Transferência CDIF

O propósito de uma transferência CDIF é suportar a transferência de: referências para as *Subject Areas*; extensões do Metamodelo Integrado CDIF; instâncias de meta-entidades, meta-relacionamentos e meta-atributos, como definido no metamodelo de transferência.

Toda transferência é usada seguindo uma sintaxe e codificação. A estrutura geral de todas as transferências CDIF consiste em: uma seção de formato único para toda transferência chamada envelope e outra seção chamada conteúdo.

O envelope de transferência consiste em uma assinatura CDIF, um identificador de sintaxe e um identificador de codificação. A seção conteúdo consiste nas seções de cabeçalho, metamodelo e modelo.

O cabeçalho contém informações sobre a transferência, por exemplo, a identificação da ferramenta exportadora do modelo e a informação de gerenciamento de informação, incluindo responsável, dia e hora da exportação.

A seção de Metamodelo define o metamodelo utilizado na transferência, contendo referências as *Subject Areas* padrões e também as extensões do Meta-modelo Integrado. As extensões são utilizadas caso o exportador necessite prover definições próprias e estas serem utilizadas na interpretação do modelo de dados importado pela ferramenta.

A seção Modelo contém instâncias de meta-entidades e meta-relacionamentos, associados com os seus meta-atributos.

3.4.3 Resource Description Framework (RDF)

Membros da *World Wide Web Consortium* (W3C) e representantes de empresas trabalharam juntos para suprir a necessidade de uma arquitetura robusta e flexível que promovesse a interoperabilidade entre diferentes padrões de metadados. Em outubro de 1997, como o resultado desse trabalho, surgiu o primeiro *draft* do *framework Resource Description Framework* (RDF).

O RDF é um *framework* para a representação de informações na Web [W3C04c]. Esse *framework* foi desenvolvido de maneira que vocabulários possam ser estendidos por camadas. A linguagem RDF e RDF Schema (RDFS) estão entre as primeiras camadas desse *framework*, sendo outras, como DAML, situadas em camadas superiores. A Figura 3.4 demonstra a arquitetura de padrões de semântica XML da W3C

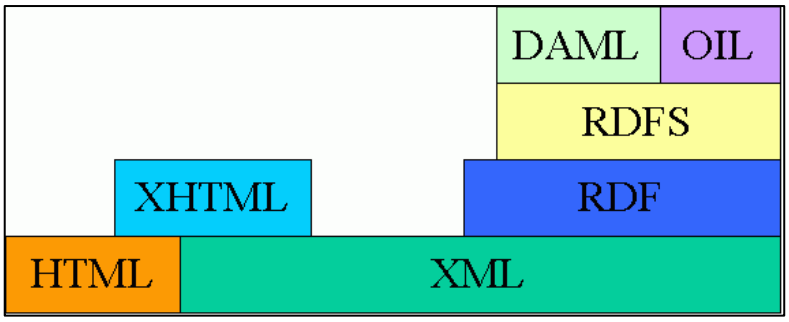

Figura 3.4 – Arquitetura de padrões W3C XML Semântica

O RDF também pode ser considerado uma linguagem para representar informações sobre recursos Web. Esses recursos, na verdade, podem ser generalizados, passando o RDF a ser usado para representar informações sobre coisas (recursos) que podem ser identificadas na Web através de identificadores de recursos uniformes (URI - *Uniform Resource Identifiers*) [W3C04d]. Esses recursos, que necessariamente não precisam estar acessíveis, são descritos pelo RDF através de simples pares propriedades/valores.

O *framework* fornecido pela recomendação RDF permite expressar informações, possibilitando a troca dessas entre aplicações sem nenhuma perda semântica. Os dados disponíveis na Internet, através da utilização de RDF, passaram a possuir semânticas ricas, possibilitando uma melhor utilização das mesmas pelas aplicações, aumentando os seus valores.

Somente o uso de XML não fornece suporte para interoperabilidade de metadados. Interoperabilidade de metadados é possibilitada através do desenvolvimento de mecanismos que suportem convenções comuns de semântica, sintaxe e estrutura [MILLER98]. RDF pode suprir esta característica não atendida pela XML, levando a semântica dos documentos ao nível de entendimento de máquinas.

A tentativa de criação de documentos XML capazes de serem compreendidos por máquinas, não se tornou realidade. O problema está no fato da linguagem XML abordar somente a sintaxe e a estrutura do documento, precisando que RDF complete-a, acrescentando semântica de maneira padronizada [W3C99b].

A especificação RDF define a linguagem XML como a sintaxe padrão para a descrição de recursos Web, objetivando classificar e acessar informações de forma mais eficiente através do uso de metadados.

A especificação RDF

RDF é uma recomendação W3C desde 1999. A recomendação consiste em dois documentos: um documento para a especificação do modelo e sintaxe (RDFMS - *RDF Model and Syntax*) [W3C99b] e outro para a especificação de esquemas (RDFS - RDF Schema) [W3C99c]. A divisão da recomendação em apenas dois documentos tornou-a complexa e confusa, dificultando a sua leitura e o seu entendimento. Atualmente, a recomendação é composta por um conjunto de seis documentos, descritos abaixo:

- *RDF Primer*: visa introduzir os conceitos básicos do RDF e a sua sintaxe XML [W3C04d], servindo como uma introdução e um tutorial de utilização do RDF e RDFS.
- *RDF Concepts and Abstract Sysntax*: define uma sintaxe abstrata, na qual RDF é baseada. Essa sintaxe abstrata serve para ligar a sintaxe RDF concreta com a semântica formal [W3C04c]. Especifica conceitos fundamentais e modelos de informação do RDF.
- *RDF Syntax*: define uma sintaxe para o RDF, utilizando a linguagem XML, chamada RDF/XML [W3C04e], i.e., define como escrever RDF na sintaxe XML.
- *RDF Semantics*: especifica a semântica e os sistemas completos de regras de inferência para o RDF e RDFS [W3C04f].
- *RDF Vocabulary Description Language*: define como utilizar o RDF para descrever vocabulários para um domínio específico. Esta especificação define além deste vocabulário, outros inicialmente estabelecidos no documento RDFMS [W3C04f].
- *RDF Test Cases*: descreve um conjunto de casos de teste que preenchem os aspectos das outras especificações e que podem ser usados para o teste automático de implementações [W3C04g].

Modelo de dados RDF

RDF possui um modelo de dados baseado em grafos simples, facilmente processado e manipulado pelas aplicações. Este modelo de dados RDF é independente de qualquer sintaxe de serialização específica. O RDF especifica uma sintaxe de serialização específica (*RDF Syntax*) utilizando XML. A linguagem XML é utilizada para codificar o modelo de dados para a troca de informações entre aplicações. O fato de RDF utilizar valores representados de acordo com tipos de dados de esquemas XML ajuda a troca de informações entre RDF e outras aplicações XML.

RDF: linguagem declarativa

RDF possui uma semântica formal, provendo uma base confiável para o processo de tirar conclusões sobre o significado de uma expressão RDF. Em particular, ele suporta rigorosamente notações definidas de implicação, fornecendo uma base para definir regras de inferência confiáveis em dados RDF.

Utilizando RDF como uma linguagem declarativa, obtém-se um meio padrão para representar metadados por meio de declarações (*statement)*, sendo essas tríades compostas de recurso, propriedade e valor.

Recurso é o objeto a ser descrito, podendo ser tudo que tenha um endereço Web (URI), i.e., uma página Web, um gráfico, um arquivo de áudio ou um de vídeo. Uma propriedade é uma característica definida de um recurso, usada para descrevê-lo, como, por exemplo, o autor de um livro ou a data de publicação da obra. E valor é o valor da propriedade do recurso específico.

As propriedades específicas de um domínio são especificadas pelas comunidades individuais de interesse. Uma comunidade definirá um conjunto de propriedades, pertencentes a um *namespace*[7], podendo definir vocabulários de metadados para descrever os recursos Web que sejam de seu interesse.

[7] RDF usa a facilidade de *namespace* existente no XML para associar cada propriedade usada com o esquema que a define.

Um exemplo de definição de propriedades por uma comunidade específica é o *Dublin Core* (DC). O conjunto DC [DC04] definiu quinze propriedades: Title, Creator, Subject and Keywords, Description, Publisher, Contributor, Date, Resource type, Format, Resource identifier, Source, Language, Relation, Coverage, Rights management. Esses nomes de propriedade vêm sendo utilizados como nomes de elementos *META HTML*, fornecendo metadados para mecanismos de busca na Web.

Abaixo, um exemplo da instrução de metadados RDF.

Tabela 3.1 – Exemplo da instrução de metadados RDF

```
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/o2/22-rdf-syntax-ns#"
      xmlns:dc="http://purl.org/meta data/dublin_core#">
      <rdf:Description about:"http://exemplo.com/Pagina">
          <dc:Creator>Luciano</dc:Creator>
      </rdf:Description>
</rdf:RDF>
```

Na Tabela 3.1, o atributo *about* do elemento *Description* identifica o recurso "http://examplo.com/Pagina". O elemento *Creator* é uma propriedade pertencente ao recurso, cujo valor é "Luciano".

3.4.4 O Open Information Model e o formato XML Encoding

O OIM é um metamodelo extensível baseado em UML, o qual padroniza sub-metamodelos especializados, cobrindo as necessidades de áreas importantes, tais como, análise e desenvolvimento, descrição de objetos e componentes e gerenciamento de conhecimento. A iniciativa facilita o compartilhamento e reuso no desenvolvimento de aplicações e no domínio de *data warehouse* através de metadados intercambiáveis [MSDN99].

A iniciativa OIM foi originalmente desenvolvida pela Microsoft® em 1996 como parte de sua tecnologia de gerenciamento de metadados e subseqüentemente transferida para a *Meta Data Coallition (MDC)* [COSTA01]. Em setembro de 2000, com a fusão da MDC e da OMG, a iniciativa foi descontinuada. Essa fusão representou a união de esforços na área de metamodelagem, fazendo emergir padrões dentro da OMG baseados nas experiências conjuntas. Um exemplo é o padrão OMG *Common Warehouse*

Metamodel (CWM) [OMG03b], o qual foi amplamente derivado dos sub-metamodelos de data warehousing do OIM [COSTA01].

O sistema de repositório, *Microsoft Repository™ 1.0*, é a implementação mais expressiva do *Open Information Model (OIM)*. Atualmente, o repositório faz parte de um conjunto de serviços destinado ao gerenciamento de metadados chamado *Meta Data Services*. O repositório é composto de dois componentes principais [BERNSTEIN99]: uma máquina de repositório que implementa um conjunto de interfaces orientadas a objetos no topo de um sistema de banco de dados *Microsoft SQL Server* e um conjunto de modelos de informação OIM.

Ainda que a UML seja o meta-metamodelo para descrever, interpretar e estender os metamodelos OIM, o repositório da *Microsoft* utiliza um meta-metamodelo proprietário para este propósito, conhecido como *Repository Type Information Model (RTIM)* [COSTA01]. O RTIM é um modelo de objetos que define como modelos de informação são armazenados em um repositório [MSDN00].

Completando a solução de gerenciamento de metadados, existe um formato para a troca de modelos baseados no OIM chamado *XML Interchange Format* (XIF). O formato utiliza o RTIM como meta-metamodelo, as estruturas de modelos de dados de acordo com os metamodelos OIM e a sintaxe XML para a troca de modelos através de um mapeamento bidirecional entre documentos XML e modelos baseados em OIM [MSDN01].

O formato XIF foi sucedido pelo formato adotado pela Microsoft® chamado *OIM XML Encoding (XML Encoding),* pertencente à *MDC*. O *XML Encoding* permite importar, exportar e publicar metadados no formato XML, possibilitando a troca de metadados entre repositórios, um repositório e uma aplicação, ou aplicações que consigam interpretar o mesmo formato XML.

Entre as diferenças entre os dois formatos pode ser citada a otimização do *XML Encoding* para a versão mais recente do OIM que inclui suporte para o UML 1.3. Esta otimização não impede a geração de documentos XML válidos e bem formados para qualquer modelo de informação, mesmo que este não utilize a recente versão, ou seja, baseado em OIM. Caso não seja baseado em OIM, as regras do *XML Encoding*

continuarão determinando quais elementos do modelo de informação serão utilizados para a estrutura dos dados do repositório.

Outra diferença está em como os formatos expressam os metadados, sendo estes expressos no formato *entity-normal* dentro do XIF, enquanto o *XML Encoding* expressa no formato *attribute-normal*.

Os dois formatos são mecanismos de codificação alternativos não compatíveis, i.e., não é possível combinar XIF e *XML Encoding* em uma mesma troca de modelo. Esta incompatibilidade está no fato de XIF não ser compatível com o formato de modelagem MDC OIM. O formato *XML Encoding* trabalha para qualquer formato de modelo de informação que seja baseado em OIM.

Uma vantagem encontrada em *XML Encoding* é a capacidade de mapear versões prévias de modelos baseados em OIM para novas versões. Esse mapeamento ocorre automaticamente durante a operação de importação e exportação, realizando isto sem modificar o modelo de informação. Essa vantagem possibilita a troca de dados entre modelos de informação baseados em diferentes versões do OIM. A única exceção ocorre para novos elementos UML e OIM que não conseguem se mapear em versões OIM anteriores.

A geração de documentos XML para a transferência é realizada através da codificação de objetos de metadados OIM em XML através de um conjunto de regras. Nessas regras, os nomes dos elementos e dos atributos usados na representação são derivados do modelo de informação que se deseja transferir.

É apresentado um conjunto de DTDs XML acompanhando a especificação OIM. Este conjunto forma uma gramática para expressar as estruturas dos documentos XML que estão sendo trocados, visando facilitar o entendimento destes e ajudar o desenvolvimento da funcionalidade de importação e exportação. Apesar de apresentarem este conjunto de DTDs XML, estes não são suficientemente expressivos para cobrir toda a semântica OIM.

3.5 Considerações Finais

Nesse capítulo foram apresentados conceitos de metadados, a arquitetura de metadados e os padrões que compuseram a RFP SMIF. Além disso, foram também apresentados os padrões RDF e OIM, fornecendo uma visão externa a OMG.

As três submissões relativas a RFP possuem o MOF como seu meta-metamodelo e trazem uma sintaxe e codificação completa para a exportar ou importar modelos e metamodelos compatíveis com MOF. Apesar de possuírem estas características, tinham como pontos fracos: necessidade de mapeamento de seu meta-metamodelo para o meta-metamodelo MOF, como no CDIF; não utilizavam a XML como sintaxe padrão, por exemplo, em UOL.

4 META OBJECT FACILITY E XML METADATA INTERCHANGE

Inúmeras ações implicam em manipulação de metadados em sistemas distribuídos, por exemplo, descrição de interfaces e do comportamento do componente e também a descrição de recursos, normalmente complexos, encontrados nesse tipo de ambiente.

Visando solucionar o problema de manipulação e gerenciamento de metadados em ambientes distribuídos, foi apresentado o padrão Meta-Object Facility (MOF) pelo Object Management Group (OMG).

4.1 Meta Object Facility (MOF)

Em junho de 1996, o OMG solicitou um Request for Proposal (RFP) para a definição da especificação MOF. Sendo a primeira especificação, versão 1.1, ratificada em 1997. Desde então, inúmeras revisões foram realizadas, resultando nas versões 1.3 em junho de 1999 e 1.4 em outubro de 2001. Atualmente, o consórcio OMG destina esforços na elaboração da especificação MOF versão 2.0, onde o enfoque principal é prover um *framework* de metadados, independente de plataforma, que amplie a unificação entre os padrões do consórcio OMG.

O padrão MOF parte da premissa básica de que haverá vários tipos de modelos, portanto, existirá mais de um tipo de linguagem de modelagem [FRANKEL03]. Sendo assim, a especificação define uma linguagem abstrata e um *framework* capaz de especificar, construir e gerenciar metamodelos de diferentes domínios, independentes de tecnologias [OMG02a].

A especificação inclui os seguintes itens:

- A arquitetura de meta-modelagem MOF.
- A definição formal do Modelo MOF, i.e., a linguagem abstrata para especificar metamodelos MOF.
- Mapeamento-padrão de metamodelos MOF para interfaces IDL CORBA, produzindo interfaces capazes de manipular os metadados, armazenados como

objetos em um repositório, que estão em conformidade com os respectivos metamodelos.

- Além das interfaces IDL CORBA adaptadas para cada metamodelo, é especificado um conjunto de interfaces reflexivas que permitem a manipulação e a introspecção de metadados armazenados em um repositório sem conhecimento prévio de suas interfaces.

4.1.1 A arquitetura de Meta-modelagem MOF

A especificação MOF possui uma abordagem central focada na extensibilidade, permitindo que novos tipos de metamodelos sejam adicionados e suportados pelo *framework* conforme a necessidade. Para tal, a especificação MOF adotou a arquitetura clássica de meta-modelagem de quatro camadas, abordada na Seção 3.3, já utilizada por comunidades padrões como CASE Definition Interchange Facility (CDIF) e ISO [ERNST97].

4.1.2 Os níveis da arquitetura MOF

A arquitetura de meta-modelagem MOF é definida em 4 níveis de abstração de informação: M3 (meta-meta-metadados/meta-metamodelo), M2 (meta-metadados/metamodelos), M1 (metadados/ modelos) e M0 (dados /objetos).

O Nível M3 - Modelo MOF

No nível M3 da arquitetura MOF está presente um meta-metamodelo capaz de descrever todos os metamodelos de nível M2 chamado Modelo MOF. Conceitualmente, o Modelo MOF é o único modelo existente no nível M3 [FRANKEL03], isto é, só existe um meta-metamodelo definido na arquitetura.

O Modelo MOF é, na verdade, um subconjunto do metamodelo UML, utilizando grande parte dos construtores UML para expressar metamodelos. As construções UML não incluídas no Modelo MOF são [FRANKEL04]: *AssociationClasses* (associações que são objetos de primeira classe), *Qualifiers* e *N-ary associations* (associações entre mais de duas classes). Esse alinhamento entre os conceitos de metamodelagem MOF e os conceitos de modelagem UML permite o uso das notações gráficas UML para expressar metamodelos MOF.

O meta-metamodelo MOF é imutável [SILVA03], sendo esta característica vinculada à necessidade de se limitar o número de níveis da arquitetura, onde um nível é descrito pelo nível imediatamente acima. Na arquitetura MOF, este processo foi interrompido através da definição do nível M3 em termos de si mesmo, isto é, o nível M3 se auto-descreve.

Nesse nível, são definidas as construções de metamodelagem utilizadas na definição de metamodelos do nível M2, ou seja, um conjunto de construções capazes de definir metamodelos. Exemplos dessas construções são: MOF::Class, MOF::Attribute, MOF::Association, entre outros.

O Nível M2 - Meta-modelos

O nível M2 é composto por metamodelos definidos através do Modelo MOF. Exemplos de metamodelos padrões definidos pela OMG são: o metamodelo UML, o metamodelo CWM e o metamodelo CORBA Component Model (CCM). A camada M2 não é restrita a metamodelos-padrão, podendo-se definir e adicionar novos metamodelos, assim como, estender-se metamodelos existentes a fim de atender requisitos específicos de um contexto.

Os elementos pertencentes aos metamodelos são descritos pelos elementos do Modelo MOF do nível M3. Assim, as construções M2 dizem-se instâncias, na terminologia UML, das construções M3, isto é, os elementos descritos neste nível são instâncias dos elementos do nível superior M3, como ilustrado na Figura 4.1.

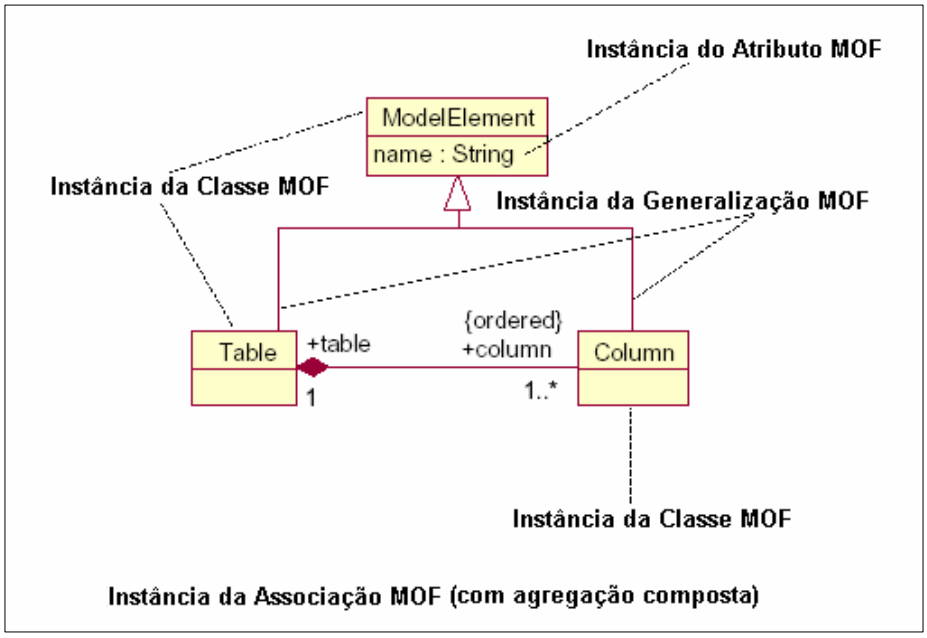

Figura 4.1 – Construções de metamodelo M2 como instância das construções M3

O Nível M1 - Modelo

No nível M1 estão os modelos representando domínios de dados/objetos específicos. Esse nível corresponde aos metadados/modelos das aplicações e sistemas do nível M0. Nessa camada está o conjunto de modelos que são instâncias dos vários metamodelos do nível M2 [MUTSCHLER00]. Cada modelo é definido de acordo com um metamodelo M2, onde o último fornece as construções necessárias para instanciar e interpretar os elementos do modelo M1. A Figura 4.2, ilustra instâncias M1 das construções Classes e Associações, de nível M2.

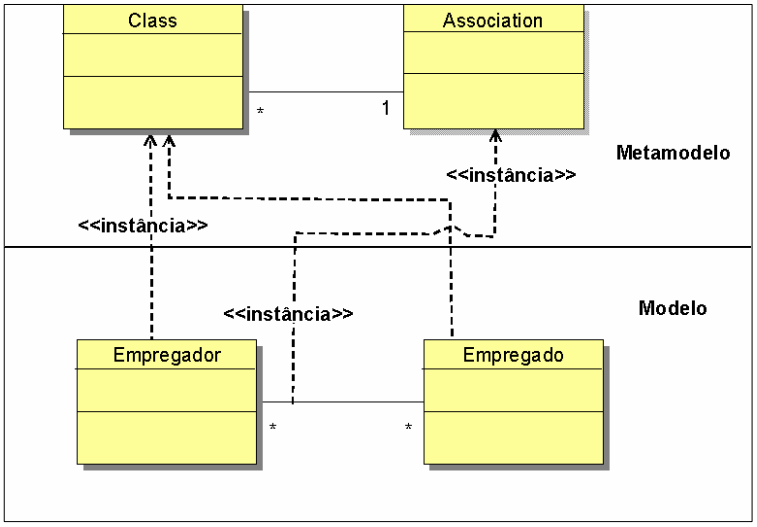

Figura 4.2 – Um modelo e o seu Metamodelo

O Nível M0 - Objetos e dados (informação)

Esse nível representa o mundo real, formado por instância dos elementos de um dado modelo/metadado de nível M1. É no nível M0 que residem os objetos que foram modelados no nível superior, por exemplo, objetos Java e C++.

4.1.3 O Modelo MOF (nível M3)

Conforme anteriormente descrito, a especificação MOF descreve um modelo pertencente ao nível M3 chamado simplesmente de "Modelo MOF". O Modelo MOF fornece um conjunto de construções de metamodelagem para construção de metamodelos [OMG02a].

Esse meta-metamodelo MOF pode ser estendido por herança ou composição para definir um novo metamodelo de informação mais rico que suporte construções adicionais. As construções pertencentes ao Modelo MOF são utilizadas para definir novos metamodelos. Desta maneira, o Modelo MOF é referido como meta-metamodelo, pois está sendo utilizado para definir metamodelos, tais como, UML e CWM.

A figura 4.3 demonstra o diagrama de classes do Modelo MOF de modo simplificado.

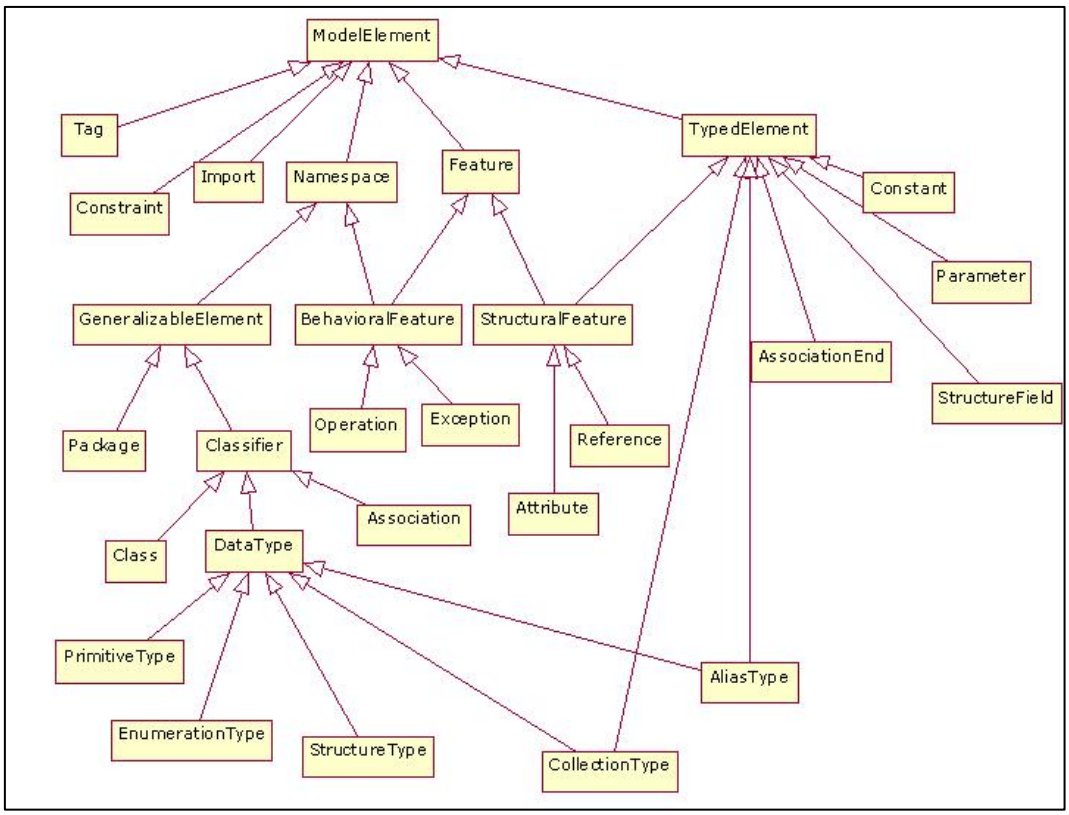

Figura 4.3 – Diagrama de classes do Modelo MOF

4.1.4 Principais construções de metamodelagem do Modelo MOF

Nessa seção serão introduzidas as principais construções de metamodelagem existentes em MOF. O conjunto de construtores fornecido pelo MOF é, essencialmente, um subconjunto das construções UML.

A metamodelagem consiste na definição de modelos de informação para metadados [OMG02a]. Em MOF, a atividade de metamodelagem utiliza a técnica de modelagem orientada a objetos. Existem inúmeras construções de metamodelagem, sendo Classes, Associações e Pacotes as principais fornecidas pela especificação MOF [OMG02b]. Outras como DataTypes e Constraints também são construções relevantes.

Alguns conceitos de modelagem de objetos aparecem em diferentes níveis de modelagem podendo causar conflito. 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 [OMG02a]. Para evitar este tipo de confusão, o desejável seria evitar o uso do prefixo "meta". Neste trabalho,

optou-se pelo uso explícito do nível em modo *top down* em relação ao Modelo MOF, nível M3. Os termos são identificados pelo nome da construção mais o nível ao qual pertence, por exemplo, classe-M2. A Figura 4.4 demonstra a construção Classe sendo utilizada em dois diferentes níveis, M2 e M3.

Classe

Classes modelam metaobjetos, ou seja, descrevem os tipos de um metamodelo. O termo metaobjeto é utilizado para referenciar um objeto abstrato ou específico de uma tecnologia que representa um metadado [OMG02a]. Exemplos são: uma classe do metamodelo UML e uma interface no metamodelo Corba IDL.

Classes definidas em nível M2 possuem instâncias em nível M1, ou seja, metaobjetos. Uma Classe MOF define o tipo de um metaobjeto ao invés de sua implementação. Uma classe pode conter as construções atributo, operação, referência, exceção, constante, tipo de dados e restrição. As classes-M1 possuem suas propriedades e comportamentos descritos por classes-M2 e, estas, por sua vez, têm suas propriedades e comportamento descrito por classes-M3.

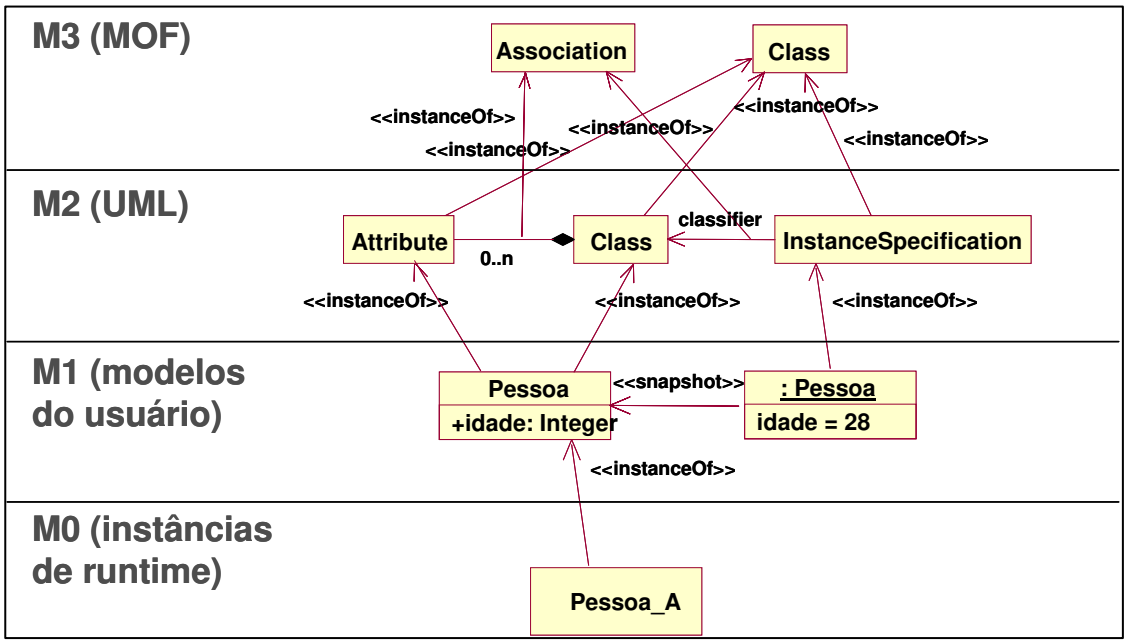

Figura 4.4 – Exemplo da arquitetura MOF utilizando o metamodelo UML

A expressão "generalização", assim como em UML, também é utilizada para descrever o relacionamento de herança em MOF. Sendo assim, Classes podem herdar de uma ou mais classes. Algumas restrições são impostas pela especificação para que os

metamodelos possam ser mapeados para um conjunto de tecnologias de implementação. Exemplos de restrições ao uso de generalização são: uma classe não pode se auto-generalizar, direta ou indiretamente, e uma classe que possui várias superclasses não pode conter ou herdar elementos que possuam o mesmo nome.

Classes podem possuir três tipos de características: atributos, operações e referência. Essas características podem ser definidas tanto no nível de metaobjetos quanto no de meta-metamodelo através do uso da definição "*instance level*" ou "*classifier level*". Por exemplo, uma operação definida como "*instance level*" em uma Classe-M2 pode somente ser invocada em Classe-M1.

Atributos são utilizados para a representação de metadados. Operações são fornecidas para suportar funções específicas do metamodelo no metadado.

Associação

Associações são os construtores primários do Modelo MOF para expressar os relacionamentos em um metamodelo [OMG02a]. Associações suportam ligações binárias entre instâncias de Classes MOF. Em nível M1, uma instância de Associação definida em nível M2 representa a ligação entre duas Classes.

Cada Associação possui apenas duas pontas, sendo estas chamadas de *Association Ends*. *Association Ends* descrevem as pontas da associação e definem as propriedades da mesma. As propriedades são: um nome único dentro da associação, o tipo da construção, nesse caso sempre uma classe, a qual está relacionada e a multiplicidade. Uma Classe do tipo *Association Ends* contém uma referência que permite a navegabilidade de uma ligação da Associação para uma instância de uma outra Classe.

Pacote

Pacote é o construtor utilizado em MOF para agrupar elementos em um metamodelo. Em nível M2, provê uma forma de modularizar ou particionar o espaço do metamodelo, podendo possuir a maioria dos elementos de metamodelagem, por exemplo, Classes, Associações, Tipos de Dados, Pacotes e etc. Em nível M1, pacotes representam o contêiner de nível mais alto ou externo para o metadado.

Pacotes podem ser reutilizados ou compostos através de quatro mecanismos definidos na especificação: generalização, aninhamento, importação e agrupamento. Uma instânica de um Pacote cluster comporta-se como se os Pacotes agrupados (clustered) estivessem aninhados. Pacotes aninhados sofrem algumas restrições impostas pela especificação como, por exemplo: pacotes aninhados não podem generalizar ou ser generalizados por outros pacotes.

Tipo de dado

Um tipo de dado representa dois tanto dados primitivos quanto complexos. Tipos de dados complexos podem ser definidos através da utilização de construtores como: *enumeration types*, *structure types*, *collection types* e *alias types*. Os tipos de dados primitivos são utilizados como parâmetros ou atributos dentro das classes. Na versão 1.4 da especificação MOF, os tipos de dados primitivos seguem os tipos primitivos de CORBA. A versão 2.0 de MOF torna-os independentes de tecnologia.

Restrição

Restrição é utilizada para associar restrições semânticas a outros elementos em um metamodelo MOF. Qualquer linguagem pode ser utilizada para expressar restrições, no entanto, há vantagens no uso de uma linguagem formal como a OMG Object Constraint Language (OCL).

4.1.5 Interfaces Reflexivas MOF

Metaobjetos possuem a vantagem de permitir que aplicações usem objetos sem o conhecimento prévio de suas interfaces. No contexto MOF, um metaobjeto permite que aplicações descubram a natureza de qualquer objeto MOF de nível M1.

Interfaces reflexivas MOF permitem a uma aplicação [OMG02a]:

- Criar, atualizar, acessar, navegar e invocar operações em objetos de instância de nível M1;
- consultar e atualizar as ligações utilizando objetos de Associação de nível M1, e
- navegar em uma estrutura de Pacote de nível M1.

O módulo de interface reflexiva MOF possui quatro interfaces abstratas que são herdadas por todas as interfaces de nível M1 de um modelo gerado a partir de um metamodelo pelo mapeamento MOF-IDL. Todas as interfaces de metamodelos também herdam esse pacote, pois são geradas pelo mesmo mapeamento. A Tabela 4.1 demonstra todas as interfaces reflexivas existentes em MOF.

Tabela 4.1 – Interfaces reflexivas MOF

Interface	Descrição
RefBaseObject	É herdada pelas outras três interfaces reflexivas. Provê operações comuns para todos os objetos MOF testar a identidade, retornar o seu metaobjeto e o container ao qual ele pertence.
RefObject	Provê a descrição do metaobjeto de um objeto Class Proxy. Possui um conjunto de operações para acessar e atualizar o objeto em um modo independente de modelo.
RefAssociation	Provê a descrição do metaobjeto de um objeto Associação. Possui operação de consulta e atualização das ligações entre as associações
RefPackage	Provê operações comuns para os objetos de Pacote de nível M1. Provê operações para acessar a descrição do metaobjeto e a coleção de objetos e suas associações.

O padrão JMI adicionou novas interfaces reflexivas para a manipulação de metadados. As interfaces reflexivas adicionadas são apresentadas na Tabela 4.2.

Tabela 4.2 – Interfaces reflexivas adicionais JMI

Interface	Descrição
RefFeature	Provêem operações comuns para os objetos Instance e Class Proxy, fornecendo um conjunto de operações para acessar e atualizar as características destes objetos. Uma classe-M2 é mapeada para dois tipos de objeto: Class Proxy e Instance. Class Proxy instancia objetos do tipo Instance e os mantêm como uma coleção. Instance é o objeto instância da classe-M2.
RefClass	Provê um conjunto de operações para objetos Class Proxy.
RefStruct	Provê um conjunto de operações para objetos do tipo StructType. Esta interface e a RefEnum são utilizadas para tipos de dados que não podem ser mapeados para tipos primitivos em Java. Neste caso, *structs* e *enumerations*, respectivamente.
RefEnum	Provê operações comuns para operações EnumType.

4.1.6 Considerações finais sobre o MOF

Mapeamentos-padrão para tecnologias específicas são utilizados pelo *framework*, fornecendo interfaces que podem ser usadas para definição e manipulação de metamodelos MOF em repositórios de metadados interoperáveis e consistentes para diferentes tecnologias. Por exemplo, o mapeamento MOF para IDL CORBA pode ser aplicado ao meta-metamodelo MOF e ao metamodelo UML para produzir interfaces CORBA para manipular metamodelos MOF e modelos UML respectivamente.

O uso de MOF dependerá do ponto de vista que está sendo adotado. A partir do ponto de vista de um desenvolvedor de sistema, que estará analisando as camadas da arquitetura de metadados de modo *top-down*, o MOF é utilizado para a definição de modelos de informação para um domínio particular de interesse. A definição, então, é utilizada para guiar desenvolvimentos e implementações de software subseqüentes. Outro ponto de vista que pode ser assumido é o de um programador, o qual terá uma visão *bottom-up* da arquitetura, precisando apenas que os objetos consigam obter descrições do modelo de informação para suportar reflexão e interoperabilidade.

Algumas características distinguem a arquitetura MOF das anteriores, como CDIF:

- O Modelo MOF é orientado a objetos, com construções de meta-modelagem que são alinhadas com construções da modelagem de objetos da UML;
- O Modelo MOF se auto-descreve, isto é, ele é definido utilizando suas próprias construções de meta-modelagem.

Esta última é de grande relevância, pois mostra que o MOF é suficientemente expressivo para a metamodelagem. Isso permite que as interfaces e os comportamentos no Modelo MOF sejam também definidos aplicando os mesmos mapeamentos existentes na especificação MOF, criando uma uniformidade semântica entre objetos computacionais que representam modelos, metamodelos e meta-metamodelos.

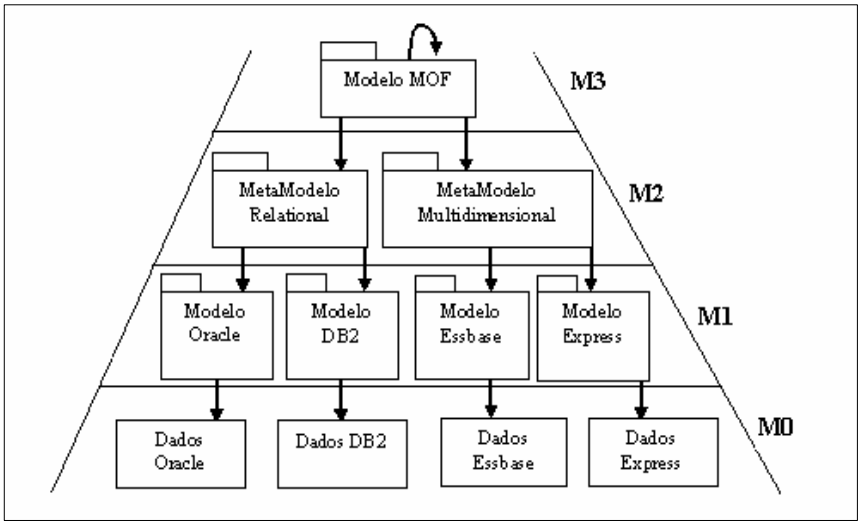

Figura 4.5 – Arquitetura MOF de 4 camadas

O padrão MOF trouxe consigo duas vantagens: primeiro, fornece a habilidade de estender metamodelos existentes ou criar novos metamodelos usando um processo padrão estabelecido; e segundo, fornece um mecanismo de interoperabilidade em nível semântico entre aplicações e ferramentas que utilizam diferentes metamodelos, através de seus mapeamentos e da utilização de diferentes níveis de abstração entre metadados/metamodelos/meta-metamodelos, conforme a Figura 4.5.

4.2 XML Metadata Interchange (XMI)

XMI é um formato de intercâmbio de metadados baseados em MOF [OMG02b]. O padrão foi resposta à solicitação da OMG para uma proposta de um formato serializado de troca de modelos (SMIF RFP).

Inicialmente, o propósito principal de XMI era permitir o intercâmbio fácil de modelos entre ferramentas de modelagem baseada em UML e repositórios de metadados. Hoje em dia, tanto MOF quanto XMI, servem para modelos de informação para qualquer domínio (CWM, Web Services, etc).

A interoperabilidade a nível sintático foi suprida através do uso da linguagem XML. Baseando-se o formato de intercâmbio de metadados da OMG nessa linguagem, vantagens são obtidas através dos inúmeros benefícios da XML, tais como: ser aberta e independente de plataforma e fabricante; suporte a padrões de conjunto de caracteres internacional ISO Unicode.

Em nível semântico, o padrão MOF mostrou-se uma solução viável e adequada. Ferramentas compatíveis com o MOF não necessitam de transformações específicas para cada tipo de metamodelo existente em seu ambiente de trabalho. Sendo esses compatíveis com MOF, é possível visualizar todos os metadados descritos pelas diferentes ferramentas.

Padrões de mapeamento fornecidos pelas especificações MOF e JMI [JCP02], conforme a Figura 4.6, expõem instâncias de metamodelos e modelos MOF através de IDLs CORBA e interfaces Java, respectivamente. O propósito principal destes mapeamentos é definir um *framework* genérico capaz de gerenciar, em termos de repositório, os modelos (metadados) descritos pelos metamodelos [FIGUEIREDO03].

Os mapeamentos-padrão descritos acima são dependentes de tecnologia de middleware, dificultando o intercâmbio de metadados entre repositórios baseados em diferentes tecnologias. O fato de o padrão XMI ser um formato de intercâmbio de metadados independente de tecnologia de middleware amplia o seu horizonte de utilização.

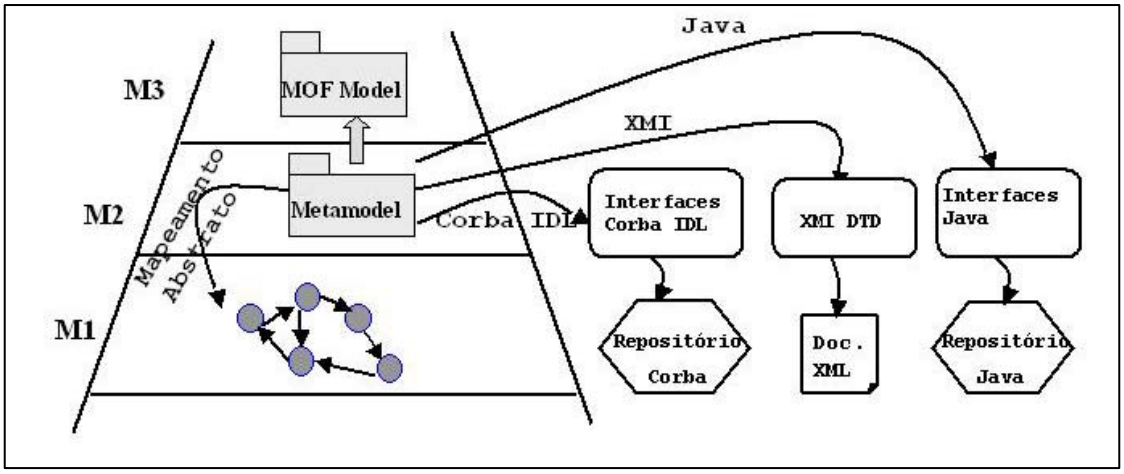

Figura 4.6 – Mapeamentos Padrões de um Metamodelo MOF

Em junho de 1998, um comitê técnico esboçou a proposta de submissão do padrão XMI, sendo revisada e, logo após, disponibilizada em novembro desse mesmo ano. O padrão, adotado em março de 1999 pela OMG, foi identificado como fundamental para o intercâmbio de modelos de informação de forma aberta. Atualmente, existem duas especificações válidas para o padrão XMI:

- XML Metadata Interchange (XMI), versão 1.2: é detalhado nesta especificação o conjunto de regras de mapeamento de metamodelos para DTDs e de modelos para documentos XMI.
- XML Metadata Interchange (XMI), versão 2.0: é detalhado nesta especificação o conjunto de regras de mapeamento de metamodelos para XML Schemas e documentos XMI.

4.2.1 Fundamentos

A especificação XMI, na visão de usuários, define um padrão de intercâmbio aberto cuja intenção inicial é oferecer a capacidade de trocar metadados entre ferramentas de modelagem baseadas em UML, repositórios de metadados baseados em MOF e aplicações em um ambiente heterogêneo distribuído. Qualquer ferramenta, aplicação ou repositório capaz de realizar a transferência de um arquivo texto ASCII, pode realizar a troca de documentos XMI. Em um contexto amplo, XMI pode ser visto como um formato de intercâmbio de metadados baseados em MOF independente de tecnologia de *middleware* [OMG02b].

XMI integra três padrões-chave desenvolvidos pela W3C e pela OMG [OMG02b]:

1. O padrão *eXtensible Markup Language* – XML - da W3C;
2. O padrão de modelagem *Unified Modeling Language* – UML – da OMG que define uma linguagem de modelagem orientada a objeto;
3. E o padrão de metamodelagem e de repositório de metadados *Meta Object Facility* – MOF.

O padrão XMI está relacionado com o padrão UML por dois motivos. Primeiro, o alinhamento entre os conceitos de metamodelagem MOF e os conceitos de modelagem UML permite o uso das notações gráficas UML para expressar metamodelos MOF. O segundo motivo, o metamodelo UML está em conformidade com o modelo MOF, i.e., o metamodelo UML reside no nível M2 de MOF. Sendo assim, o XMI pode ser diretamente utilizado para o intercâmbio de modelos UML.

O XMI utiliza XML como linguagem padrão para formatação dos dados a serem transferidos, realizando um mapeamento direto entre metamodelos MOF e a linguagem XML. Modelos e metamodelos baseados em MOF são, respectivamente, mapeados em documentos XML e XML DTDs ou Schemas. As regras desses mapeamentos são descritas em dois componentes principais detalhados na especificação [OMG02b]:

1. Regras de produção para produzir XML Document Type Definitions (DTDs) para metadados codificados em XMI. XMI DTDs servem como uma especificação de sintaxe para documentos XMI, e permitem que ferramentas XML genéricas possam ser usadas para compor e validar documentos XMI.
2. Regras de produção para codificação de metadados em documentos XMI. As regras de produção também podem ser aplicadas de modo reverso para a decodificação de documentos XMI para reconstrução do metadado.

Estas regras de produções produzem documento XMI e DTD XMI que auxiliam o intercâmbio tanto de modelos quanto de metamodelos expressos utilizando MOF. XMI auxilia também o intercâmbio de partes de um metamodelo ou modelo, podendo ser utilizado para a codificação de fragmentos de metadados. Atualmente, o XMI não impõe a utilização da multiplicidade especificada no metamodelos na criação de DTD [OMG02b]. Desta forma, todos os DTDs criados suportam o intercâmbio de fragmentos de modelos.

Extensões de metadados específicos às ferramentas podem ser adicionadas aos documentos XMI a serem exportados. Elementos de extensão provêem uma forma de inclusão de novas informações ao metamodelo. Um documento XMI pode conter zero ou mais elementos de extensão, devendo estes ser declarados nos DTDs XMI.

Além das regras de produção, a especificação traz também princípios de projeto de DTDs e documentos XMI, discutindo o uso, a geração e as partes padrões. Além disso, especifica o DTD para o metamodelo UML e para o Modelo MOF. A Figura 4.7 demonstra a obtenção de artefatos XMI correspondente a metamodelos e modelos.

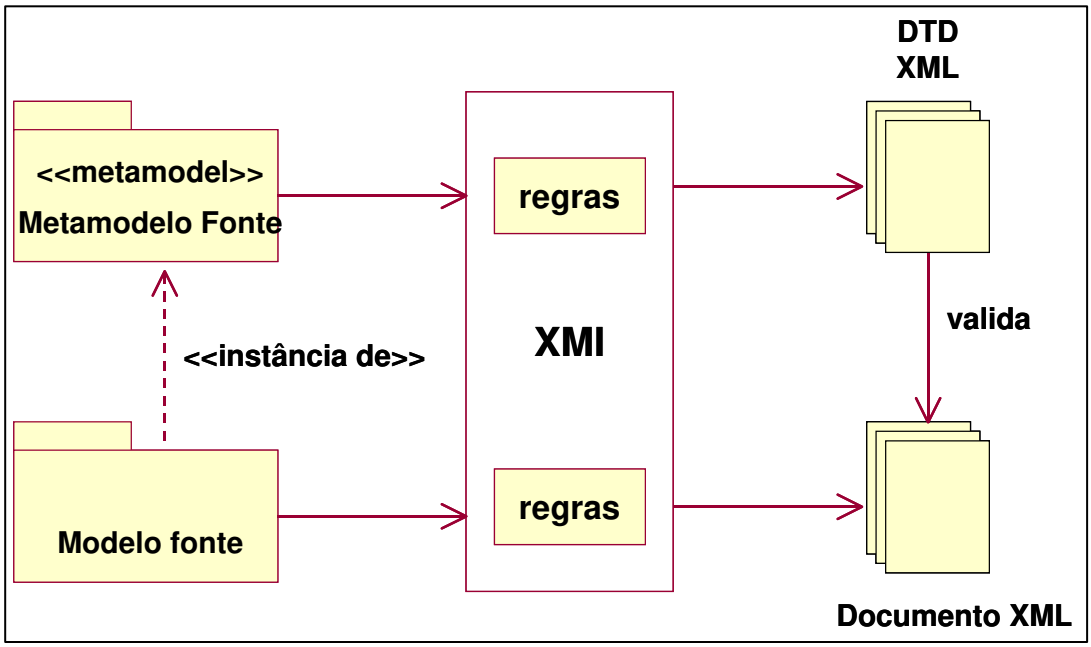

Figura 4.7 – Obtendo artefatos XML de modelos MOF

XMI provê uma rota possível para o intercâmbio de metadados entre repositórios cujos metamodelos não sejam baseados em MOF [OMG02b]. Este intercâmbio torna-se possível através da utilização de XMI ou de MOF. Utilizando XMI, é necessário o mapeamento "ad hoc" entre um documento e o metamodelo nativo do repositório. Entretanto, pode ser realizado um mapeamento em nível de meta-metamodelo, como, por exemplo, um mapeamento entre o meta-metamodelo CDIF e o Modelo MOF.

A especificação, por ser baseada em MOF, pode ser aplicada a metadados e metamodelos de vários domínios. O fato de MOF possuir em sua arquitetura um único meta-metamodelo, o Modelo MOF, permite que informação de metamodelo seja representada uniformemente, provendo uma associação entre seus respectivos modelos. Podendo assim, XMI ser igualmente aplicado a metamodelos que cobrem diferentes domínios como: gerenciamento de data warehouse, objetos distribuídos e gerenciamento de objetos de negócios [OMG02b].

O XMI pode ser utilizado em qualquer nível da arquitetura MOF de metamodelagem, possibilitando tanto a troca de modelos quanto de metamodelos. O intercâmbio de metamodelos pode ser realizado através da aplicação das regras de produção de DTD e documentos XMI ao meta-metamodelo MOF e ao metamodelo, respectivamente. Com isto, torna-se possível verificar e validar metamodelos intercambiados entre ferramentas e repositórios em relação ao Modelo MOF.

Por ser baseado em XML, o XMI é um formato de transferência mais flexível que as interfaces (IDL) CORBA, não precisando de conectividade entre ORBs para realizar uma transferência, podendo ser visto como um formato independente de middleware. Ferramentas ou repositórios precisam somente ter a capacidade de codificar e decodificar documentos XMI para que possa realizar o intercâmbio de metadados entre si [OMG02b].

4.2.2 O arquivo XMI

A produção de arquivos ou documentos XMI pode ser realizada através de dois métodos distintos de especificação dos elementos de modelagem utilizados: por *Object Containment* ou *Package Extent*.

No primeiro método, as regras de produção são aplicadas a partir do objeto raiz de um metamodelo por toda a hierarquia de composição, navegando através dos links de composição. Esse método parte do princípio de que a maioria dos metamodelos é caracterizada por uma hierarquia de composição e que alguns tipos de elementos de modelagem podem ser compostos por outros elementos de modelagem. Por exemplo, o Pacote mais externo de um metamodelo contém todos os outros elementos: Classes, Associações, etc.

A representação de um conjunto de elementos de modelagem através de composição não é sempre possível ou útil. Através da utilização de interfaces reflexivas MOF, por exemplo, *RefPackage* que retorna o pacote mais externo, é possível definir a estrutura a qual reflete o metamodelo. A abordagem *Package Extent* utiliza essas interfaces para definir a estrutura de um metamodelo.

Modelos complexos criam situações onde o uso das duas abordagens apresenta conseqüências distintas. Por exemplo, atributos definidos como *classifier-level* são todos incluídos pelo método *Package Extent*, enquanto pelo *Object Containment* estes são ignorados.

Apesar de diferentes abordagens de produção de documentos XMI, todo documento XMI possui uma estrutura padrão definida pela especificação. Esta estrutura de documento é inicializada pelo elemento raiz XMI que consiste em uma declaração de *namespace* e atributos para controle de versão, data e hora e verificação.

O atributo de verificação indica se o arquivo XMI foi verificado previamente pela aplicação que o gerou. Uma vez realizada esta verificação, o arquivo XMI representando o modelo pode somente possuir erros de codificação ou de transmissão, não possuindo erro semântico em relação ao metamodelo.

O elemento raiz XMI é composto por seções opcionais padronizadas chamadas: *header*, *content*, *difference* e *extensions*. A Tabela 4.3 ilustra um exemplo de arquivo XMI contendo apenas as seções *header* e *content*.

Seção header

Esta seção é composta de elementos XML que identificam o modelo, metamodelo e meta-metamodelo para o metadado sendo transmitido, fornecendo também informações sobre a ferramenta geradora do documento.

Os elementos XML que compõe a seção são: *documentation*, *model*, *metamodel*, *metametamodel* e *import* [OMG02b]. O elemento *metametamodel* refere-se sempre a versão do Modelo MOF. Estes elementos podem se tornar simples W3C XLink em futuras versões do XMI.

Seção content

Todas as informações relativas ao modelo ou metamodelo corrente que está sendo transferido é armazenada nesta seção. Os elementos de modelagem que constituem o modelo são declarados nesta seção através da criação de elementos XML e atributos.

O nome dos elementos XML, representando Classes, Pacotes e Associações do metamodelo, é antecedido pelo prefixo de *namespace* seguido de ":".

Tabela 4.3 – Exemplo de um documento XMI

```
<XMI xmi.version="1.2" xmlns:UML="org.omg/standards/UML">
  <XMI.header>
      <XMI.metamodel name="UML" version="1.3" href="UML.xml"/>
      <XMI.model name="exemplo" version="1" href="exemplo.xml"/>
  </XMI.header>
  <XMI.content>
      <UML:Class name="Pessoa">
          <UML:Classifier.feature>
                  <UML:Attribute name="nome" visibility="private"/>
          </UML:Classifier.feature>
      </UML:Class>
  </XMI.content>
</XMI>
```

Seção difference

A atualização de modelos complexos em ambientes distribuídos poder-se-ia tornar árdua e consumir tempo e largura de banda na sua transmissão. A ineficiência em se transmitir o arquivo inteiro para refletir apenas pequenas mudanças pode ser resolvida pela seção *difference* existente em XMI.

A seção *difference* é composta pelas seções opcionais *add*, *delete* e *replace*, podendo esta ser declarada dentro da própria seção *content* ou em uma seção própria, chamada *XMI.difference*.

Informações presentes na seção *add* são acrescentadas ao metadado base. Similarmente, informações posicionadas na seção *delete* serão apagadas e na seção *replace* substituídas no arquivo XMI alvo.

Seção extensions

Elementos XML que contêm informações adicionais ao metamodelo ou modelo são adicionados à seção *extensions*. Por exemplo, informações sobre a apresentação do modelo em determinada ferramenta. Atributo de identificação, *xmi.extender*, identifica a ferramenta que adicionou a extensão ao documento XMI. Extensões adicionadas pelas ferramentas podem ser ignoradas por outras antes do conteúdo da seção *extensions* ser processada [OMG02b].

4.2.3 Problemas encontrados no padrão XMI

A quantidade de informações adicionais que um arquivo XMI pode suportar é limitada pela sintaxe padrão. Várias informações pertinentes às ferramentas são convertidas em *Tagged Values*[8] e incorporadas aos arquivos XMI como informações adicionais ou de extensão.

A capacidade de importar e exportar arquivos XMI é um meio importante de publicação de informações para serem consumidas por outras ferramentas. No entanto, informações adicionais, tratadas como elementos de extensão, são freqüentemente ignoradas pelas ferramentas de diferentes fabricantes. Uma ferramenta pode importar um novo arquivo, ignorando as extensões, alterá-lo, adicionar novas extensões e novamente exportá-lo. Apesar das ferramentas poderem desconsiderar as extensões dentro de um arquivo XMI importado, de acordo com a especificação, estas devem novamente adicioná-los ao arquivo quando forem reexportá-lo.

O fato dos arquivos DTD XMI serem baseados diretamente no metamodelo, faz com que eles consigam expressar somente o que está contido no metamodelo, não permitindo assim que informações adicionais sejam representadas. Um exemplo desta limitação ocorre no mapeamento do metamodelo UML em um DTD XMI. O metamodelo UML contem apenas informações sobre modelos e não sobre os diagramas, os quais possuem informações de apresentação. Modelos UML mapeados em documentos XMI necessitam assim mapear informações de apresentação dentro da seção *extensions*.

[8] Tagged Values é um mecanismo de extensão, ele estende as características dos blocos construtivos, permitindo a inserção de novas propriedades na especificação de um elemento.

Apesar desta não ser uma limitação do padrão XMI, a sua solução, através da utilização de elementos de extensão, nos remete novamente ao problema anterior.

A representação de informações que não fazem parte do metamodelo através de extensões XMI não acaba com o problema de transferência de informações de apresentação, por exemplo. O simples fato de atualmente, não existir uma maneira padronizada para codificar informações deste tipo dentro de um arquivo XMI, ocasiona perda de informações na transferência de modelos utilizando este padrão.

O problema neste caso, não está somente na perda da diagramação do modelo, mas no fato de que o posicionamento dos elementos do modelo também fornece informações importantes, como, por exemplo, duas classes próximas geralmente possuem um forte relacionamento em comparação com outras mais distantes.

Outro problema comum que não possui uma solução geral ocorre quando uma ferramenta cria um elemento UML juntamente com uma extensão XMI contendo mais informações sobre ele. Supondo que uma outra ferramenta importe este arquivo, ignore as extensões e altere o elemento UML. A extensão XMI adicionada pela primeira fica sendo inconsistente com o estado atual do elemento. Uma maneira de facilitar a detecção de modificações aos elementos que possuem extensões desde a sua criação seria, por exemplo, a adição de um atributo de *timestamp* a cada elemento.

4.2.4 Usabilidade do padrão XMI

O uso do padrão XMI pode ser estendido para várias áreas de Tecnologia da Informação (TI) ajudando a solucionar alguns dos problemas encontrados atualmente. Um exemplo de utilização é na área de Serviços Web (*Web Services*). Serviços Web são aplicações de software, identificadas por um URI, possuindo interfaces definidas, descritas e descobertas por artefatos XML, que suportam interações diretas com outras aplicações de software usando mensagens baseadas em XML via protocolos de Internet [W3C02].

O processo de produzir Serviços Web deve ser o máximo possível independente das tecnologias (SOAP, UDDI, WSDL, etc) utilizadas. Por exemplo, a linguagem UML poderia ser utilizada para modelar os Serviços Web, onde operações WSDL seriam

mapeadas em operações UML. Uma vez modelado o serviço, este poderia ser, através de XMI, mapeado em DTDs, Schemas e documentos XMI. Desta forma, os serviços em formato XMI poderiam ser intercambiados entre diversas ferramentas específicas para cada tipo de mapeamento.

Existem outras aplicações para o padrão XMI, por exemplo, na área de *data warehouse* juntamente com o padrão CWM. Várias ferramentas com diferentes metadados são necessárias em um ambiente de *data warehouse*. O CWM facilita o acesso a diferentes metadados que podem ser intercambiados através de XMI.

4.3 Model Driven Architecture (MDA)

Hoje, um ambiente de TI é composto de uma variedade de soluções personalizadas e de aplicações implementadas com diferentes tecnologias, dificultando a interoperabilidade entre estas. Além disto, somam-se os fatos de serem ambientes altamente distribuídos e em constante renovação tecnológica.

Dentro deste contexto, o consórcio OMG ratificou a proposta Model-Driven Architecture (MDA) [OMG03d]. MDA é uma tecnologia emergente, anunciada em fevereiro de 2002. 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.

MDA une o mundo de modelagem (UML), metadados (MOF e XML) e middleware (perfis UML para Java, EJB, IDL, EDOC, etc). MDA é uma abordagem para utilização de modelos no desenvolvimento de softwares, separando a especificação de funcionalidade da sua implementação em uma tecnologia específica, utilizando modelos formais e semiformais e transformações entre os modelos resultantes.

Duas categorias principais de modelos são utilizadas em MDA: modelos independentes de plataforma (*Platform Independent Model* - PIM) e modelos específicos de plataforma (*Platform Specific Model* - PSM). A transformação consiste no mapeamento de modelos PIM especificados em linguagens independentes de plataforma como MOF e UML, em modelos específicos de plataforma através de regras de mapeamento formais.

Um modelo MDA é parte de uma função, estrutura ou comportamento do sistema em uma linguagem com uma sintaxe bem definida e semântica formal. Modelos MDA podem ser modelos UML, DTDs XML, descrições de serviços Web (WSDL), etc. Plataforma, em MDA, é definida como um conjunto de tecnologias que provê um conjunto coerente de funcionalidades através de interfaces e padrões de uso especificados, onde subsistemas dependentes da plataforma podem utilizá-las, sem preocuparem-se com detalhes de como as funcionalidades fornecidas pela plataforma foram implementadas [OMG03d].

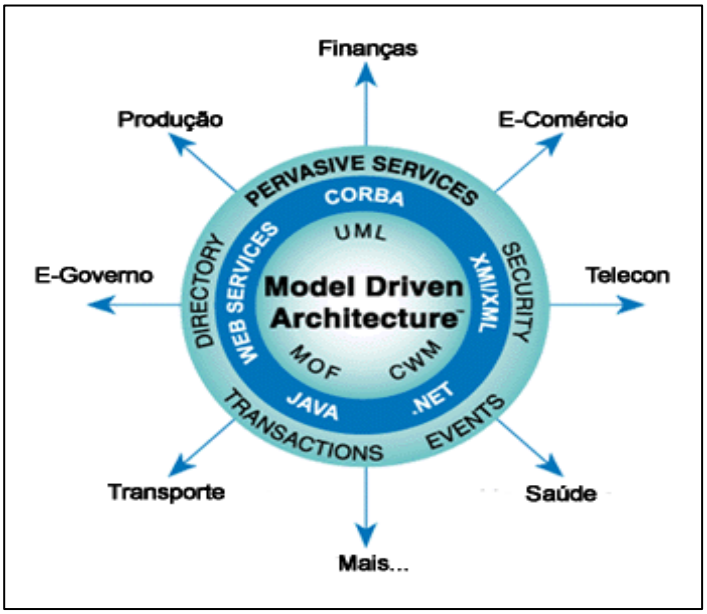

Figura 4.8 – Arquitetura Orientada a Modelos

A Figura 4.8 apresenta um exemplo de uma arquitetura MDA onde o núcleo é composto por padrões de modelagem e metamodelagem do consórcio OMG, o anel intermediário representando as tecnologias de implementação e plataforma de *middleware* e o externo, facilidades tais como serviços de diretório e segurança.

4.4 Considerações Finais

A premissa básica de que haverá vários tipos de modelos e, com isso, existirá mais de um tipo de linguagem de modelagem, cria a necessidade de uma linguagem de metamodelagem, capaz de descrever essas diferentes linguagens. Além disso, é necessário um *framework* capaz de especificar, construir e gerenciar metamodelos de

diferentes domínios. O padrão MOF supre essas necessidades, fornecendo uma linguagem abstrata capaz de definir linguagens para diferentes domínios e um *framework* para o gerenciamento de metadados baseados nele.

O padrão XMI permitiu o intercâmbio de modelos entre ferramentas e repositórios de metadados baseados em MOF. O fato de o padrão XMI ser baseado na linguagem XML tornou o formato de intercâmbio de metadados MOF independente de tecnologia de *middleware*.

5 MÓDULO GERADOR DTD XMI

Um grande limitador da interoperabilidade entre aplicações é a incompatibilidade dos metadados. Os diferentes padrões de metadados existentes dificultam a integração e a interoperabilidade de aplicações desenvolvidas com diferentes tecnologias de middleware.

O padrão MOF mostrou ser uma solução viável e adequada para o problema de manipulação e gerenciamento de metadados através de sua capacidade de descrever metamodelos de diferentes domínios e de manipular metadados através de mapeamentos-padrão.

Mapeamentos-padrão CORBA IDL e JMI definem um *framework* genérico capaz de gerenciar, em termos de repositório, os metadados descritos pelos metamodelos baseados em MOF. Apesar desses mapeamentos, era necessário um formato padrão capaz de permitir o intercâmbio de modelos entre ferramentas e repositórios e entre repositórios. O padrão XML Metadata Interchange (XMI) fornece este formato, sendo fundamental para o intercâmbio de modelos de informação de forma aberta, independente de plataforma de middleware.

A importância da realização de intercâmbio de metadados entre repositórios, ferramentas e aplicações utilizando o formato XMI conduziu à implementação de um módulo gerador de DTD XMI no Gerenciador de Repositório de Metadados (GRM). Este módulo permite a validação de arquivos XMI, verificando se o modelo representado está em conformidade com o seu respectivo metamodelo.

5.1 Gerenciador de Repositórios de Metadados (GRM)

O Gerenciador de Repositórios de Metadados (GRM) é um sistema gerenciador de metadados que utiliza o Modelo MOF como seu modelo de informação. O seu principal objetivo é: suportar a definição e o gerenciamento de diferentes metamodelos e metadados que estejam em conformidade com o padrão MOF.

O mercado oferece hoje ferramentas para o gerenciamento de metadados baseados em MOF, tais como, *Informatica SuperGlue* e *Unisys Universal Repository* (UREP). O

objetivo do desenvolvimento de um gerenciador próprio foi criar uma alternativa de baixo custo que ao mesmo tempo possuísse requisitos específicos como:

- Independência de sistema operacional e plataforma;
- Aderência aos padrões abertos propostos no contexto de sistema distribuído;

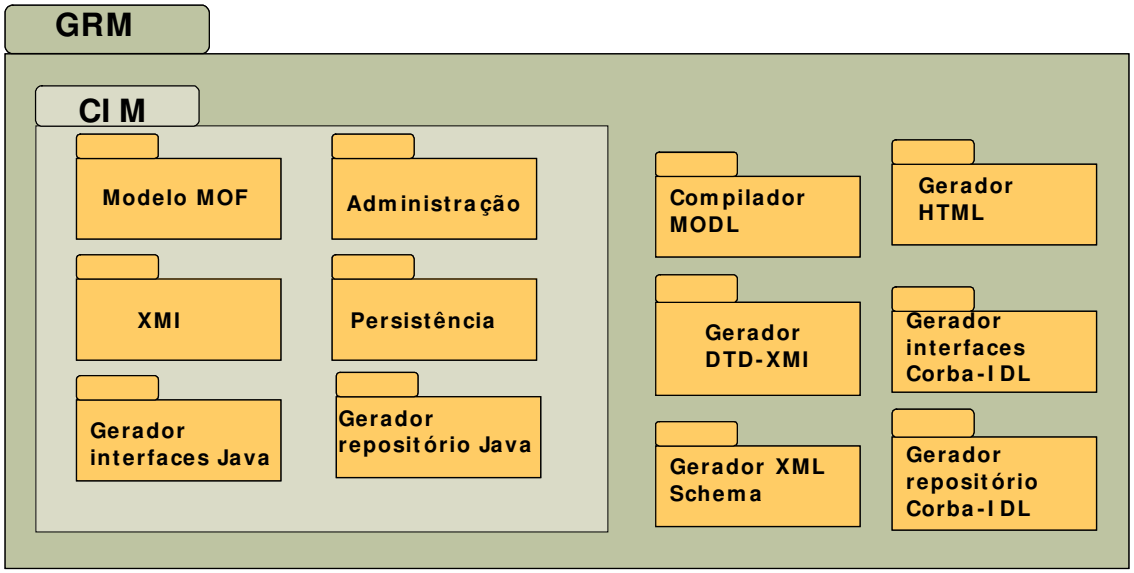

Figura 5.1 – Pacotes do Gerenciador de Repositórios de Metadados (GRM)

O repositório foi desenvolvido utilizando o *Open Source Complex Information Manager* (CIM) [UNISYS01] da *Unisys Corporation*. O CIM é um repositório que oferece capacidades de gerenciamento de metadados que aderem a especificação MOF em um ambiente de desenvolvimento Java puro.

5.2 Complex Information Manager (CIM)

O CIM foi todo desenvolvido em linguagem Java, tendo utilizado o conjunto de pacotes *Java Development Kit 1.2.2* (JDK 1.2.2) como base para o desenvolvimento de uma infra-estrutura de metadados baseada nas especificações MOF 1.4 e JMI. CIM oferece suporte para o gerenciamento de metamodelos e metadados, persistência e serviços distribuídos utilizando *Object Request Brokers* (ORB) [UNISYS01].

A Figura 5.1 apresenta a estruturação do sistema GRM por meio do diagrama de pacotes UML, onde as funcionalidades nativas do software CIM estão contidas no pacote CIM. A versão padrão do CIM está disponível com as seguintes características:

- **CIM Workbench**: ferramenta administrativa (GUI-based) utilizada para configurar e gerenciar.
- **Um gerador de interfaces Java**: gera automaticamente interfaces Java para acesso e manipulação dos elementos de um metamodelo armazenado no repositório.
- **Um gerador de repositórios Java**: gera automaticamente código de um componente Java. Este componente Java é um software padrão de um repositório, o qual utiliza o metamodelo mapeado como sistema de informação.
- **Gerador de arquivos XMI**: permite a serialização de metadados no formato XMI.
- **O Modelo MOF**: modelo de informação do repositório.
- **Persistência utilizando arquivos no formato XMI**: o padrão MOF não define como os modelos são fisicamente armazenados no repositório, podendo armazená-los, por exemplo, em banco de dados relacionais, orientados a objetos ou simplesmente como arquivos XML.

CIM introduz alguns conceitos que não estão especificados em MOF:

- **Facility**: o mecanismo de persistência. CIM oferece persistência em memória, banco de dados utilizando tecnologia JDBC e arquivos XMI. No projeto GRM, foi utilizada a persistência no formato de arquivos, sendo os metamodelos e modelos persistidos no formato XMI.
- **Repositório**: em MOF, todo metamodelo possui um pacote mais externo (outermost package) onde estão todos os elementos pertencentes ao metamodelo. No entanto, a especificação não descreve como eles devem ser criados em um software. Em CIM, esses pacotes mais externos são chamados de repositório. Um repositório pode ser considerado como sendo um tipo de banco de dados em

memória onde os metaobjetos ou meta-metaobjetos pertencentes, respectivamente, ao modelo ou metamodelo podem ser manipulados e gerenciados através de suas interfaces.

- **Servidor de Metadados**: é a implementação das interfaces Java de um determinado metamodelo. Um Servidor de Metadados pode ser gerado para qualquer metamodelo, representando-o como metaobjetos expostos pelas suas interfaces.

5.3 Arquitetura do GRM

Atualmente, foi adicionado ao CIM um conjunto de ferramentas visando a construção do GRM. A figura 5.2 ilustra a arquitetura do GRM atual. Os softwares clientes adicionados ao CIM buscam criar um sistema de repositório completo de metadados baseados em MOF.

Estes softwares clientes usam interfaces JMI mapeadas a partir de um metamodelo para manipular os metaobjetos, podendo criar novos metaobjetos, ler os já existentes, atualizá-los e apagá-los do repositório.

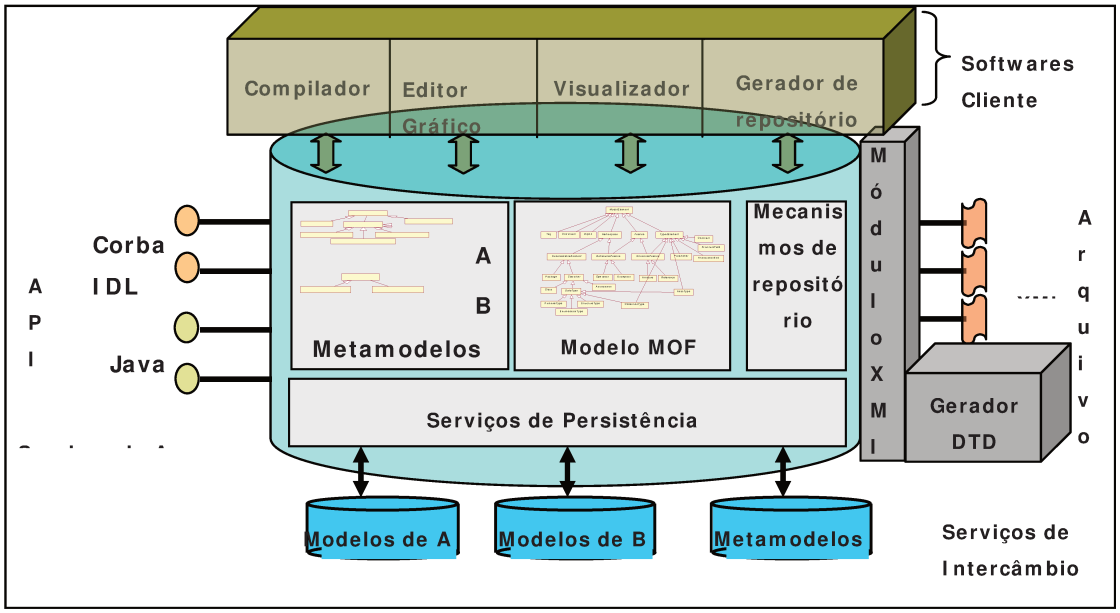

Figura 5.2 – Arquitetura do GRM

GRM provê um conjunto de ferramentas de metamodelagem para a definição e manipulação de metamodelos armazenados no repositório [FIGUEIREDO04a]. Ferramentas relavantes são descritas a seguir:

- **Compilador MODL**. A Meta-object Description Language [DSTC01] é uma notação textual para descrever metamodelos MOF. Um compilador MODL é uma alternativa em relação aos editores gráficos, provendo uma maneira para descrever metamodelos completos usando uma notação textual. A implementação de um compilador MODL foi importante porque no CIM o único modo de popular o repositório é através da importação de metamodelos expressos em arquivos XMI.
- **Gerador XML DTD**. Esta ferramenta gera automaticamente DTD XMI de metamodelos através da aplicação das regras de transformação definidas na especificação MOF. Os DTDs XMI gerados são usados para validar modelos perante os seus metamodelos e também para tornar persistente os metamodelos do GRM.
- **Visualizador de Metamodelo**. Permite a navegação através dos metamodelos e seus respectivos modelos.

5.4 Módulo Gerador DTD-XMI

Este módulo gerador de DTD XMI possui duas utilizações distintas: uma é a geração automática de DTDs XMI para persistência dos metamodelos do GRM, e a segunda, é a possibilidade de validação dos modelos perante o seu respectivo metamodelo. Esta validação determina se todos os elementos requeridos pelo metamodelo estão presentes no documento XMI que está sendo transferido. Além disto, é verificado se todos os atributos necessários aos elementos estão presentes e se os valores desses estão corretos.

Do ponto de vista de um usuário de um repositório MOF, o XMI representa uma maneira de transferir metadados entre repositórios, de um repositório para uma ferramenta ou aplicação e entre aplicações. Com isto, surge a necessidade de que se

possua uma forma de validar os arquivos que estão sendo trocados. Um Gerador XML DTD foi implementado para suprir esta necessidade.

Outro fato relevante, ferramentas e aplicações geralmente só precisam trabalhar com modelos e, com isto, precisam apenas do metamodelo correspondente, não precisando do Modelo MOF em si. As informações necessárias para transferência de modelos em formato XMI já podem estar no repositório, no formato de arquivos DTD XMI gerados automaticamente, podendo ser utilizados pelas ferramentas e aplicações para salvar e carregar os modelos em XMI.

A geração automática de DTDs XMI elimina a ocorrência de erros, geralmente obtidos quando a geração é realizada de maneira manual. A geração automática de DTDs XMI também proporciona o mapeamento correto e único para um dado metamodelo, pois é realizado segundo as regras de produção especificadas no padrão XMI.

5.4.1 Principais requisitos adotados

Os requisitos para a produção de DTD-XMI, segundo [OMG02b] são os seguintes:

- Todos os elementos XML definidos pela especificação XMI devem ser declarados no DTD.
- Cada construção de metamodelo (classes, atributos e associações) deve ter uma declaração de elemento correspondente e pode ter uma declaração de atributos. A declaração de elementos pode ser realizada através de entidades.
- Quaisquer elementos XML que representem extensões ao metamodelo devem ser declarados em um DTD interno ou externo.

5.4.2 Detalhes de implementação

As principais classes implementadas para a produção do módulo DTD-XMI são apresentadas na Tabela 5.1, juntamente com a sua descrição. Exceção apenas relativa a classe *AbstractWorkbench*, citada pela sua importância dentro do sistema. Os nomes das classes e métodos implementados na dissertação estão todos em inglês, mantendo o padrão do *Open Source* CIM utilizado.

Tabela 5.1 – Principais classes do módulo DTD-XMI

Nome	Descrição
WalkerPackageDTD	Consiste em uma matriz n-dimensional, possuindo operações de acesso e leitura de elementos contidos no metamodelo, por exemplo, findAllM2Package() e getElements().
AbstractWorkbench	Classe principal do sistema GRM, a qual executa todas as principais operações, por exemplo, criar e apagar facilidades, servidores e repositórios.
DTDElement	Consiste em uma classe, onde atributos são utilizados para armazenagem de informações como namespace, o objeto e um vetor com o conteúdo deste objeto. Possui operações de acesso e leitura de objetos e verificação de restrições sobre estes.
DTDProduction	Classe a qual possui operações para a construção de DTDs-XMI a partir da leitura de um objeto WalkerPackageDTD.
FixedContent	Classe onde está declarado o conjunto fixo de elementos que necessitam ser acrescentados no início do DTD XMI.

5.4.3 Funcionamento do Módulo DTD-XMI

Inicialmente, é selecionado um repositório de metadados no GRM através da ferramenta administrativa CIM *Workbench*. A Figura 5.3 apresenta a tela da ferramenta administrativa onde o usuário possui acesso aos repositórios e servidores de metadados e as funcionalidades do GRM. Na figura está selecionado o repositório de dos pacotes do metamodelo "rdb", o qual será utilizado como exemplo na demonstração dos detalhes de implementação.

Após a seleção, o usuário solicita a geração do XML-DTD através da seleção do *menu* correspondente. Esta seleção é capturada pelo *listener* existente na classe *AbstractWorkbench*.

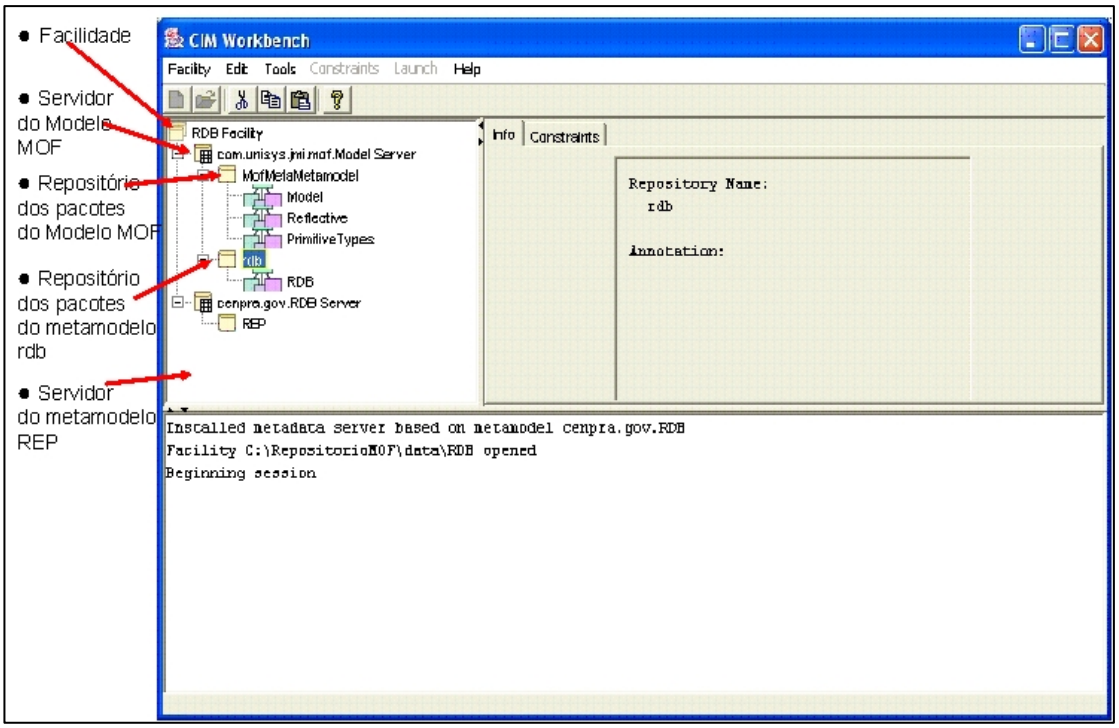

Figura 5.3 – Tela da ferramenta administrativa do sistema GRM

AbstractWorkbench é classe principal do sistema GRM, a qual executa todas as principais operações, por exemplo, criar e apagar facilidades, servidores e repositórios. Esta irá chamar o método *generateDTD()*.

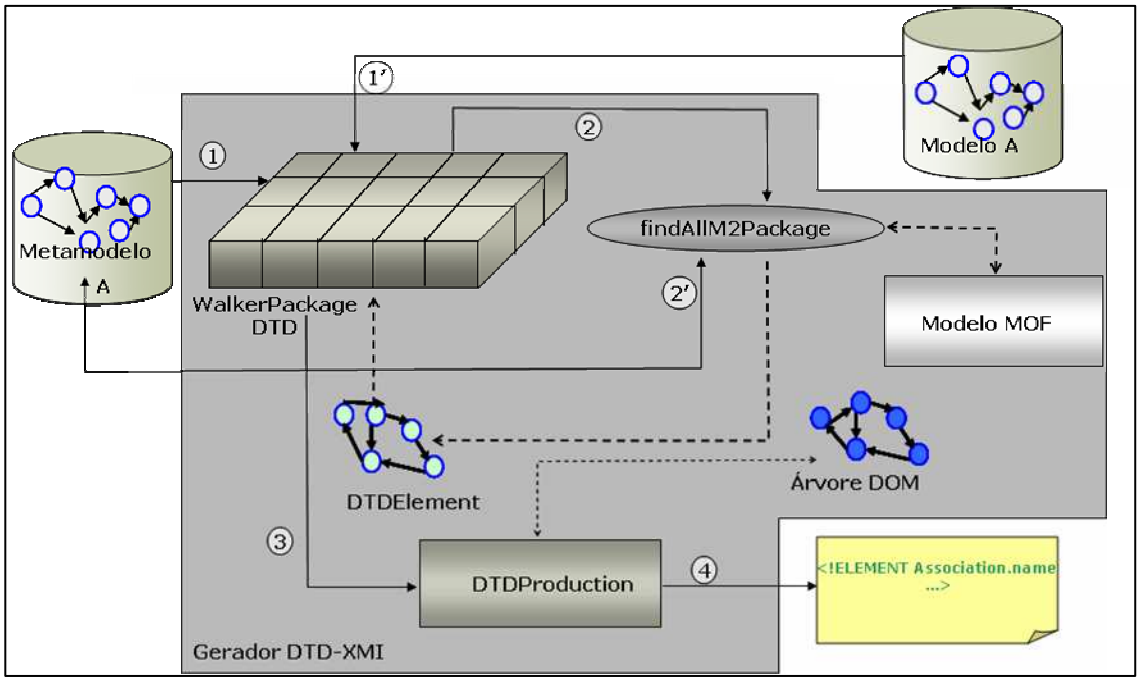

Figura 5.4 – Arquitetura do Módulo Gerador DTD-XMI

O método *generateDTD()* irá instanciar um objeto *WalkerPackageDTD*. O objeto *WalkerPackageDTD* é uma estrutura n-dimensional, preenchida com os elementos do metamodelo que está sendo transformado, possuindo métodos para a leitura de todos os elementos contidos no metamodelo. A Figura 5.4 apresenta a arquitetura do módulo gerador DTD-XMI, destacando as classes *WalkerPackageDTD*, *DTDProduction* e *DTDElement*.

Em repositórios de metadados, o foco principal está nas construções de nível M2 e nas instâncias de nível M1. Estas instâncias são referenciadas como metaobjetos, isto é, objetos representando metadados.

Elementos existentes no Modelo MOF (pacotes, classes e associações) possuem *namespace,* isto é, podem conter outros elementos declarados dentro deles. A cada elemento deste tipo encontrado no metamodelo uma nova dimensão é criada para os elementos pertencentes ao *namespace*.

A leitura dos elementos de um metamodelo é realizada através do método *findAllM2Package()* pertencente à classe *WalkerPackageDTD*. Este método utiliza as interfaces reflexivas para descobrir o tipo do objeto que está sendo lido no momento.

Todas as interfaces geradas a partir de um metamodelo herdam de um conjunto fixo de interfaces chamado interfaces reflexivas MOF. As interfaces reflexivas possuem todas as funcionalidades que as interfaces específicas do metamodelo possuem, porém elas são menos conveniente para usar, pois suas assinaturas não são adaptadas para o metamodelo específico. Esta característica de não ser customizada para um respectivo metamodelo gera a vantagem de uso geral.

O método *findAllM2Package()* descobre, identifica e adiciona os elementos pertencentes ao metamodelo do repositório selecionado. Para isto, o metamodelo selecionado é verificado e, com isto, descoberto o pacote mais externo (outermost package) através do método *getTopLevelPackage()*, pertencente a classe *ModelRepository*.

Após a descoberta do pacote mais externo, é utilizada a interface reflexiva *RefPackage* para acessar a coleção de objetos e suas associações contidas no pacote. Através das operações *refAllPackages*, *refAllClasses* e *refAllAssociations* são retornados todos os pacotes, classes e associações, respectivamente. Estes elementos são adicionados a matriz existente no objeto *WalkerPackageDTD* e uma nova dimensão é criada a partir de cada um.

Metaobjetos de outros tipos como exceção, tipos de dados, constantes são descobertos através da utilização do método *getContents()* disponível em metaobjetos que possuem *namespace*. Este método retorna uma lista de todos os elementos contidos a um determinado metaobjeto. Esta lista é percorrida através do método *addContents()* da classe *WalkerPackageDTD*, onde o seu conteúdo será adicionado na dimensão relativa ao elemento ao qual estes estão contidos. São utilizadas interfaces reflexivas para a leitura de cada metaobjeto encontrado. Por exemplo, em cada classe definida há uma interface reflexiva correspondente *RefObject* e, para cada Associação, há uma *RefAssociation*. Estas interfaces fornecem todas as informações dos elementos que representam. Para cada elemento identificado é instanciado um novo *DTDElement*, o qual irá armazenar o objeto e *namespace* ao qual ele pertence. Neste momento, o *DTDElement* é adicionado ao *WalkerPackageDTD*.

Depois de percorrido todos os elementos do metamodelo, uma instância da classe *DTDProduction* é criada, passando o *WalkerPackageDTD* como parâmetro. É realizada a leitura completa do *WalkerPackageDTD* e para cada *DTDElement* é criado o código correspondente no arquivo DTD XMI. Com isto, é gerado um novo documento DTD XMI para o metamodelo selecionado no início da operação.

No *DTDProduction* todas as regras da especificação XMI versão 1.2 para a produção de DTDs XMI são respeitadas. Exemplo destas regras: nomes dos elementos de um DTD gerado pelo gerador são nomes qualificados. Um nome qualificado consiste em um nome de namespace XML opcional, mais o sinal de dois pontos e um nome de Classe, Pacote ou Associação.

A especificação define um conjunto fixo de elementos que necessitam ser acrescentados no início do DTD XMI. Este conjunto fixo está declarado em uma classe chamada *FixedContent*. Os elementos fixos definem a estrutura de um arquivo XMI, como por exemplo, documentos XMI devem possuir o "XMI" como o seu elemento raiz e que o elemento “XMI” possui o elemento “XMI.content”, o qual contêm os dados atuais transferidos.

A implementação do DTD-XMI seguiu a regra de sintaxe de criação de DTD sem entidades. Nesta abordagem, elementos DTD são criados para cada Pacote, Classe e Associações que não possuem referências [OMG02b]. Esta abordagem possui vantagens de simplicidade e de um mapeamento limpo entre as construções do metamodelo e os elementos DTD-XMI usados para representá-los. A principal desvantagem dessa abordagem está na repetição de elementos que poderiam ser substituídos por entidades, não criando desta forma DTD-XMI extensos.

A Figura 5.5 mostra o diagrama de seqüência simplificado de um exemplo de geração de um DTD-XMI, ilustrando o processo anteriormente descrito. Exemplo de DTD-XMI gerado pelo módulo é demonstrado na Figura 5.6, onde um fragmento de um documento XMI representando um metamodelo contendo um pacote chamado RDB é importado e carregado pelo sistema GRM e, com isto, gerado o seu respectivo DTD-XMI.

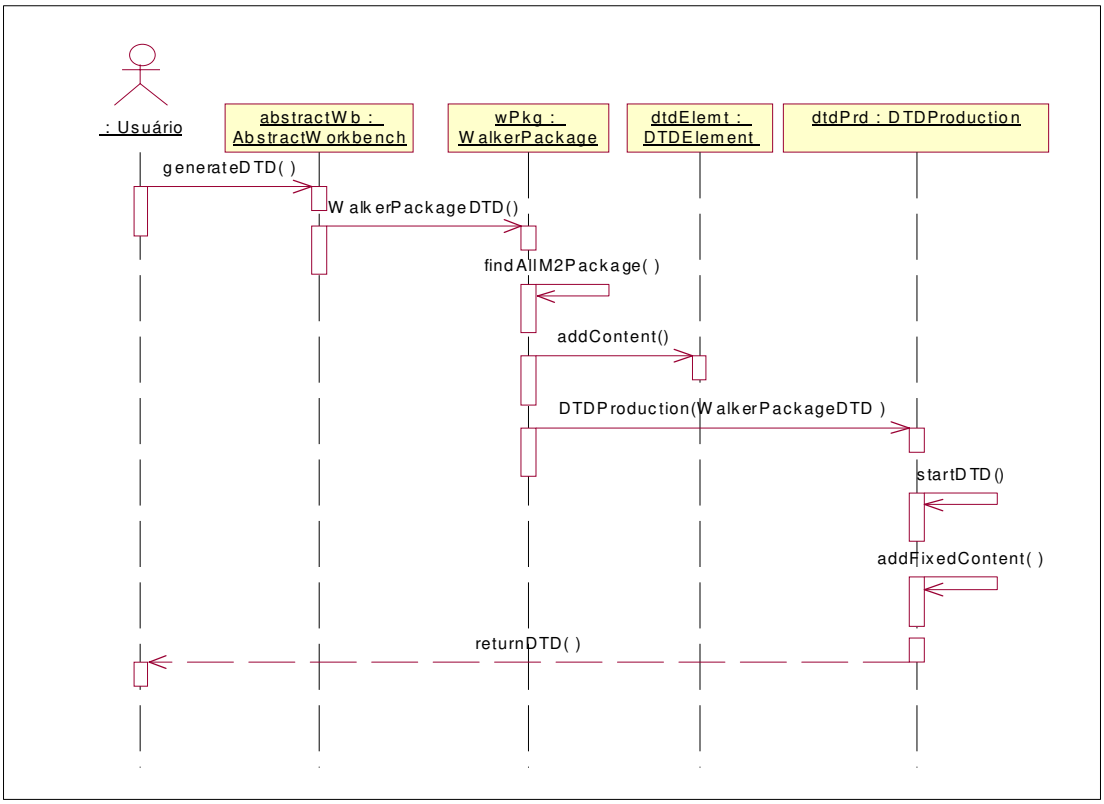

Figura 5.5 – Diagrama de Seqüência simplificado da geração do DTD

```
<?xml version="1.0" encoding="UTF-8" ?>
<XMI>
<XMI.header>
    <XMI.metamodel xmi.name="Model" xmi.version="1.2"/>
</XMI.header>
<XMI.content>
  <Model.Package xmi.id="a0"
    xmi.uuid="6b9f1027a2579218:003043d2:000000f41181ecd3:810c"
    name="RDB" annotation="" isRoot="false" isLeaf="false" isAbstract="false"
    visibility="public_vis"/>
</XMI.content>
</XMI>
```

```
<!ELEMENT ModelPackage.name (#PCDATA|XMI.reference)*>
<!ATTLIST Association %XMI.element.att; %XMI.link.att; name CDATA
#IMPLIED annotation CDATA #IMPLIED isRoot CDATA #IMPLIED isLeaf
CDATA  #IMPLIED isAbstract CDATA #IMPLIED visibility CDATA
#IMPLIED isDerived CDATA #IMPLIED>
```

Figura 5.6 – Fragmento de um documento XMI e o seu respectivo DTD

5.4.4 Representação DTD XMI para classes de metamodelo

A Tabela 5.2 exemplifica como as informações das classes de metamodelos foram representadas em um DTD XMI gerado pela classe *DTDProduction*. A geração seguiu as regras de produção da especificação XMI, versão 1.2, declaradas na notação Extended Backus Naur Form (EBNF) conforme demonstrado na Tabela 5.3.

Tabela 5.2 – Representação DTD de classes de um metamodelo

Especificação de	Exemplo
Classe: A representação de uma classe "C" de um metamodelo que não possua atributos, associações ou relacionamentos.	<!ELEMENT C (XMI.extension)*> <!ATTLIST C %XMI.element.att; %XMI.link.att; >
Herança: é representada declarando os atributos, referências e composições juntamente com as declarações locais da classe. O exemplo ilustra uma classe C1 que possui uma superclasse C0. A classe C0 possui um atributo a0, referência r0 e composição c0.	<!ELEMENT % C1 (C0.a0| C1.a1| C0.r0| C1.r1| C0.comp0| C1.comp1| XMI.extension)*> <!ATTLIST C1 a0 CDATA #IMPLIED A1 CDATA #IMPLIED r0 CDATA #IMPLIED r1 CDATA #IMPLIED %XMI.element.att; %XMI.link.att; >
Atributo: a declaração de atributos em XMI utiliza duas representações: atributos XML ou elementos XML. O exemplo ao lado ilustra as duas formas de maneira simples.	Como elemento XML: <!ELEMENT c.a (tipo especificado)*> e como atributo XML: a CDATA #IMPLIED
Associação: cada papel da associação é representado em uma entidade XML, um elemento XML e um atributo XML. Para um papel "r" da associação temos o exemplo ao lado.	<!ELEMENT r (content)*> r IDREFS #IMPLIED

A tabela acima demonstra apenas situações simples encontradas em um metamodelo. XMI possui várias limitações que devem ser tratadas na produção de um DTD XMI de um metamodelo mais complexo. Por exemplo, o fato do padrão XML não possuir uma forma de representar herança dificulta a representação desta pelo padrão XMI.

Tabela 5.3 – Regra de produção para DTD em EBNF

```
1. <DTD> ::= <1b:FixedContent>
                <1d:XMIAttList>?
                <2:PackageDTD>+
1a. <XMIFixedAttribs> ::= "%XMI.element.att;" "%XMI.link.att;"
1b. <FixedDeclarations> ::= //Fixed declarations//
1c. <Namespace> ::= ( //Name of namespace// ":" )?
1d. <XMIAttList> ::= "<!ATTLIST" "XMI" ("xmlns:"
                        //Name of namespace// "CDATA" "#IMPLIED")+
                    ">"
```

No contexto XMI, heranças são tratadas realizando uma cópia completa de todos os atributos, referências e composições da superclasse para a subclasse. Herança múltipla é tratada de maneira que atributos, referências e composições que ocorrem mais de uma vez na hierarquia de herança sejam incluídos apenas uma vez. No módulo gerador de DTD XMI, quando um novo elemento está sendo instânciado como *DTDElement*, é realizada uma verificação pelo método *verifyInheritance()* para atender esta restrição.

No caso de atributos, a ferramenta considera a utilização de elementos XML e atributos XML na geração de DTD-XMI conforme a especificação. Se os tipos de atributos do metamodelo são primitivos, elementos XML são declarados. No caso de *enumerations*, tanto elementos quanto atributos XML são gerados. A Tabela 5.4 demonstra como estão sendo declarados os tipos boolean e enumeration.

Tabela 5.4 – Representação DTD de atributos de metamodelos do tipo boolean e enumeration

Tipo do atributo	Exemplo
Enumeration	<!ELEMENT a EMPTY> <!ATTLIST a xml.value (enum1| enum2| ...) #REQUIRED>
Boolean	<!ELEMENT a EMPTY> <!ATTLIST a xml.value (true| false) #REQUIRED>

5.4.5 Testes

Para a validação dos arquivos DTD XMI gerados foram realizados alguns testes. Os testes realizados foram a geração de arquivos DTDs XMI relativos aos metamodelos de

padrões OMG, tais como UML e CWM, e a geração de arquivos DTDs XMI para validação de documentos XMI. Estes testes demonstraram alguns problemas nos arquivos gerados, sendo um dos principais problemas o tamanho dos DTDs-XMI.

Uma solução seria a adoção de outra abordagem de implementação, utilizando entidades XML na geração dos DTD XMI. O uso de entidades torna o DTD gerado mais compacto, ao mesmo tempo em que mantém todas as informações do metamodelo. Esta abordagem evita o grande número de repetições, tais como "ModelElement.name" que ocorrem na solução adotada para a implementação do módulo DTD-XMI.

A abordagem utilizada, sem o uso de entidades, falha pelo fato da herança possuir um papel fundamental em MOF. Por exemplo, um atributo de uma classe próxima ao topo da árvore de herança de um metamodelo torna-se um atributo de cada uma de suas subclasses.

Um DTD-XMI representando o metamodelo UML versão 1.3 produzido utilizando entidades XML iria conter aproximadamente 8 invocações de entidades no elemento DTD para a classe Class [OMG02a]. Enquanto, no DTD-XMI gerado pelo módulo DTD-XMI aparecem 39 declarações de elementos individuais para a mesma classe Class.

Outros testes foram realizados carregando no repositório documentos XMI relativos aos metamodelos das especificações OMG UML [OMG03a] e OMG CWM [OMG03b]. Arquivos DTDs foram gerados e comparados aos existentes nas respectivas especificações. Além disto, o DTD-XMI correspondente ao metamodelo UML foi utilizado na validação de documentos XMI de modelos UML.

Ferramentas de editoração XML (Altova XMLSpy [ALTOVA05], MS XML Validation Tool [MS05]) foram utilizadas para a verificação dos arquivos gerados, verificando se estavam bem formados. Além disso, foi realizada a validação de documentos XMI com os seus respectivos DTD-XMI.

5.5 Gerador HTML

O sistema GRM possuía uma deficiência de um meio rápido e independente de plataforma para a visualização de modelos e metamodelos armazenados no repositório. O Gerador HTML foi implementado visando suprir essa deficiência.

O Gerador HTML permite a visualização de metadados e metamodelos armazenados no repositório através de páginas HTML. A notação utilizada para descrever os metadados foi baseada na linguagem *Meta-Object Definition Language* (MODL) [DSTC01]. MODL é uma linguagem textual cuja sintaxe é baseada em Corba IDL.

Apesar de não se tratar de uma norma, a linguagem MODL é adotada para a definição de metamodelos em GRM, inseridos na ferramenta através do compilador MODL implementado em [FIGUEIREDO04b]. A Figura 5.7 apresenta um exemplo de utilização do gerador através da demonstração do metamodelo RDB.

O sistema GRM torna persistentes as suas informações em documentos XMI. Portanto, poderia ter sido implementado um módulo visualizador que utilizasse folhas de estilos XML para a transformação de documentos XMI em outros formatos para apresentação. Nesta primeira versão do gerador essa possibilidade não foi considerada, mas deixada como um dos trabalhos futuros relativos a este módulo.

Outra forma seria a simples apresentação dos documentos XMI referentes ao metadado selecionado na ferramenta de administração do sistema. Esta possibilidade foi descartada pelo fato de XMI ser focado na interpretação por máquinas, resultando em documentos de difícil compreensão por seres humanos.

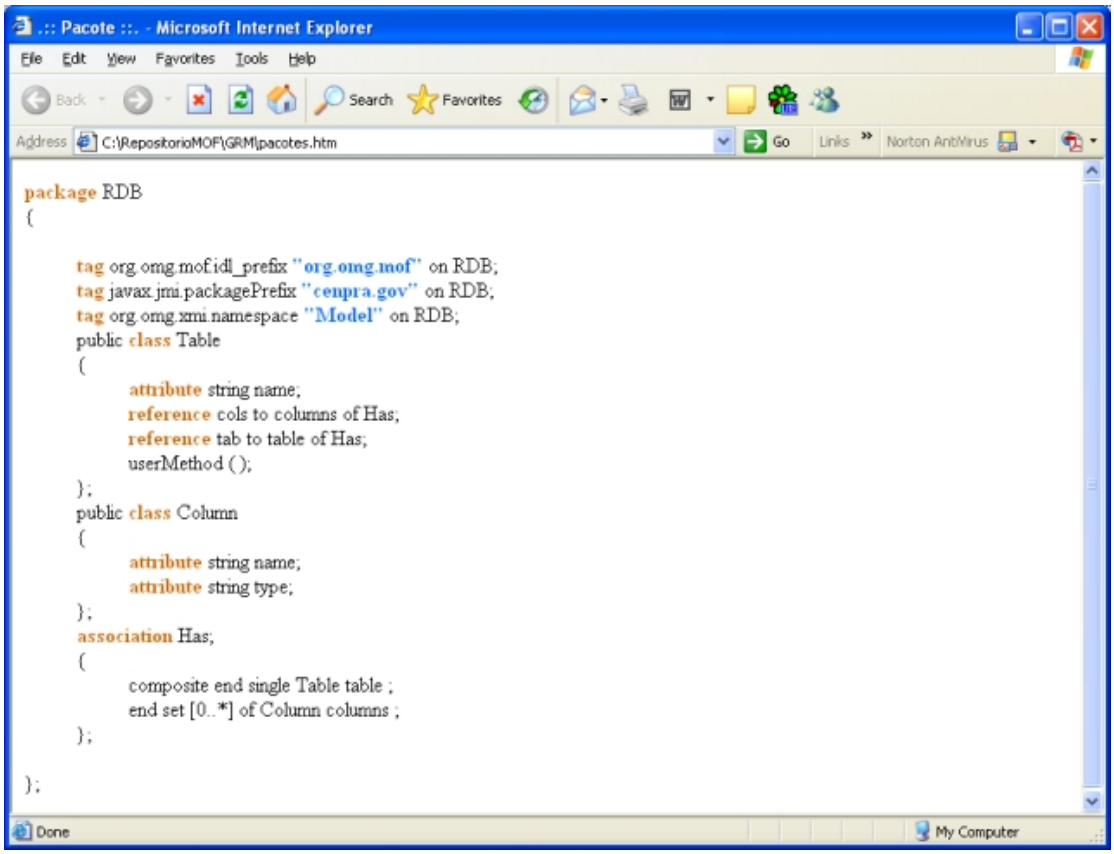

Figura 5.7 – Página HTML gerada pelo Gerador HTML.

5.5.1 Detalhes de Implementação

O gerador HTML utiliza a estrutura já criada para o Gerador de DTD XMI para percorrer os metamodelos e obter todos os seus elementos, elementos aninhados e elementos externos referenciados ou que possuam dependência. A partir dessa estrutura, em vez de montar um arquivo DTD XML, é montado um arquivo HTML, onde todos os elementos estão representados.

A principal diferença está na montagem do arquivo HTML em relação ao DTD. Em um DTD-XMI todas as heranças devem ser declaradas dentro do próprio documento. Desta forma, se uma classe A herda de uma classe B declarada em um outro metamodelo, esta terá todos os seus atributos e operações importadas e declaradas dentro do próprio elemento representando a classe A que está sendo criado no DTD.

No gerador, para cada referência a uma superclasse herdada de outro modelo ou uma importação de um modelo pela declaração *import* é construído uma página HTML completa, representado o modelo importado ou que contenha a superclasse declarada.

Essa característica possibilita a navegação entre modelos ou metamodelos através da ferramenta de visualização.

A Tabela 5.5 apresenta a identificação e a descrição das principais classes implementadas na geração da ferramenta de visualização de metadados.

Tabela 5.5 – Principais classes do Gerador HTML

Nome	Descrição
HTMLExtents	Gera uma representação do tipo String que pode ser adicionado ao arquivo HTML.
BuildViewModel	Possui duas operações principais: generateView(), realiza a leitura do conteúdo de um modelo, e getOutermostPkg(), retorna o pacote mais externo do modelo.
ViewContents	Classe onde toda a estrutura do arquivo HTML é montada. Nesta classe é identificado o elemento que está sendo lido do WalkerPackageDTD e adicionado as constantes a ele pertencente. Estas constantes estão definidas na classe ViewConstants.
ViewPanel	Esta classe possui operações para a apresentação e invocação do arquivo HTML correspondente ao metamodelo ou modelo selecionado na ferramenta administrativa do sistema GRM. Possui também métodos para atualização do painel de informação da ferramenta administrativa.
ViewModel	Realiza a leitura e acesso a todos os elementos pertencentes ao modelo que está sendo lido.
ViewMetamodel	Realiza a leitura e acesso a todos os elementos pertencentes ao metamodelo que está sendo lido.
ViewConstants	Possui a definição de todas as constantes necessárias para a criação da página HTML contendo informações de metamodelo ou modelo.

5.6 Considerações Finais

Diferentes domínios podem criar diferentes metamodelos, cada um específico para a sua área de atuação, com as informações-chave que podem ser transferidas e utilizadas

na geração de um DTD-XMI. Conforme os domínios crescem ou sofrem transformações, o padrão XMI pode ser aplicado novamente para a geração de novos DTDs-XMI, refletindo as mudanças ocorridas.

A capacidade de representação do conhecimento de um determinado domínio através de MOF permite o intercâmbio de suas informações através de documentos XMI e, além disso, a validação dessas informações trocadas.

O módulo gerador DTD-XMI torna a geração desses DTDs XMI automática, eliminando a ocorrência de erros através de um mapeamento correto e único para um dado metamodelo.

6 CENÁRIO DE USO

A pressão contínua para corte de custos e a necessidade de se prover um amplo e eficiente acesso público a informação tem levado governos de todo o mundo a desenvolver meios eletrônicos de atenderem aos cidadãos. Governos buscam, a exemplo do setor privado, a adoção de novas tecnologias que possam aumentar a produtividade do setor. O fornecimento e suprimento de informações e serviços governamentais através de ambiente eletrônico denominam-se Governo Eletrônico.

Atualmente, a importância de se utilizar tecnologias de informação e comunicação na administração pública tornou-se parte fundamental da agenda governamental. Governo eletrônico significa uma re-interpretação e também uma revolução no gerenciamento do poder público, suportado pela formação de tecnologia com a intenção de suportar as mudanças governamentais [GARCIA04]. Governo eletrônico poderá ser utilizado como um instrumento de administração pública, tornando possível servir melhor o cidadão.

Governos operam em uma grande variedade de áreas, algumas similares ao setor privado, enquanto outras são completamente diferentes. Por este motivo, algumas propriedades singulares do setor público devem ser ressaltadas [CUNHA04]:

- Governos, geralmente, são maiores e mais complexos que qualquer empresa privada.
- Comunicações com a sociedade e com outras entidades de governo devem ser realizadas através de padrões legais regidos, freqüentemente, por leis.
- Garantia de igualdade de acesso aos serviços públicos.
- Governos lidam somente com serviços.
- O pagamento dos serviços governamentais é realizado pela imposição de impostos ou taxação.

Serviços de governo são organizados pelos tipos de usuários, geralmente classificados em: cidadãos, empresas e governos. Serviços são disponibilizados aos cidadãos e a empresas em pontos únicos de acessos, reduzindo a redundância de coleções de dados e utilizando tecnologias existentes para melhor comunicação entre eles.

Inúmeros trabalhos estão sendo realizados nesta área visando solucionar problemas não apenas tecnológicos, mas também questões administrativas para que governo eletrônico não seja apenas a transferência da burocracia de governo para a Internet.

O desenvolvimento de novas aplicações governamentais está sendo fundamentado no paradigma de computação orientada a serviço. Este paradigma considera os serviços como elementos fundamentais para o desenvolvimento de aplicações. A aplicação do paradigma na Web manifesta-se através da tecnologia *Serviços Web* [SANTOS04].

O gerenciamento de metadados é um facilitador natural de arquiteturas orientadas a serviços (*Service Oriented Architecture* – SOA). Este capítulo visa mostrar a importância de metamodelos em uma solução de governo eletrônico, ressaltando a utilização de documentos e DTDs XMI entre as aplicações governamentais.

6.1 Interoperabilidade de metadados em plataformas de governo eletrônico

Conforme ilustra a Figura 6.1, governos são compostos por diferentes unidades administrativas como secretarias e agências públicas, formando uma entidade multi-organizacional. Geralmente, serviços governamentais são oferecidos através de um ponto único de acesso, necessitando acessar diferentes unidades de governo para que possam ser completados. Esta colaboração entre diferentes unidades irá prover serviços mais ajustados aos seus potenciais usuários (cidadãos e empresas).

A estrutura organizacional de governo remete a um ambiente de tecnologia de informação heterogêneo, combinando tecnologias distintas de sistemas operacionais, linguagens de programação, tecnologias de rede e de banco de dados. Sistemas especializados são desenvolvidos para agências específicas de governo visando a realização de determinado serviço.

Uma plataforma de governo eletrônico (PGovE) está sendo desenvolvida no Centro de Pesquisa Renato Archer (CenPRA). O objetivo principal deste projeto é a criação de uma infra-estrutura para suportar o desenvolvimento de serviços colaborativos de Internet no domínio de governo.

Uma peça fundamental nesta infra-estrutura é o sistema GRM onde artefatos complexos como modelos de dados legados e modelos de processos de negócio podem ser armazenados. O sistema GRM suporta a definição e armazenamento de diferentes metamodelos, através do uso de MOF, possibilitando a interoperabilidade entre diferentes metadados.

Duas soluções principais podem ser ressaltadas nesta arquitetura devido ao uso intenso de metadados:

- Integração de serviços: provendo a capacidade de compor diferentes serviços básicos para a formação de novos serviços mais complexos que atendam melhor as necessidades dos cidadãos.
- Integração de dados legados: provendo mecanismos de modelagem para representação e acesso.

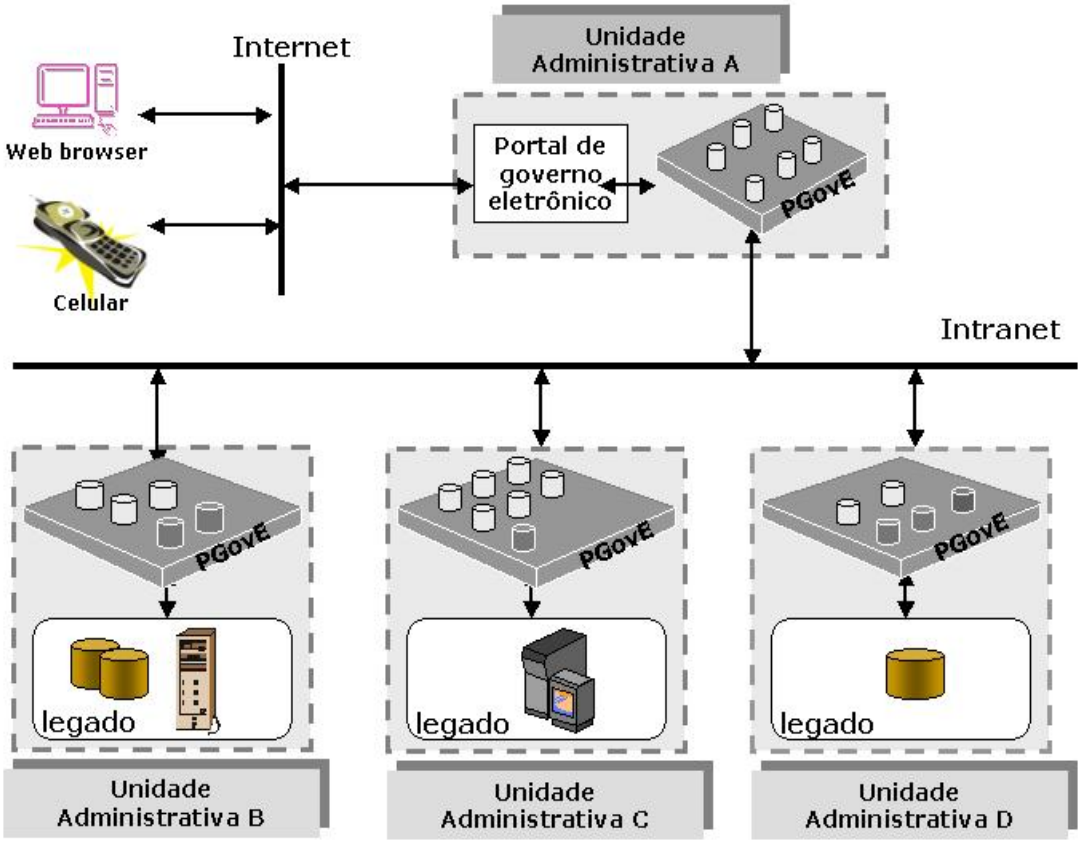

Figura 6.1 – Uma utilização da plataforma PGovE [FIGUEIREDO04]

6.1.1 Integração de serviços

Aplicações governamentais podem ter suas funcionalidades expostas através de interfaces de serviços web. Estas interfaces são descritas e acessadas através de WSDL e SOAP, respectivamente.

Há uma convergência entre as estratégias de diferentes governos eletrônicos que apontam para serviços integrados entre agências [TIZZO04]. Serviços específicos de diferentes unidades de governo poderiam, então, ser compostos para a realização de serviços mais complexos. A composição destes serviços é capturada através da definição um *workflow*.

Um *workflow* é definido como a automação de um processo de negócio (em sua totalidade ou em parte) durante a qual documentos, informações e tarefas são transferidos de um participante ao outro (para que seja realizada uma ação), em conformidade com um conjunto de regras de procedimento [MEJIA02].

Processos de negócios capturam a lógica de coordenação dos vários recursos envolvidos na realização do serviço. Processos de negócio são modelados com semântica bem definida e de fácil entendimento a fim de se obter modelos precisos e amistosos. A adoção de um padrão de metamodelo para processos de negócio baseado em MOF supre a necessidade de uma linguagem para a modelagem de processos de negócio.

O perfil UML para aplicações empresariais distribuídas orientadas a objeto (*Enterprise Distributed Object Computing* – EDOC) é composto de várias partes reunidas sob a arquitetura de colaboração empresarial (*Enterprise Collaboration Architecture* – ECA) [OMG04a], sendo cada uma definida através de metamodelos baseados em MOF.

O perfil para processo de negócio (*Business Process Profile* – BPP) representa um dos metamodelos EDOC, provendo conceitos que permitem a modelagem processos de negócio no estilo *workflow* no contexto de componente e entidades que modelam o serviço [OMG04a].

Neste contexto, *workflows* podem ser definidos utilizando o metamodelo BPP e, então, armazenados no sistema GRM através do compilador MODL. Dessa forma, o sistema GRM é utilizado para o armazenamento do metamodelo e seus respectivos modelos.

No sistema GRM é necessário criar um repositório de metadados BPP para o armazenamento do seu respectivo metamodelo e modelos, sendo assim, os passos necessários são: o metamodelo BPP é definido em linguagem MODL ou por meio de documento XMI por um especialista do domínio; através do compilador MODL ou da importação do documento XMI o metamodelo é armazenado no sistema. Dessa forma, o código Java do software servidor do sistema de repositório de modelos descritos pelo metamodelo pode ser gerado automaticamente.

As características estruturais de governos como, por exemplo, descentralização e autonomia, devem ser preservadas quando uma solução eletrônica for adotada. Por exemplo, serviços relativos a veículos devem ser mantidos sob a responsabilidade independente do departamento de trânsito. A Figura 6.1 demonstra o ambiente federativo de um governo, onde um serviço deve ser capaz de consultar o repositório de entidades de outra unidade administrativa.

A interoperabilidade entre os repositórios é permitida através da utilização de interfaces padronizadas e intercâmbio de documentos no formato XMI, conforme a Figura 6.2. Assim sendo, modelos BPP armazenados no sistema GRM podem ser intercambiados com outras instituições de governo, através do uso de documentos XMI, para que possam ser utilizados em outras aplicações para a realização de serviços mais complexos.

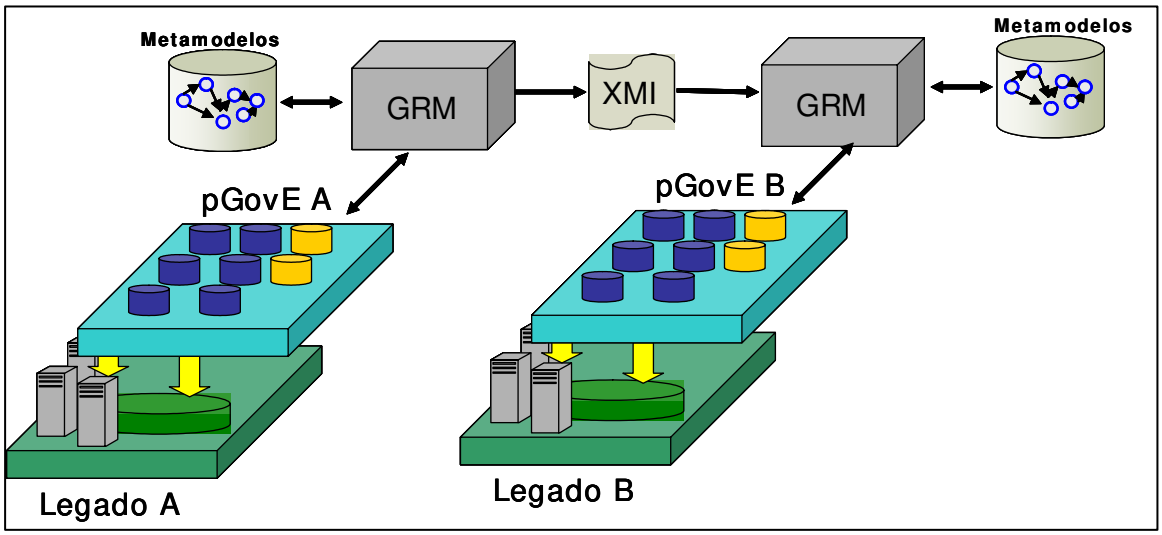

Figura 6.2 – Interoperabilidade entre repositórios através de XMI

O módulo DTD-XMI ficaria encarregado de realizar a validação dos documentos XMI transmitidos e, além disso, poderia também realizar o armazenamento persistente do metamodelo BPP no formato DTD XMI. A Figura 6.3 demonstra este cenário.

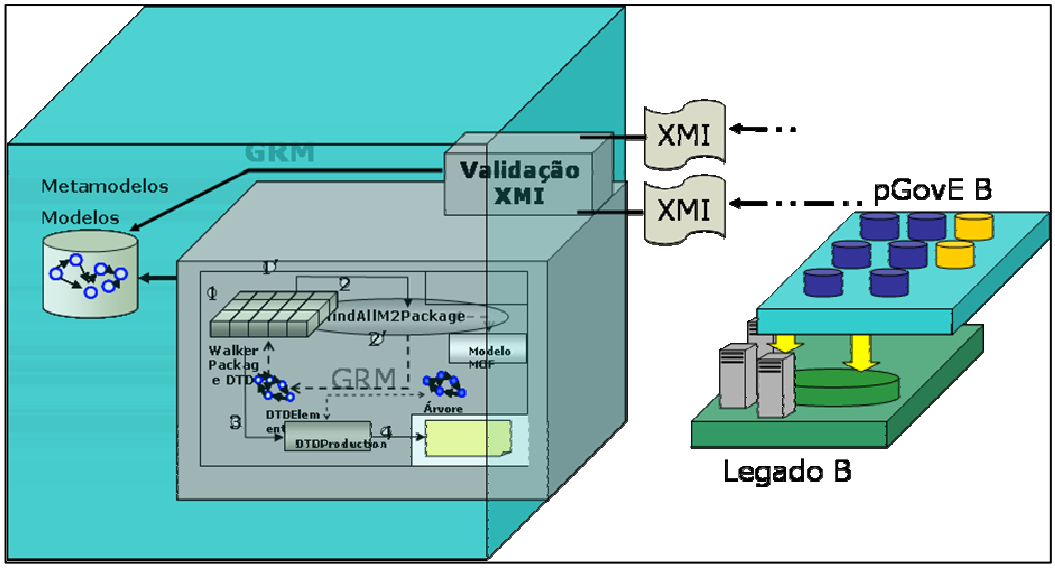

Figura 6.3 – Utilização do módulo DTD-XMI dentro da pGovE

A Figura 6.4 representa a seqüência de atividades de um serviço de governo de emissão de segunda via de carteira de identidade. O serviço precisa integrar diferentes serviços básicos para a realização do seu processo de negócio. Os serviços básicos são: verificação e cancelamento da via corrente da carteira de identidade no sistema do Instituto de Identificação Civil (IIC), verificação de antecedentes criminais no sistema Criminal e a emissão da segunda via através do sistema do IIC.

O ponto importante que deve ser ressaltado é que a modelagem neste caso é realizada independente de plataforma de execução, com isto, adotando a MDA, processos de negócios definidos em modelos BPP podem ser mapeados para diferentes tipos de tecnologias.

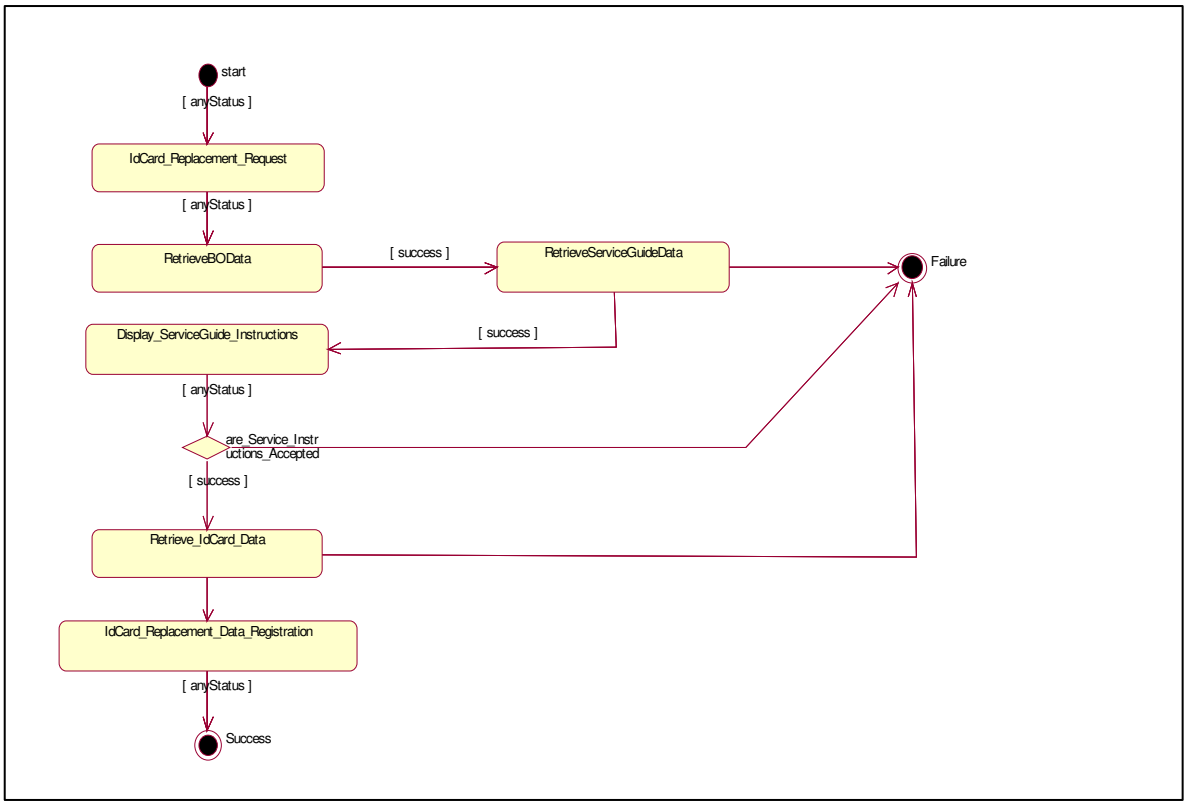

Figura 6.4 – Seqüência de atividades de um serviço de governo de emissão de segunda via de carteira de identidade.

6.1.2 Integração de dados legados

O perfil para integração de aplicações empresariais (*UML Enterprise Aplication Integration* – EAI) [OMG04b] foca na integração de sistemas legados, fornecendo metamodelos com definições detalhadas para semântica de colaboração, especificação detalhada de fluxo de mensagens no desenvolvimento de subsistemas integrados e para invocar e traduzir informações de aplicações.

Vários serviços disponibilizados pelos governos eletronicamente podem ser iniciados em servidores Unix ou Windows, mas irão ser completados em mainframes, onde a maioria dos dados governamentais reside.

EAI Common Application Metamodel (CAM) é um grupo de metamodelos de interfaces de aplicações empresariais, linguagens de programação e representação física. CAM provê representações de tipos de dados e mapeamentos de armazenagem para suportar transformações de dados de uma linguagem de programação e plataforma em outras.

TypeDescriptor (TD), *TypeDescriptor Language* (TDLang) e *Basic Mapping Support* (BMS) são metamodelos CAM que fornecem suporte para transformação de dados em um ambiente de integração de aplicações. A Tabela 6.1 apresenta os metamodelos e a sua respectiva descrição. Os metamodelos CAM são baseados em MOF, sendo assim, instâncias M2 do Modelo MOF.

Tabela 6.1 – Metamodelos CAM de representação física

Metamodelo	**Descrição**
TypeDescriptor (TD)	Apresenta um meio de descrever tipos de implementação como, por exemplo, arrays e tipos estruturados, de maneira independente de linguagem e plataforma. Seu metamodelo é representado na Figura 6.5.
TypeDescriptor Language (TDLang)	Servem como classes bases para metamodelos de linguagens provendo uma camada de abstração entre o metamodelo TD e um metamodelo de linguagem de programação, tais como, C, COBOL e PL/I.
Basic Mapping Support (BMS)	Captura o metadado associado a telas formatadas para aplicações de mainframe.

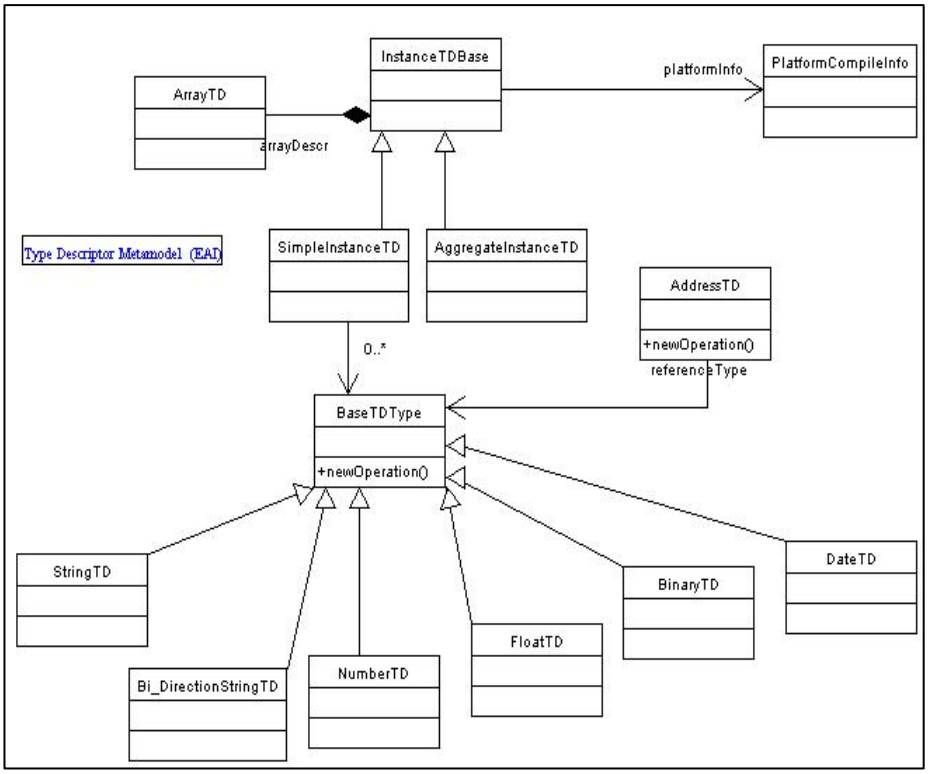

Figura 6.5 – Metamodelo Type Descriptor

Neste contexto, o sistema GRM é utilizado para o armazenamento dos metamodelos e seus respectivos modelos. Sendo o módulo DTD-XMI utilizado novamente para a validação dos modelos intercambiados entre diferentes plataformas de governo. A Tabela 6.2 mostra uma parte do DTD-XMI do metamodelo Type Descriptor gerado pelo módulo DTD-XMI, podendo ser utilizado na validação de dcumentos XMI trocados entre ferramentas e repositórios em um ambiente de governo eletrônico.

Tabela 6.2 – Exemplo de parte do DTD-XMI gerado a partir do metamodelo TD

```
<!-- PACKAGE: TypeDescriptor                                              -->
<!-- ___________________________________________________________________ -->
<!-- CLASS: AggregateInstanceTD                                           -->
<!-- ___________________________________________________________________ -->
<!ELEMENT AggregateInstanceTD.union EMPTY >
<!ATTLIST AggregateInstanceTD.union
            xmi.value (true | false) #REQUIRED>
<!ELEMENT AggregateInstanceTD (InstanceTDBase.offsetFormula |
                               InstanceTDBase.contentSizeFormula |
                               InstanceTDBase.allocSizeFormula |
                               InstanceTDBase.accessor |
                               InstanceTDBase.formulaInBit |
                               AggregateInstanceTD.union |
                               XMI.extension|
                               InstanceTDBase.languageInstance |
                               InstanceTDBase.platformInfo |
                               InstanceTDBase.arrayDescr)* >
<!ATTLIST AggregateInstanceTD
            offsetFormula CDATA #IMPLIED
            contentSizeFormula CDATA #IMPLIED
            allocSizeFormula CDATA #IMPLIED
            accessor(readOnly| writeOnly | readWrite | noAccess)#IMPLIED
            formulaInBit (true | false) #IMPLIED
            union (true | false) #IMPLIED
            languageInstance IDREFS #IMPLIED
            platformInfo IDREFS #IMPLIED
            %XMI.element.att;
            %XMI.link.att;
>
```

6.2 Considerações finais

A ferramenta de visualização permite em ambos os casos demonstrados a leitura de modo fácil e prático, devido ao seu acesso via browser, de qualquer metadados ou metamodelo armazenado no sistema GRM.

Nos dois casos citados, os metamodelos ou os modelos serão lidos pela ferramenta de visualização e, em seguida, seria montada uma página HTML representando o metadado selecionado.

Esta ferramenta é importante dentro do sistema GRM, pois é a única forma de visualização rápida das informações armazenadas no repositório.

7 CONCLUSÃO

A Internet e a Web aceleraram as expectativas de se obter interoperabilidade entre aplicações. O uso de metadados possibilitou a integração e a interoperabilidade de sistemas de informação. No entanto, um grande limitador de interoperabilidade atualmente ainda está na incompatibilidade de metadados.

Em ambientes de sistemas de computação distribuídos, os diferentes padrões de metadados existentes dificultam a integração e a interoperabilidade de aplicações. Arquiteturas de metadados estão sendo introduzidas para lidar com o problema de heterogeneidade entre os diversos tipos de metadados, a fim de se obter interoperabilidade. Neste contexto, foi desenvolvido o padrão *Meta-Object Facility* (MOF) pelo *Object Management Group* (OMG).

A especificação *Meta Object Facility* define uma linguagem abstrata e um *framework* para especificar, construir e gerenciar metamodelos independentes de tecnologias. MOF possui definições precisas do significado das características e dos atributos das instâncias dos metamodelos, permitindo mapeamentos destas características para linguagens e formatos de intercâmbios específicos.

MOF fornece mapeamentos para tecnologias específicas permitindo o gerenciamento e o intercâmbio de metamodelos e modelos. O formato para intercâmbio de modelos, *XML Metadata Interchange* (XMI), permite exportar ou importar modelos e metamodelos compatíveis com o padrão de metamodelagem MOF. Por ser baseado em XML, o XMI se tornou um formato de troca de metadados e metamodelos independente de middleware.

Todo este cenário gerou a necessidade de desenvolvimento de um sistema gerenciador de metadados, nascendo então o sistema GRM. Desta forma, foi estabelecido um conjunto de ferramentas para descrever, acessar e manipular metadados. A implementação de um módulo dentro do sistema GRM capaz de mapear metamodelos armazenados no sistema em DTDs XMI, permitiu que os documentos XMI pudessem ser validados perante o seu respectivo DTD. Documentos XMI válidos permitem certificar que todos os elementos e atributos definidos em um metamodelo estão representados.

Similarmente, os valores do conteúdo dos elementos e dos atributos também são verificados se estão em conformidade com o DTD.

A necessidade de visualização dos modelos e metamodelos armazenados no GRM levaram à implementação do gerador HTML. A implementação utilizou parte da estrutura codificada para a leitura dos metamodelos para a geração de DTD-XMI.

Em setembro de 2003, com o início do projeto eGOIA [eGOIA04], 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 de 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.

7.1 Principais Contribuições do Trabalho

As principais contribuições deste trabalho são destacadas a seguir:

- Construção de um módulo gerador de DTD-XMI e de um vizualizador de metamodelos e modelos MOF

  Os metamodelos armazenados no repositório podem ser utilizados para a validação dos modelos intercambiados entre repositórios ou aplicações e como um formato alternativo de persistência. A geração seguiu todas as regras de produção definida pela especificação XML Metadata Interchange (XMI) versão 1.2 para produção de DTDs-XMI de metamodelos baseados em MOF.

- Estudo e compilação das especificações MOF e XMI

  A dissertação discute a utilização das especificações MOF e XMI na obtenção de interoperabilidade de metadados. O estudo destas especificações é fundamental para o entendimento de novas propostas do consórcio OMG, por exemplo, MDA.

- Compilação das características principais dos padrões de interoperabilidade de metadados envolvidos na proposta SMIF.

7.2 Trabalhos Futuros

O sistema GRM continua em fase de desenvolvimento, possuindo uma versão operacional sem todas as funcionalidades implementadas. Novas funcionalidades precisam ser adicionadas ao sistema GRM, por exemplo, um módulo gerador de arquivos XML Schemas para metamodelos conforme a especificação XMI versão 2.0 [OMG03c]. Atualmente, o sistema oferece o compilador MODL, visualizador de metadados, geração de repositórios JMI, geração de XML DTD e intercâmbio de metadados no formato XMI.

Mudanças no visualizador de metadados devem ser realizadas para que este demonstre as informações dos metadados no formato Human-Usable Text Notation (HUTN) [OMG02c]. Além disso, o fato de HUTN ser uma notação textual e, com isto, possuir deficiência para uma visualização completa do modelo, aponta para a implementação de um meio de visualização gráfico.

A migração para MOF2 [OMG03e] é um caminho natural para o sistema GRM em um futuro próximo. Revisões no mapeamento de DTD-XMI para metamodelos precisarão ser realizadas uma vez que o módulo está desenvolvido com base na versão 1.4 de MOF.

8 REFERÊNCIAS BIBLIOGRÁFICAS

8.1 Artigos e Livros

[ATKINSON97] Atkinson, Colin - **Meta-Modeling for Distributed Object Environments** – 1º Conferência International Enterprise Distributed Object Computing (EDOC '97), Gold Coast, Australia, outubro de 1997.

[BERGHOLZ00] Bergholz, André. **Extending your Markup: An XML Tutorial.** IEEE Internet Computing, julho de 2000.

[BERNSTEIN99] Bernstein, P. A., Bergstraesser, T., Carlson, J., Pal, S., Shutt, D., **Versions and workspaces in Microsoft repository.** Proceedings ACM SIGMOD Conferência International em Gerenciamento de Dados, páginas: 532 – 533, Filadelfia, Pensilvania, EUA, 1999.

[BOS97] Bos, Bert. **The XML data model**, 1997**.**

http://www.w3.org/XML/Datamodel.html acessado em outubro de 2004

[BACKER97] Baker, N. L., Le Goff, J. **Meta Object Facilities and their Role in Distributed Information Management Systems**. Divisão ECP, CERN, Genova, Suiça. Conferência ICALEPCS97, Pequim, China, novembro1997.

[CLARCK97] Clarck, James. **Comparison of SGML and XML**. W3C - World Wide Web Consortium Note, dezembro de 1997.

[COSTA01] Costa, F. M.. **Combining Meta-Information Management and Reflection in an Architecture for Configurable and Reconfigurable Middleware.** Departmento de Computação da Universidade de Lancaster. Tese doutorado, agosto 2001.

[COVER98] Cover, Robin. **XML and Semantic Transparency**, Technology Reports, novembro de 1998.

http://www.oasis-open.org/cover/xmlAndSemantics.html acessado em outubro de 2004

[CRAWLEY97] Crawley, S., Davis S., Indulka J., McBride S., Raymond, K.. **Meta-meta is better-better**, CRC for Distributed System Technology (DSTC), University of Queensland, Australia. Conferência IFIP DAIDS, novembro de 1997.

[CUNHA04] Cunha, Amauri Marques e Costa, Paulo Mendez, **Towards Key Business Process for e-Government**, Núcleo de Computação Eletrônica – Universidade Federal do Rio de Janeiro, IFIP World Computer Congress, páginas 3-21, Toulouse – França, agosto de 2004.

[ECKERSON99] Eckerson, Wayne W. and Manes, Anne Thomas. **Paving the way for transparent application and data integration**, Sun Microsystems, Inc. novembro 1999.

[eGOIA04] eGOIA - **Electronic Government Innovation and Access**

http://www.egoia.info/ acessado em agosto de 2004

[ERDMANN00] Erdmann, M.; Decker, S.. **Ontology-aware XML-Queries**. WebDB 2000.

[ERNST97] Ernst, Johannes**. Introduction to CDIF**, setembro 1997

http://www.eigroup.org/cdif/intro.html acessado em setembro 2004.

[FIGUEIREDO03] 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, 2003.

[FIGUEIREDO04a] 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 de 2004.

[FIGUEIREDO04b] Figueiredo A., Mendes M., Kamada A., Borelli J., Rodrigues M., Damasceno L., Souza G., Tizzo N., **Applying MDA Concepts in an e-Government Platform**, DEXA 2004 - 3rd International Conference on Electronic Government, Zaragoza – Espanha, agosto de 2004.

[FLATSCHER96] Flatscher, Rony G.. **An Overview of the Architecture of EIA's CASE Data Interchange Format (CDIF).** Departamento de Gerenciamento de Sistemas de Informação, Universidade Economia e Administração de Negócios de Viena, janeiro de 1996.

[FRANKEL03] Frankel, David S.. **Model Driven Architecture - Applying MDA to Enterprise Computing**, 1º Edição, Ed. John Wiley and Sons, 2003.

[FRANKEL04] Frankel, David S.. XMI – **The OMG's Metadata Interchange Standard – Using model-centric architecture**, XML Journal, 2004.

[GARCIA04] Garcia G., Thais, Bigliazzi H., **Democracy in the Eletronic Government Era**. IFIP World Computer Congress, Toulouse – França, agosto de 2004.

[GLACE03] Glace, Jessica L. e Crawford, Mark R.. **Recommended XML Namespace for Government Organizations**, agosto de 2003.

[GOLDFARB96] Goldfarb, Charles F.. **The Roots of SGML.** SGML Source Home Page.

http://www.sgmlsource.com/history/roots.htm acessado em fevereiro de 2005

[JEFFERY98] Jeffery, Keith G. **METADATA: AN OVERVIEW AND SOME ISSUES**, ERCIM News No.35, Reino Unido, outubro de 1998.

[LAURENT99] Laurent, Simon St.. **Describing your Data: DTDs and XML Schemas**, XML.com, dezembro de 1999.

http://www.xml.com/lpt/a/1999/12/dtd/index.html acessado em outubro de 2004

[LEMESLE98] Lemesle, Richard. **Meta-modeling and modularity: Comparison between MOF, CDIF & sNets formalisms**. Conferência OOPSLA 98, Vancouver, British Columbia, Canada, outubro de 1998

[MEJIA02] 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.

[MILLER98] Miller, Eric. **An Introduction to the Resource Description Framework**, revista D-Lib, Corporation for National Research Initiatives, Virginia, EUA, maio de 1998.

[MOSHER02] Mosher, Chuck. **A New Specification for Managing Metadata**, abril 2002.

http://java.sun.com/developer/technicalArticles/J2EE/JMI/ acessado em outubro de 2004

[MSDN00] The Microsoft Developer Network (MSDN). **Microsoft Repository Type Information Model**, 2000.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/repospr/rpobjectarch_16lo.asp acessado em outubro de 2004.

[MSDN01] The Microsoft Developer Network (MSDN). **Microsoft: XML Interchange Format (XIF)**, 2000.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/repospr/rpobjectarch_16lo.asp acessado em outubro de 2004.

[MSDN04a] The Microsoft Developer Network (MSDN). **Schema Migration from Previous Versions of BizTalk Server**, 2004.

[MUTSCHLER00] Mutschler, Gene. **Generating MOF M1-level XMI Document**

Type Definitions, Unisys Corporation, 2000.

[PAPAZOGLOU03] Papazoglou, M.P e Georgakopoulos, D. **Service-Oriented computing**. Communications of ACM, 46(10), páginas 25-28, 2003.

[PATEL-SCHNEIDER02] Patel-Schneider, Peter; Siméon, Jérôme. The Yin/Yan **Web: XML Syntax and RDF Semantics**. WWW 2002, Honolulu, Hawaii, EUA, maio 2002.

[RAY01] Ray, Erick T.. **Learning XML**. Editora O'REILLY, 2001.

[SANTOS04] SANTOS, Ivo J. G. e Madeira, Edmundo R. M., **VM-Flow: Using Web Services Orchestration and Choreography to Implement a Policy-based Virtual Marketplace**, Instituto de Computação – Universidade Estadual de Campinas, IFIP World Computer Congress, páginas 266-285, Toulouse – França, agosto de 2004.

[SHAW81] Shaw, M.. ALPHARD: **Form and Content**, Springer-Verlag Inc., 1981.

[SILVA03] Silva, Cláudio Rodrigues Muniz da. **Um Modelo Independente de Plataforma para a Execução de Processos de Negócios em Arquiteturas baseadas em Modelos**. Universidade Estadual de Campinas - UNICAMP - Faculdade de Engenharia Elétrica e de Computação, julho 2003.

[SIMPSON00] Simpson, John E.. **Will XML replace HTML?** O'Reilly XML.com., dezembro de 2000.

http://www.xml.com/pub/a/2000/12/13/xmlhtml.html acessado em fevereiro de 2005.

[TIZZO04] Tizzo, N. P., Borelli, J. R., Mendes, M. de J., Damasceno, L.L., Batista K., dos Reis E., **Service Composition System apllied to E-Government**, IFIP World Computer Congress, Toulouse – França, agosto de 2004.

[WALSH99] Walsh, Norman**. Understanding XML Schema**. XML.com, julho de 1999.

http://www.xml.com/lpt/a/1999/07/schemas/index.html acessado em outubro de 2004

[WEGNER96] Wegner, Peter. **Interoperability.** ACM Computing Surveys, vol. 28, março 1996.

8.2 Especificações

[CDIF94] EIA/CDIF - Electronic Industries Association/CDIF - CASE Data Interchange Format, **Família de Padrões CDIF**, 1994.

[DC04] Dublin Core Metadata Element Set, versão 1.1, dezembro de 2004

http://dublincore.org/documents/dces/ acessado em fevereiro de 2005.

[DSTC01] DSTC - 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 outubro de 2004

[EIA94] EIA - Electronic Industries Association, **CDIF - CASE Data Interchange Format – Overview**, janeiro de 1994

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

[JCP02] JCP - 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 setembro de 2004

[ISO86] ISO - International Organization for Standardization**. ISO 8879:1986(E). Standard Generalized Markup Language (SGML)** - First edition. International Organization for Standardization, Genebra, Suiça, 1986.

[ISO98] International Organization for Standardization - **ISO. ISO/IEC JTC1/SC7 N1919. Information Technology - CDIF Framework** - Part 1: Overview. International Organization for Standardization, Genebra, Suiça, maio de 1998.

[ISO99] International Organization for Standardization - **ISO. ISO/IEC N2112:1999. Study Report on the Feasibility of Mapping Modelling Languages for Analisys and Design Models.** International Organization for Standardization, Genebra, Suiça, 1999.

[MDC99] Meta Data Coalition, **Open Information Model,** versão 1.0, abril de 1999.

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

[MSDN99] MSDN - Microsoft Developer Network. **Microsoft OIM Resources and Documentation**, 1999.

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/reposad/raoim_1lpq.asp acessado em outubro de 2004.

[OMG97] OMG - Object Management Group, Solicitação de Proposta (Request For Proposal - RFP). **Stream-based Model Interchange Format (SMIF)**. Documento OMG: ad/97-12-03, 1997.

www.omg.org/docs/ad/97-12-03.txt acessado em outubro de 2004

[OMG98] OMG - Object Management Group, **Universal Object Language**, versão 1.2, julho de 1998.

http://www.omg.org/docs/ad/98-07-07.pdf acessado em outubro de 2004

[OMG02a] OMG - 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 outubro de 2004

[OMG02b] OMG - Object Management Group, **XML Metadata Interchange (XMI) Specification**, versão 1.2, janeiro de 2002

http://www.omg.org/docs/formal/02-01-01.pdf acessado em outubro de 2004

[OMG02c] OMG - 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 outubro ce 2004

[OMG03a] OMG - 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 outubro de 2004

[OMG03b] OMG - 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 outubro de 2004

[OMG03c] OMG - 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 outubro de 2004

[OMG03d] OMG - Object Management Group, **MDA Guide Version 1.0.1**, junho de 2003

http://www.omg.org/docs/omg/03-06-01 acessado em outubro de 2004

[OMG03e] OMG - 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 outubro de 2004

[OMG04a] OMG - 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 outubro de 2004

[OMG04b] OMG - 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 outubro de 2004

[SAX02] **Simple API for XML (SAX)**, janeiro de 2002.

http://sax.sourceforge.net/ acessado em outubro de 2004

[W3C99a] W3C - World Wide Web Consortium. **XML Path Language (XPath)** Versão 1.0, fevereiro de 1999.

http://www.w3.org/TR/1999/REC-xpath-19991116 acessado em setembro de 2004

[W3C99b] W3C - World Wide Web Consortium. **Resource Description Framework (RDF) Model and Syntax Specification**, fevereiro 1999.

http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/ acessada em outubro de 2004.

[W3C99c] W3C - World Wide Web Consortium. **Resource Description Framework (RDF) Schema Specification**. Recomendação proposta pela W3C, março de 1999.

http://www.w3.org/TR/2000/CR-rdf-schema-20000327/ acessada em outubro de 2004.

[W3C00] W3C - World Wide Web Consortium. **Document Object Model (DOM)**, setembro de 2000.

http://www.w3.org/DOM/#what acessado em 24/08/2004

[W3C01a] W3C - World Wide Web Consortium. **XML Schema Part 1: Structures**, maio de 2001.

http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/ acessado em outubro de 2004.

[W3C01b] W3C - World Wide Web Consortium. **XML Schema Part 2: Datatypes**, maio de 2001.

http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/ acessado em outubro de 2004

[W3C02] Brown, Allen. **Web Services Glossary.** World Wide Web Consortium (W3C), junho de 2002.

[W3C04a] W3C - World Wide Web Consortium. **Namespaces in XML** versão1.1, fevereiro de 2004.

http://www.w3.org/TR/2004/REC-xml-names11-20040204 acessado em outubro de 2004.

[W3C04b] W3C - World Wide Web Consortium. **XML Specification v1.0** (Terceira edição), fevereiro de 2004.

http://www.w3.org/TR/2004/REC-xml-20040204

acessado em setembro de 2004

[W3C04c] W3C - World Wide Web Consortium. **Resource Description Framework (RDF): Concepts and Abstract Syntax**, fevereiro de 2004.

http://www.w3.org/TR/rdf-concepts/. acessado em setembro de 2004

[W3C04d] W3C - World Wide Web Consortium. **Resource description framework (RDF) primer**, fevereiro de 2004.

http://w3.org/TR/2002 acessado em setembro de 2004

[W3C04e] W3C - World Wide Web Consortium. **Resource description framework (RDF) Syntax**, fevereiro de 2004.

http://www.w3.org/TR/rdf-syntax-grammar/ acessado em setembro de 2004.

[W3C04f] W3C - World Wide Web Consortium. **Resource description framework (RDF) Semantics**, fevereiro de 2004.

http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ acessado em setembro de 2004.

[W3C04g] W3C - World Wide Web Consortium. **Resource description framework (RDF) Test Cases**, fevereiro de 2004.

http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/ acessado em setembro de 2004.

8.3 Software

[ALTOVA05] Altova, **Altova XMLSpy 2005**, 2005.

http://www.altova.com/products_ide.html acessado em fevereiro de 2005.

[BABYLON02] Babylon**, Babylon Dictionary**, 2002.

https://ecommunity.unisys.com acessado em agosto de 2004.

[MSDN05] MSDN - Microsoft Developer Network, **XML Validation Tool**, 2005.

http://www.microsoft.com/serviceproviders/downloads/xml_resource_listP62359.asp acessado em fevereiro de 2005.

[UNISYS02] Unisys eCommunity , **JMI-RI Documentation CIM**, Versão 1.3, junho de 2002