Uma Proposta para o Mapeamento entre a API DOM e o Padrão MOF

Hélio Lopes dos Santos
Universidade Federal de Pernambuco(UFPE), Centro de Informática(CIn),
Recife, Brasil, 50.732-970
hls@cin.ufpe.br

Maísa Soares dos Santos
Universidade Estadual do Sudoeste da Bahia (UESB), Departamento de Ciências Exatas (DCE),
Vitória da Conquista, Brasil, 45.000-000
maisa@uesb.br

Roberto Souto Maior de Barros
Universidade Federal de Pernambuco(UFPE), Centro de Informática (CIn),
Recife, Brasil, 50.732-970
Roberto@cin.ufpe.br

Abstract

This paper presents a metamodel aimed at mapping DOM interfaces into MOF structures such as packages, classes, associations, etc. The DOM API makes XML data available to programs in the form of objects. The MOF standard defines an abstract language and a framework aimed at specifying, implementing and managing platform independent metamodels. MOF tools will be able to manage all kinds of XML documents using the proposed metamodel.
Keywords: XML, DOM, MOF, Metamodel.

Resumo

Este artigo apresenta um metamodelo que faz o mapeamento entre as interfaces do DOM e os elementos do MOF como pacote, classe, associações, etc. A API DOM disponibiliza dados para as aplicações XML em forma de objetos. O padrão MOF define uma linguagem abstrata e um framework para especificação, construção e gerenciamento de metamodelos independentes de tecnologia de implementação. As ferramentas que implementam o padrão MOF poderão gerenciar qualquer tipo de documento XML utilizando o metamodelo proposto.
Palavras chave: XML, DOM, MOF, Metamodelo.

1. INTRODUÇÃO

Nos últimos anos, com o crescimento dos sistemas de informação, metadados tornaram-se peças chave no gerenciamento de todo o ciclo de vida desses sistemas. Muitos esforços recentes, tanto das áreas acadêmicas quanto da indústria, estão sendo concentrados em pesquisas relacionadas a metadados [8,1,18]. Um dos produtos dessas pesquisas é o MOF (*Meta Object Facility*) [10]. Este padrão foi projetado pela OMG (*Object Management Group*) especificamente para dar suporte à modelagem, gerenciamento e intercâmbio de metadados e está voltado ao contexto de sistemas de informação. O MOF possui como uma de suas características principais permitir interoperabilidade entre ferramentas de software que produzem e compartilham metadados.

A partir dessas pesquisas, verificou-se também que o padrão XML é adequado para ser utilizado como formato de descrição, armazenamento e intercâmbio de metadados, pois XML representa dados semi-estruturados, flexível, voltado para o contexto web e suportado por uma variedade de ferramentas [1]. Para dar suporte às suas funções, XML incorpora alguns padrões, como o DOM (*Document Object Model*) [4] utilizado para acessar e manipular documentos XML.

Este artigo propõe o projeto e a implementação de um metamodelo MOF para o DOM. Um metamodelo define um modelo para construir modelos. Ele é um conjunto de metadados inter-relacionados utilizados para definir modelos [8]. O metamodelo MOF proposto permite que qualquer documento XML seja gerenciado por uma ferramenta que implemente o padrão MOF. Atualmente, o MOF permite importar documentos XML apenas no formato XMI (*XML Metadata Interchange*) [7], que é o formato padrão para intercâmbio de metadados do MOF. Com o metamodelo proposto os usuários poderão gravar documentos XML com qualquer conjunto de tags, não necessariamente sendo XMI.

Uma das vantagens desta proposta é um conjunto de interfaces comuns para acessar e gerenciar os metadados descritos no metamodelo MOF proposto. Estas interfaces são equivalentes às interfaces do DOM e permitem gerenciar documentos XML em repositórios MOF. Atualmente, o MOF permite gerar interfaces CORBA, através do mapeamento IDL -> CORBA [10] e Java, através do JMI (*Java Metadata Interface*) [7].

Uma outra vantagem é a utilização do padrão XMI como uma alternativa a mais para intercâmbio dos metadados. O XMI é um padrão suportado por várias ferramentas nas mais diversas áreas como desenvolvimento de sistemas, por exemplo, ferramenta *Netbeans*[1]*; e*specificação e modelagem de sistemas, por exemplo, Rational Rose[2] e ARGO UML[3]; ferramentas de *Data warehouse,*como o DB2 Warehouse Manager[4] e Oracle Warehouse Builder[5]; repositórios de metadados como, por exemplo, Universal Repository (UREP)[6], MDR (*Metadata Repository*) [17] e dMOF [3].

Nas próximas seções são apresentados os trabalhos relacionados, o padrão DOM e MOF. Logo após, é apresentada a arquitetura e o padrão de mapeamento para interfaces Java e CORBA do MOF. É apresentado também o metamodelo MOF proposto para o padrão DOM, as interfaces geradas a partir deste metamodelo e um estudo de caso utilizando tais interfaces. E, na última seção, são apresentados as conclusões e os trabalhos futuros.

2. TRABALHOS RELACIONADOS

Outros metamodelos baseados no padrão MOF são o UML (*Unified Modeling Language*) [11] e o CWM (*Common Warehouse Metamodel*) [14].

A UML é uma linguagem para especificação, visualização, construção e documentação de artefatos de software. Ela provê uma linguagem de modelagem orientada a objetos, independente de plataforma, com conceitos bem definidos como classes, objetos, uses cases, associação, generalização, e diagramas gráficos.

O CWM é o padrão da OMG para integração de ferramentas de *Data Warehousing*. Essa integração se dá pelo compartilhamento de metadados. CWM define um conjunto de metamodelos para as diversas subáreas de *Data Warehousing*. CWM utiliza UML para definir os diversos metamodelos, chamados também de pacotes. Cada pacote possui um metamodelo sobre um domínio específico, mas existem dependências entre os mesmos.

O artigo [15] discute a integração entre padrões de metadados RDF (*Resource Description Framework*) e RDF Schema do W3C (*Word Wide Web Consortiun*) e o MOF da OMG. O artigo [16] apresenta um metamodelo para gerenciar metadados em XML e DTD, porém, tal metamodelo não foi modelado a partir da API DOM, além do que, o artigo discute mais o processo de desenvolvimento da aplicação de metadados, ressaltando as vantagens de se utilizar XML. Enquanto que este artigo foca mais no processo de mapeamento entre a API DOM e as interfaces geradas pelo repositório MOF.

Uma outra iniciativa importante da empresa *Sun Microsystem* é um metamodelo para a linguagem Java [2], que propõe uma maneira de gerenciar código de programas Java em repositórios MOF.

---

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

3. DOM – DOCUMENT OBJECT MODEL

A necessidade de prover acesso padronizado às informações em documentos XML levou a W3C (*World Wide Web Consortium*) a desenvolver o DOM. De acordo com a definição deste consórcio, DOM é uma API (*Application Programming Interface*) para documentos XML e HTML, que define a estrutura lógica de documentos e a forma como esses documentos são acessados e manipulados [4].

O objetivo principal de DOM é fornecer uma interface de programação padrão que possa ser usada em uma grande variedade de ambientes e aplicações. A especificação DOM oferece apenas as definições de interfaces para as bibliotecas de DOM, e estas interfaces são independentes de plataforma e de linguagem [4].

DOM é uma estrutura de dados abstrata que representa os documentos XML como uma árvore de nós. O DOM compreende um conjunto de interfaces que oferece os serviços para gerenciar documentos XML. Um *parser*, ou seja, uma ferramenta que implementa este conjunto de interfaces, lê todo o documento e constrói uma árvore de objetos na memória. A partir dessa árvore é possível navegar por sua estrutura, adicionar, modificar, e remover elementos do documento.

Um documento XML é uma árvore composta de nós de vários tipos, como: elemento, atributo, texto, comentário, entidade, instrução de processamento, etc. Para cada um desses tipos, DOM define uma interface. Todos os nós na árvore DOM são uma instância da interface Node. Através dela é possível navegar e manipular a árvore. Node fornece métodos para obter propriedades como nome, valor, tipo, lista de filhos e atributos de um nó, bem como para inserir, remover e substituir nós.

Um elemento do documento XML é representado pela interface Element. A maioria dos métodos de Element manipula atributos. Esta interface declara funções que possibilitam verificar a existência e obter os atributos de um elemento, e adicionar e remover atributos.

Um atributo é representado pela interface Attr. Os métodos definidos em Attr permitem obter o nome completo (*namespace* URI e nome local) do atributo, verificar se ele é especificado no documento ou não, obter o elemento ao qual o atributo pertence, além de obter e alterar o valor do atributo.

O W3C mantém atualmente três especificações para DOM: DOM 1 [4], DOM 2 [5] e DOM 3 [6]. O nível 2 de DOM acrescenta algumas funcionalidades ao DOM1, entre elas suporte a *namespace*, módulos de eventos, de folha de estilo, de atravessamento e intervalo de documentos e de visões. O DOM 3 focaliza o carregamento, salvamento, além de suporte para validação de documentos, entre outros.

4. MOF – META OBJECT FACILITY

O MOF define uma linguagem abstrata e um *framework* para especificação, construção e gerenciamento de metamodelos independentes de tecnologia de implementação. Alguns exemplos incluem o metamodelo UML, CWM e o próprio MOF. O MOF possui ainda um conjunto de regras para implementação de repositórios, que manipulam metadados descritos pelos metamodelos. Essas regras definem um padrão de mapeamento entre os metamodelos MOF e um conjunto de APIs para gerenciamento de metadados, instâncias do metamodelo. Por exemplo, o mapeamento *MOF -> IDL* (*Interface Definition Language*) é aplicado aos metamodelos MOF (UML, CWM) para produzir uma API CORBA que gerenciem os metadados, instâncias desses metamodelos. O mapeamento *MOF -> Java*, através do padrão JMI (*Java Metadata Interface*), define as mesmas regras de mapeamento para API Java. Este trabalho utiliza JMI.

A especificação MOF compreende o seguinte:

- Uma definição formal para o meta-metamodelo MOF, ou seja, uma linguagem abstrata para a definição dos metamodelos.
- As regras para o mapeamento dos metamodelos MOF para uma tecnologia de implementação, por exemplo, CORBA ou Java.
- Padrão XMI para intercâmbio, em XML, dos metadados e metamodelos entre as ferramentas. XMI define um conjunto de regras que mapeiam os metamodelos MOF e os metadados em documentos XML.

4.1 Arquitetura de Metadados da OMG

O MOF é um *framework* extensível, ou seja, novos padrões de metadados podem ser adicionados ao mesmo. Para isto, conta com uma arquitetura em quatro camadas, também chamada de *OMG Meta Data Architecture* [10, 13], mostrada na Tabela 1.

Tabela 1 – A arquitetura de metadados da OMG

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

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

4.2 O Modelo MOF

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

O MOF é descrito em termos dos seus próprios conceitos, ou seja, ele é auto descritivo. Isto significa que o modelo MOF é o seu próprio metamodelo. A especificação MOF o descreve de forma narrativa e através do uso de notação UML, tabelas e expressões OCL (*Object Constraint Language*) [11]. UML é utilizada apenas por possuir uma representação gráfica dos modelos, o que facilita a leitura por parte dos leitores. Ela não define a semântica do modelo MOF, que está completamente definida na especificação MOF e não depende da semântica de qualquer outro modelo. O MOF não especifica qual linguagem deve ser utilizada para definir as restrições dos seus metamodelos, porém ele utiliza como linguagem padrão a OCL.

O MOF é um modelo orientado a objetos e possui um conjunto de elementos de modelagem que são utilizados na construção dos metamodelos, incluindo regras para o seu uso. São exemplos destes elementos, classes, associações, pacotes, tipos de dados, constantes, entre outros.

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

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

Class proxy – Cada classe do nível M2 possui uma classe *proxy* correspondente. Existe um objeto *proxy* para cada classe do nível M2. Este tipo de meta objeto serve para produzir meta objetos do tipo *instance*.

Instance – As instâncias das classes do nível M2, ou seja, dos metamodelos, são representados pelos meta objetos do tipo *instance*. Um meta objeto *instance* manipula os estados dos atributos das classes do nível M2.

Association – Esses objetos manipulam as coleções de *links* correspondentes às associações do nível M2. São objetos estáticos e possuem como *containers* meta objetos do tipo *package*. As interfaces dos objetos do tipo *Association* disponibilizam operações para consultar um *link* no conjunto de *links; o*perações para adicionar, modificar e remover *links* a partir do conjunto; e operações que retornam todo o conjunto de *links*.

4.3 O Mapeamento MOF -> CORBA IDL/Java

Atualmente, faz parte da própria especificação MOF o mapeamento MOF -> CORBA IDL. A comunidade Java definiu o mapeamento MOF->Java e o chamou de JMI (*Java Metadata Interface*) [7]. As regras gerais de mapeamento são praticamente as mesmas para qualquer plataforma.

O resultado deste mapeamento é um conjunto de interfaces que permite ao usuário criar, alterar e consultar metadados, instâncias dos metamodelos MOF. Por exemplo, se as interfaces forem geradas a partir do mapeamento MOF->CORBA IDL, o usuário pode utilizar clientes CORBA para acessar as interfaces. Se for JMI, o usuário poderá utilizar clientes Java.

Esta seção descreve o padrão de herança para as interfaces mapeadas a partir dos metamodelos MOF. A Figura 1 apresenta um exemplo de metamodelo MOF expresso em UML que consiste de dois pacotes P1 e P2. O primeiro pacote P1 contém as classes C1 e C2, onde C2 é subclasse de C1 e uma associação A que conecta C1 e C2. O segundo pacote P2 é definido como subpacote de P1.

A Figura 2 apresenta o diagrama UML que mostra gráfico de herança gerado a partir do mapeamento MOF para Java. A raiz do gráfico é um grupo de interfaces que fazem parte do pacote reflexivo do MOF. Toda interface gerada a partir dos metamodelos herda direta ou indiretamente das interfaces reflexivas.

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

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

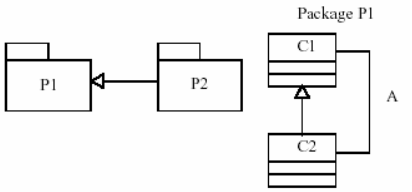

Figura 1 - Exemplo de um metamodelo MOF

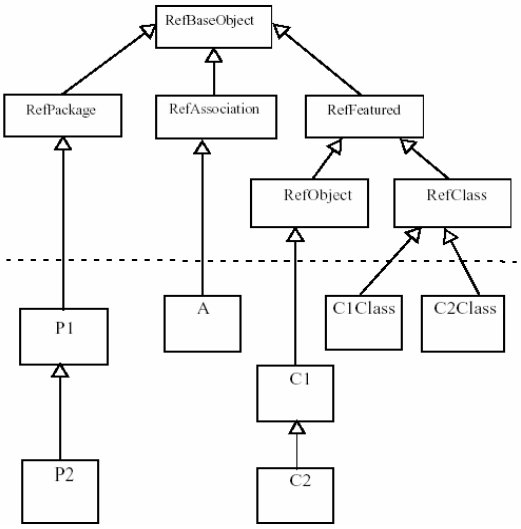

Figura 2 - Exemplo do metamodelo MOF mapeado para interfaces Java

Para o exemplo da Figura 2, foram geradas duas interfaces *package* referentes aos pacotes P1 e P2. O pacote P1, que não possuía supertipo, herdou de *RefPackage*, enquanto P2, que possuía, herdou de P1. Foi gerada também a interface *A* para a associação entre as classes C1 e C2 do metamodelo. Para cada classe do metamodelo foram geradas duas classes: uma para os meta objetos *instances* e outra para as *class proxy*. As interfaces *C1Class* e *C2Class* representam as interfaces *proxy* geradas, respectivamente, a partir das classes C1 e C2 do metamodelo, e herdam diretamente de *RefClass*. As interfaces *C1* e *C2* representam as interfaces para os meta objetos instances. Apenas a interface *C1*, cuja classe do metamodelo não possuía supertipo, herdou de *RefObject*.

5. O METAMODELO MOF PROPOSTO PARA O PADRÃO DOM

Diversas ferramentas suportam o MOF, como por exemplo, o MDR (*Metadata Repository*) [17], dMOF [3] e UREP (*Universal Repostiory*). Algumas fazem o mapeamento para CORBA (dMOF) e outras fazem para JMI (MDR). Algumas armazenam os objetos em SGBD (Sistema Gerenciador de Banco de Dados) oferecendo maior escalabilidade e robustez para as aplicações de metadados. Esta seção apresenta o metamodelo DOM que poderá ser utilizado por estas ferramentas para dar suporte ao gerenciamento de metadados descritos em XML.

O metamodelo proposto é um mapeamento direto entre as interfaces do DOM e os elementos do MOF como pacotes, associação, classes, etc. Por exemplo, a classe *DOMNode* é a mais genérica do metamodelo e representa a interface *Node* do DOM. Ela possui os atributos *nodename* e *nsURI* que representam, respectivamente, o nome do *nó* e o seu *namespace*. Muitos métodos da interface *Node* não foram mapeados para a classe *DOMNode*. Por exemplo, os métodos para manipular os *nós* filhos como inserir, remover, alterar, recuperar, etc. Esses métodos são encontrados na classe que representa a associação *DOMContains* do metamodelo.

Nem todas as subclasses usam efetivamente todos os atributos da classe *DOMNode*. Por exemplo, o nó comentário (*DOMComment*) não possui *namespace* e o seu valor é representado pelo próprio nome do *nó*.

A associação *DOMContains* permite a composição de *nós* XML. Alguns *nós* não podem conter outros, por exemplo, um nó texto ou atributo não pode conter outros nós. Para que o modelo fique consistente foi necessário adicionar *constraints* OCL ao metamodelo que restringem algumas composições possíveis segundo a própria especificação DOM.

A classe *XMLDocumentType* representa a interface *DocumentType* de DOM. Os textos são representados pela classe *DOMText*. Os elementos XML são representados pela classe *DOMElement* que representa a interface *Element* de DOM. A classe *DOMAttr* representa a iterface *Attr* de DOM que manipula atributos de XML. As interfaces *Entity*, *EntityReference, ProcessingInstrucion* são mapeadas, respectivamente para as classes *DOMEntity*, *DOMEntityReference, DOMProcessingInstrucion.* A classe DOMNodeValue não possui nenhuma interface correspondente na API DOM, porém ela foi acrescentada para oferecer um atributo *nodevalue* que é comum a algumas classes do metamodelo como entidades, atributos e instruções de processamento. Um documento XML é representado pela interface *Document* em DOM e pela classe *DOMDocument* no metamodelo MOF. Em DOM, os métodos para a criação dos *nós* como atributos e elementos fazem parte da interface *Document*. Em MOF, estes métodos farão parte das classes *proxys*.

A Figura 3 apresenta o metamodelo DOM em notação UML. Todos os componentes de um metamodelo MOF (classes, atributos, associações) devem estar dentro de um ou mais pacotes. Para o metamodelo DOM foi criado o

pacote *DOMMetamodel*. As classes que estão em cinza são abstratas, ou seja, elas não podem instanciar objetos diretamente, apenas suas subclasses.

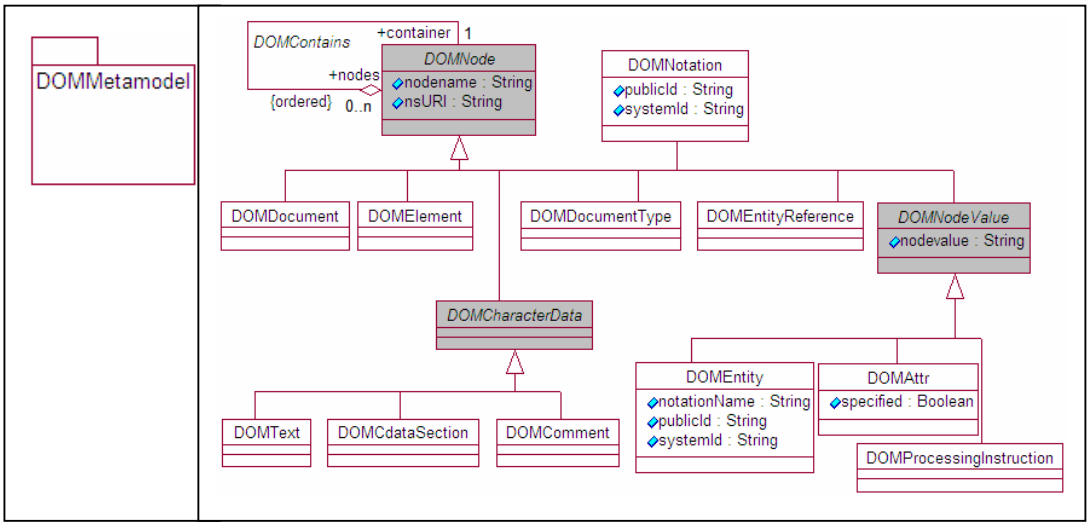

Figura 3 – O metamodelo DOM proposto – DOMMetamodel

Algumas informações não puderam ser apresentadas no metamodelo, como por exemplo, as restrições (*constraints)*. Estas foram acrescentadas diretamente ao documento XMI referente ao metamodelo DOM e limitam alguns tipos de agregações entre os *nós* XML. Como explicado na Seção 4.2, o MOF utiliza OCL como linguagem para a definição de restrições. A Figura 4 apresenta um exemplo de *constraint* descrita em XMI que diz que um documento XML deve possuir exatamente um elemento raiz. A ferramenta MDR ainda não suporta *constraints*, porém qualquer outra ferramenta que implemente a especificação MOF e suporte *constraints* poderia utilizá-las.

```
- <Model:Constraint annotation="" evaluationPolicy="deferred" language="OCL"
    xmi.id="C0">
    <Model.ModelElement.name
      xml:space="preserve">DOMDocumentContemUmDOMElement</Model.ModelElement.name>
    <Model.Constraint.expression xml:space="preserve">context document :
      DOMDocument inv C0:document.nodes->select(obj:DOMNode |
      (obj.oclIsTypeOf(DOMElement)))->size=1</Model.Constraint.expression>
  - <Model:Constraint.constrainedElements>
      <Model:ModelElement xmi.idref="a3D2A1A46009D" />
    </Model:Constraint.constrainedElements>
  </Model:Constraint>
```

Figura 4 – Exemplo de uma constraint OCL

O atributo *language* refere-se à linguagem em que está escrita a *constraint* enquanto que *evaluationPolicy* pode possuir os valores *immediate* ou *deferred* e se refere ao momento para a verificação de violação da *constraint*. Além dos atributos, existem também os subelementos *Model:ModelElement.name* e *Model:Constraint.expression* que armazenam, respectivamente, o nome e a própria *constraint*, e *Model:Constraint.contrainedElements* que diz a quais elementos do metamodelo tal *constraint* se aplica.

5.1 O As Interfaces Geradas a partir do Metamodelo DOM

Os usuários podem gravar metadados no repositório de duas maneiras. A primeira é importando os metadados descritos em documentos XMI. A segunda é através das interfaces geradas a partir do metamodelo. Essas interfaces permitem ao usuário criar, alterar e acessar as instâncias do metamodelo, que são os metadados, utilizando programas Java.

Foi gerado o documento XMI referente ao metamodelo construído e, após, foi importado para o repositório MOF, implementado pela ferramenta MDR.

Como foi visto na Seção 4.4, a especificação MOF define um conjunto de regras para a geração de um conjunto de interfaces a partir de um metamodelo específico. Uma das regras diz que para cada classe do metamodelo, duas interfaces são geradas: a primeira representa as instâncias da classe, os meta objetos *instance*, e a segunda representa uma classe *proxy*, os meta objetos *class proxy*.

A Listagem 1 apresenta a interface *DOMNode* gerada a partir do metamodelo DOM. Essa interface oferece métodos para acessar e manipular o estado dos atributos e as referências da classe do metamodelo. A Listagem 2 apresenta duas interfaces *proxys*: uma para *DOMNode* e outra para *DOMAttr*. As interfaces *proxys* possuem operações que

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

Listagem 1 - Interface Domnode do metamodelo DOM

```
public interface Domnode extends javax.jmi.reflect.RefObject {
    public java.lang.String getNodename();
    public void setNodename(java.lang.String newValue);
    public java.lang.String getNsURI();
    public void setNsURI(java.lang.String newValue);
    public dommetamodel.Domnode getContainer();
    public void setContainer(dommetamodel.Domnode newValue);
    public java.util.List getNodes();
}
```

Listagem 2 – Interfaces DomnodeClass e DomattrClass do metamodelo DOM

```
public interface DomnodeClass extends javax.jmi.reflect.RefClass {
}
public interface DomattrClass extends javax.jmi.reflect.RefClass {
          public Domattr createDomattr();
          public Domattr createDomattr(java.lang.String nodename,
                                       java.lang.String nsURI,
                                       java.lang.String nodevalue,
                                       boolean specified);
}
```

Para cada associação do metamodelo é gerada uma interface com um conjunto de métodos para acessar, alterar, inserir as instâncias das associações. Essas instâncias representam ligações entre os objetos, instâncias das classes do metamodelo. O metamodelo DOM possui a associação *DOMContains* que diz que um *nó* do tipo *DOMNode* poderá conter outros *nós* do mesmo tipo. A Listagem 3 apresenta a interface *DOMContains*.

Listagem 3 – Interface Domcontains do metamodelo DOM

```
public interface Domcontains extends javax.jmi.reflect.RefAssociation {
       public boolean exists(dommetamodel.Domnode nodes,
                                    dommetamodel.Domnode container);
       public java.util.List getNodes(dommetamodel.Domnode container);
       public dommetamodel.Domnode
                            getContainer(dommetamodel.Domnode nodes);
       public boolean add(dommetamodel.Domnode nodes,
                                     dommetamodel.Domnode container);
       public boolean remove(dommetamodel.Domnode nodes,
             dommetamodel.Domnode container);
}
```

Para cada pacote do metamodelo é gerada uma interface que contém métodos para acessar todos os objetos *proxys* referentes a todas as classes e associações do metamodelo. A Listagem 4 apresenta a interface para o pacote *DOMMetamodelPackage*, único pacote do metamodelo DOM.

Listagem 4 - Interface DommetamodelPackage do metamodelo DOM

```
public interface DommetamodelPackage extends
                                       javax.jmi.reflect.RefPackage {
    public dommetamodel.DomcdataSectionClass getDomcdataSection();
    public dommetamodel.DomcommentClass getDomcomment();
    public dommetamodel.DomdocumentClass getDomdocument();
    public dommetamodel.DomdocumentTypeClass getDomdocumentType();
    public dommetamodel.DomtextClass getDomtext();
    //unica associação do metamodelo
    public dommetamodel.Domcontains getDomcontains();
 ... }
```

Após a geração das interfaces, o próximo passo será a compilação dessas interfaces para serem utilizadas. A próximas seções ilustram o uso do metamodelo através das interfaces geradas.

5.2 O Mapeamento DOM -> MOF e MOF -> DOM

Uma das vantagens deste trabalho é permitir que repositórios MOF possam importar e exportar não somente documentos XMI, mas qualquer tipo de documento XML. Isto é importante, pois os usuários podem possuir ferramentas que produzem metadados em diversos formatos de XML, não necessariamente sendo XMI. Desta forma, basta importar os mesmos para o repositório MOF. Para isto, é necessário construir utilitários que façam o mapeamento entre os metadados descritos em XML para o seu metamodelo MOF.

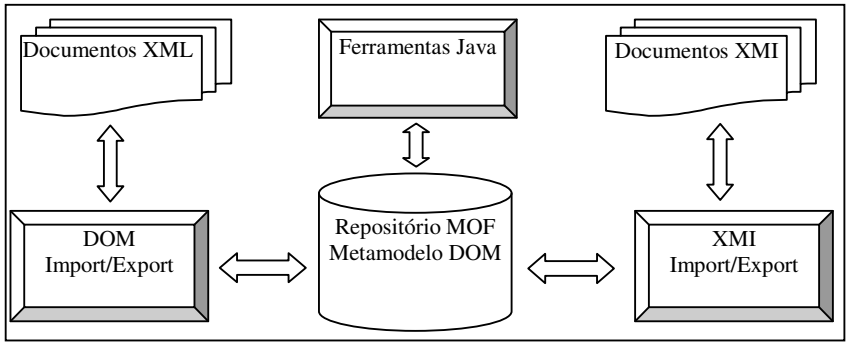

Figura 5 - Arquitetura dos Módulos Exportação e Importação de Documentos XML e XMI

A Figura 5 apresenta a arquitetura dos módulos que fazem a exportação e importação de documentos nos formatos XML e XMI para o repositório MOF. A importação e exportação de documentos XMI já é oferecida por qualquer ferramenta que implementa o MOF. Já a exportação e importação de documentos XML foi implementada através de um conjunto de métodos que transformam objetos do repositório MOF em objetos DOM e *vice-versa.* Além dos módulos de exportação e importação, o usuário poderá utilizar programas Java para acessar diretamente o repositório sem necessariamente usar as estruturas de DOM.

O processo de importação consiste em submeter o documento XML a um *parser* DOM, que processa o documento criando a estrutura de árvore de objetos na memória. Após, é necessário converter essa estrutura de objetos DOM em objetos do repositório MOF como instância do metamodelo DOM. O módulo *DOM Import* executa as seguintes tarefas:

1. Processar o documento XML, verificando se ele é bem formado, ou seja, se os dados do arquivo passado como parâmetro estão mesmo na sintaxe XML. Este passo criará a árvore de objetos DOM.
2. Iniciar o repositório MOF e pesquisar o repositório DOM, representado pelo metamodelo DOM. Esta tarefa consiste em achar uma instância da classe *MOFPackage* do metamodelo MOF cujo nome é *DOMMetamodel.*
3. Criar uma instância da classe *DOMDocument* que representa o novo documento a ser importado para o repositório.
4. Para cada objeto do tipo *Element* da estrutura DOM, criar uma instância da classe *DOMElement* no repositório MOF.
5. Para cada objeto do tipo *Attr* da estrutura DOM, criar uma instância da classe *DOMAttr* em MOF.
6. Similar aos passos 4 e 5, fazer os outros mapeamentos, como comentários, entidades, instrução de processamento, entre outros.
7. Além de criar os objetos, é necessário criar as ligações entre os mesmos através da classe *DOMContains,* única associação do metamodelo DOM.

Listagem 5- Parte do Código Java para Pesquisar um Documento XML no Repositório MOF

```
1 - if (pkg !=null){
2 -          javax.jmi.reflect.RefClass refclass =
                                    pkg.refClass("DOMDocument");
3 -          java.util.Collection c = refclass.refAllOfClass();
4 -          java.util.Iterator iter = c.iterator();
5 -          while(iter.hasNext()) {
6 -              Domdocument mofdoc = (Domdocument)iter.next();
7 -             if (doc.getNodename().equals(docname)){
8 -                  Node node =
                    DOMDocumentFactory.mof2dom(xmldoc,mofdoc,pkg);
9 -                  xmldoc.appendChild(node);
10-                 saveXML(xmldoc); }
          }
  }
```

O processo de exportação consiste na geração de documentos XML a partir do repositório MOF. Para isto, é necessário inicializar o repositório MOF e pesquisar o sub-repositório DOM, ou seja, o pacote *DOMMetamodel.*

Após, consultar a instância da classe *DOMDocument,* dentro do pacote, que representa o documento XML a ser exportado. Esta instância é passada como argumento do método *mof2dom* da classe *DOMDocumentFactory*. Este método gerará a estrutura DOM a partir do objeto, instância da classe *DOMDocument*.
A Listagem 5 apresenta uma parte do código Java utilizado para pesquisar um objeto, instância da classe *DOMDocument* no repositório MOF. O método *refClass* (linha 2) da interface reflexiva *Refpackage,* retorna a interface *proxy* de uma determinada classe do metamodelo que é passada como parâmetro. Neste caso, a classe é *DOMDocument* e a interface retornada é *DOMDocumentClass*. O método *refAllOfClass* retorna uma lista de todos os objetos instância de uma determinada classe do metamodelo, no exemplo *DOMDocument*. A próxima tarefa é pesquisar, através do atributo *nodename*, o objeto na lista de objetos e passar para o método *mof2dom* (linha 8) que retorna um *node* XML contendo todo o documento XML.

6. ESTUDO DE CASO

Além de importar/exportar documentos XML, uma outra maneira de gerenciar documentos XML é utilizar diretamente as interfaces geradas a partir do metamodelo. Os usuários podem construir programas clientes que conectem ao repositório MOF, obtenham o pacote referente ao metamodelo e utilizem-no na construção de novos metadados que serão instâncias do metamodelo. A Listagem 6 apresenta uma parte de um código em Java para acessar o repositório DOM, representado pelo pacote *DOMMetamodel*.
A linha 1 do programa *MDRManager.getDefault().getDefaultRepository()* retorna o repositório padrão armazenado. Este repositório vai ser sempre o próprio metamodelo MOF. Após, é necessário pesquisar no repositório padrão o pacote *proxy* referente ao metamodelo DOM. O método *repository.getExtent("dommetamodel"),* na linha 2, procura um determinado pacote *proxy* dentro do repositório MOF. Ele recebe o nome do pacote como parâmetro e retorna um objeto do tipo *RefPackage,* que faz parte do pacote reflexivo de JMI. Se a pesquisa ao pacote desejado for bem sucedida, o próximo passo será utilizar o pacote para gerenciar metadados descritos em XML.

Listagem 6 – Código Java para criar documentos XML no repositório MOF

```
1 - MDRepository repository =
                        MDRManager.getDefault().getDefaultRepository();
2 - DommetamodelPackage pkg =
            (DommetamodelPackage)repository.getExtent("dommetamodel");
3 - if (pkg !=null)   {
4 -  repository.beginTrans(true);
5 -  Domdocument docroot =
     pkg.getDomdocument().createDomdocument("DATAWAREHOUSES.xml","");
6 -  DomprocessingInstruction   pi   =
  pkg.getDomprocessingInstruction().createDomprocessingInstruction(
                        "xml","","version=\"1.0\" encoding=\"UTF-8\"");
7 -  DomdocumentType dt=
                    pkg.getDomdocumentType().createDomdocumentType(
                              "\"C:\\Metadata\\dw.dtd\"");
8 -  Domelement elroot =
              pkg.getDomelement().createDomelement("DATAWAREHOUSES");
9 -  pkg.getDomcontains().add(pi,docroot);
10  -pkg.getDomcontains().add(dt,docroot);
11  -pkg.getDomcontains().add(elroot,docroot);
12  -pkg.getDomcontains().add(
       pkg.getDomcomment().createDomcomment("Metadados sobre os
                                    esquemas dos datawarehouses",""),
                                                docroot);
13  -repository.endTrans();
}
```

Para criar qualquer objeto no metamodelo é necessário obter a referência à interface *proxy* do objeto a ser criado. Por exemplo, para criar um novo documento XML é necessário *DomdocumentClass*, que é obtido pelo método *getDomdocument()* do pacote *DommetamodelPackage*. A interface *DomdocumentClass* possui o método para criar um novo documento XML, instância de *Domdocument.* Da mesma maneira, para se criar um objeto, instância de qualquer classe do metamodelo, é necessário obter a referência à sua interface *proxy*. Em DOM, os métodos para criar os objetos fazem parte da interface *Document,* em MOF, os métodos para a criação de objetos estão nas classes *proxys* referentes aos objetos.
A execução do código da Listagem 6 cria um objeto instrução de processamento, um objeto *DocType* e um objeto elemento, que é a raiz do documento XML. Além disso, é necessário fazer as associações entre os objetos. Isto é feito através da associação *DOMContains* do metamodelo. A Figura 6 apresenta o documento XML criado pela execução do código Java da Listagem 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DATAWAREHOUSES SYSTEM "C:\metadata\dw.dtd">
<!--Metadados sobre os esquemas dos datawarehouses-->
<DATAWAREHOUSES></DATAWAREHOUSES>
```

Figura 6 - Documento XML criado através da execução do código da Listagem 6

7. Conclusões e Trabalhos Futuros

Este artigo apresentou o projeto e a implementação de um metamodelo MOF que visa representar objetos DOM em repositórios MOF. O padrão DOM oferece uma maneira das aplicações manipularem documentos XML em forma de objetos, dispostos em uma hierarquia, também chamada de árvore DOM. As ferramentas que implementam o MOF possuem a capacidade de importar metadados no formato XMI. Com o metamodelo DOM, essas ferramentas passam a importar e exportar qualquer tipo de documento XML. O padrão XML é independente de plataforma, semi-estruturado, suportado por uma variedade de ferramentas e adequado para representar metadados das mais diversas fontes.

Foi gerado um documento XMI a partir do metamodelo proposto. Qualquer repositório MOF poderá importar este documento e passará a ter suporte ao gerenciamento de metadados descritos pelo metamodelo. Além do metamodelo, os metadados que são instâncias desses metamodelo também poderão ser exportados e importados por qualquer repositório MOF.

Apresentamos também um estudo de caso ilustrando como as interfaces do metamodelo poderão ser utilizadas por outras ferramentas para o gerenciamento de metadados descritos pelo padrão modelado.

Um trabalho que poderia ser desenvolvido a partir deste seria a implementação do módulo de importação e exportação usando a API SAX (Simple API for XML) [9]. Esta interface é orientada a eventos. O *parser* que implementa SAX lê um documento e diz à aplicação quais os símbolos que encontra, à medida que os encontra. Uma das vantagens do uso desta API seria o uso de menos recursos computacionais para processar os documentos XML, pois esta não armazena o documento na memória.

References

[1].AHMED, Kal; AYERS, Danny; et al. “Professional XML Metadata”. 2001. UK. Wrox Press Ltda.

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

[3].Distributed Systems Technology Centre - DSTC. “DMof - An OMG Meta Object Facility Implementation”. Disponível em: http://www.dstc.edu.au/Products/CORBA/MOF/. June, 2001.

[4].Document Object Model (DOM) Level 1 Specification Version 1.0 (1998) -http://www.w3.org/TR/REC-DOM-Level-1/

[5].Document Object Model (DOM) Level 2 Core Specification version 1.0 (2000) - http://www.w3.org/TR/DOM-Level-2-Core/

[6].Document Object Model (DOM) Level 3 Core Specification Version 1.0 (2002) - http://www.w3.org/TR/DOM-Level-3-Core/

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

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

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

[10].Object Management Group, “Meta Object Facility Specification, Version 1.3”. Disponível em: http://www.dstc.edu.au/Research/Projects/MOF/rtf/. Veja também em: http://www.omg.org/. September, 1999.

[11].Object Management Group. “Unified Modeling Language Specification, Version 1.4”. Disponível em: http://cgi.omg.org/docs/formal/01-09-67.pdf. September, 2001.

[12].Object Management Group. “XML Metadata Interchange Specification, Version 1.1”. Disponível em: http://www.omg.org/. June, 2000.

[13].OMG Architecture Board MDA Drafting Team. "Model-Driven Architecture: A Technical Perspective". Disponível em: ftp://ftp.omg.org/pub/docs/ab/01-02-01.pdf. February, 2001.

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

[15].SANTOS, Hélio; BARROS, Roberto; FONSECA, Decio. “A Proposal for Management of RDF and RDF Schema Metadata in MOF”. Proc. Int. Conf. on Ontologies, Databases and Applications of SEmantics (ODBASE’2003), Sicily, Italy, November 2003. Lecture Notes in Computer Science. Springer, 2003

[16].SANTOS, Hélio; BARROS, Roberto; FONSECA, Decio. “Uma Proposta para Gerenciamento de Metadados XML e DTD em MOF”. Anais 18° Simpósio Brasileiro de Banco de Dados, Manaus, Brasil, Outubro, 2003.

[17].Sun Microsystems. “Metadata Repository Home”. Disponível em: http://mdr.netbeans. org/. 2002.

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