

Usando MDA na Transformação de Modelos no domínio UML em Modelos no domínio de Banco de Dados Objeto-Relacional [♦]

**Marco Antonio Pereira, Antonio Francisco do Prado, Mauro Biajiz,
Valdirene Fontanette**

Universidade Federal de São Carlos - Departamento de Computação
São Carlos, Brazil, Cep.13565-905
pemarco@gmail.com, {prado, mauro, valdirene}@dc.ufscar.br

Abstract

This article presents the use of Model Driven Architecture (MDA) to represent the software specifications from its modelling to implementation. In a lower abstract level there is a logical model in an Object-Relational Data Base (ORDB) domain obtained from a static model in an Object-Oriented (OO) domain. Transformation rules do the mapping among the models of different domains. Codes in Structured Query Language (SQL) are obtained from models in the ORDB domain. A case study illustrates the instantiation of an OO Model, its transformation in an BDOR model and finally, the codes in SQL.

Keywords: MDA, UML, CWM, SQL.

Resumo

Este artigo apresenta a utilização da Model Driven Architecture (MDA) para representar as especificações do *software* desde a sua modelagem até a implementação. Em nível menos abstrato têm-se um modelo lógico no domínio de Banco de Dados Objeto-Relacional (BDOR) obtido a partir de um modelo estático no domínio Orientado a Objetos (OO). Regras de transformações fazem o mapeamento entre os modelos de diferentes domínios. Códigos em *Structured Query Language* (SQL) são obtidos a partir de modelos no domínio BDOR. Um estudo de caso ilustra a instanciação de um Modelo OO, sua transformação em um modelo BDOR e por fim, códigos em SQL.

Palavras chave: MDA, UML, CWM, SQL.

[♦] Projeto Financiado pelo Programa de Apoio à Inovação Tecnológica em Pequenas Empresas (FAPESP - PIPE)

1. Introdução

Muitas organizações que desenvolvem software utilizam diferentes linguagens, metodologias e ferramentas para aumentar a produtividade da equipe de desenvolvimento e diminuir os custos com o processo de desenvolvimento de *softwares*. Porém, a complexidade obtida pelo uso de diferentes soluções pode acarretar perda de qualidade e de reusabilidade. Visando aumentar a reusabilidade e a qualidade, modelos de *software* podem ser mantidos em diferentes domínios, a fim de prover o uso de modelos abstratos para se obter novos modelos de um determinado domínio.

Neste contexto encontra-se o *Model-Driven Development* (MDD) [22], que é um termo usado para expressar a idéia do desenvolvimento orientado a modelos. O foco do MDD são os modelos que representam a abstração do mundo real. Esses modelos não são apenas documentação auxiliar, mas são também artefatos de *softwares* que podem ser compilados diretamente em outros modelos e códigos em linguagens de programação [4]. O *Object Management Group* (OMG) propôs uma abordagem ao MDD chamada de *Model Driven Architecture* (MDA) [8]. A MDA utiliza linguagens padronizadas pela OMG para representar os artefatos de software nos diferentes níveis de abstração da arquitetura [13].

As linguagens utilizadas na MDA são denominadas de meta-metalinguagens, metalinguagens e linguagens de acordo com o nível de abstração em que são utilizadas. Essas linguagens podem ser integradas em uma ferramenta *Computer-Aided Software Engineering* (CASE) com o objetivo de operacionalizar o desenvolvimento de *softwares*. Neste artigo, particular ênfase é dada na integração das linguagens da MDA à ferramenta MVCASE e na definição de regras de transformação de modelos. Essas regras de transformação são definidas com objetivo de mapear os elementos de um modelo especificado no domínio Orientado a Objetos (OO) para elementos de um modelo especificado no domínio de Banco de Dados Objeto-Relacional (BDOR). Um transformador utiliza as regras de transformação para gerar, a partir de um modelo OO, um novo modelo BDOR de acordo com o paradigma da orientação a objetos do domínio de BDOR [14] e finalmente, códigos em *Structured Query Language* (SQL) [21] e [6] são obtidos a partir do modelo BDOR. A MDA é uma tecnologia que vem evoluindo com o passar do tempo, mas fatores fundamentais para a adoção prática da MDA, como integração das linguagens para operacionalização de seus conceitos e a definição das transformações entre os modelos ainda continuam em evidência entre os pesquisadores.

Este artigo está estruturado da seguinte maneira: na seção 2 são contextualizadas as linguagens *Meta Object Facility* (MOF) [16], *Unified Modeling Language* (UML) [2], *Common Warehouse Metamodel* (CWM) [19] e *XML Metadata Interchange* (XMI) [25] com os níveis de abstração da MDA; na seção 3 é apresentada a definição de regras de transformação de modelos; na seção 4 é apresentado um protótipo da MDA que implementa as idéias e conceitos abordados neste artigo e um pequeno estudo de caso, o qual utiliza regras de transformação para transformar instâncias de um modelo independente de plataforma, no caso um modelo especificado no domínio OO, em instâncias de um modelo dependente de plataforma, no caso um modelo especificado no domínio de BDOR; na seção 5 são descritos trabalhos correlatos e finalmente, na seção 6, têm-se as conclusões.

2. Model Driven Architecture

A *Model Driven Architecture* (MDA) fundamenta-se na idéia da separação entre as especificações de um sistema e os detalhes de sua implementação [13]. As especificações do *software* são definidas em altos níveis de abstração, as quais podem originar novas: i) especificações em níveis menos abstratos, como por exemplo, a obtenção de especificações em CORBA e Java a partir de um modelo em *Unified Modeling Language* (UML) [2] e; ii) especificações em diferentes domínios, como por exemplo, a obtenção de modelos no domínio de Banco de Dados a partir de um modelo no domínio da Orientação a Objetos especificado em UML.

O OMG padronizou linguagens para facilitar a integração entre os modelos da MDA. Essas linguagens são classificadas em um determinado nível de abstração na arquitetura. No nível mais abstrato, encontra-se a meta-metalinguagem, que se auto descreve e descreve as metalinguagens, as quais por sua vez, descrevem as linguagens.

A Figura 1 ilustra os diferentes níveis de abstrações (M_0 , M_1 , M_2 e M_3) da MDA. As meta-metalinguagem, metalinguagens e linguagens da arquitetura são descritas pelos seus correspondentes meta-metamodelo, metamodelos e modelos.

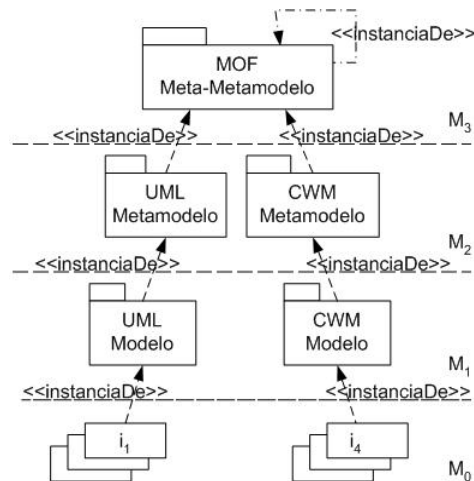


Figura 1 – Níveis de abstrações da MDA [16]

No nível M_3 , o mais abstrato, encontra-se o meta-metamodelo *Meta Object Facility* (MOF) [16], que é uma instância do seu próprio meta-metamodelo. O MOF descreve uma meta-metalinguagem abstrata utilizada para descrever outras metalinguagens para diferentes domínios [12] e também descreve um repositório de metadados para suportar metamodelos baseados no próprio MOF.

Os metamodelos no nível M_2 , instâncias do MOF, são criados de acordo com os requisitos de um particular domínio do problema, como por exemplo, para o domínio da orientação a objetos, o *UML Metamodelo* descreve uma metalinguagem para especificar modelos nesse domínio e para o domínio *Data Warehouse* (DW) e *Business Intelligence* (BI), o *Common Warehouse Metamodel* (CWM) [19] *Metamodelo* descreve uma metalinguagem para especificar modelos nesse domínio.

Os modelos no nível M_1 , o *UML Modelo* e o *CWM Modelo*, são instâncias de seus correspondentes metamodelos do nível M_2 . Finalmente no nível menos abstrato, o M_0 , têm-se as instâncias executáveis (i_1, i_2, \dots, i_n) dos modelos, que são descritas de acordo com as correspondentes linguagens de seus modelos do nível M_1 .

Os meta-metamodelo, metamodelos e modelos da MDA podem ser descritos em *XML Metadata Interchange* (XMI) [25]. XMI é uma linguagem da OMG que define um conjunto de regras para mapear meta-metamodelo, metamodelos e modelos para documentos *eXtensible Markup Language* (XML) [26]. Os documentos XML têm o objetivo de prover, de maneira simples e independente de plataforma, a interoperabilidade através do uso de cadeias de textos dotadas de descrições. XMI utiliza a *Document Type Definition* (DTD) para validar os documentos XML de acordo com os seus respectivos metamodelos. A DTD define a estrutura dos elementos que podem ser descritos nos documentos XML.

3. Transformação de Modelos

A transformação de modelos pode ocorrer entre mesmos níveis e entre diferentes níveis de abstração e também pode ocorrer entre mesmos domínios e diferentes domínios. A transformação é a geração automática de um modelo *destino* a partir de um modelo *origem*. A transformação é definida por um conjunto de regras que juntas descrevem como um modelo na linguagem *origem* pode ser transformado em um ou mais modelos na linguagem *destino* [11].

A Figura 2 ilustra a idéia da transformação de modelos. Um modelo independente de plataforma, o *Platform Independent Model* (PIM), pode ser transformado em novos modelos PIM e novos *Platform Specific Model* (PSM). Os novos PIM podem possuir algumas características específicas, porém essas não o classificam como um modelo dependente de plataforma, já os novos PSM possuem características específicas de uma determinada plataforma. Um PSM pode ser transformado em novos PSM e novos PIM. Os novos PSM são refinamentos do PSM original, já os novos PIM são modelos que substituem as características de uma determinada plataforma para características de uma plataforma independente.

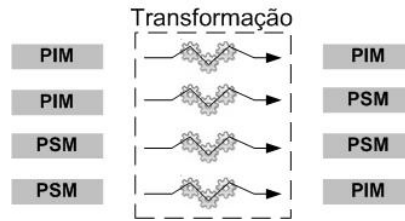


Figura 2 – Transformação de Modelos

As transformações de modelos podem ser chamadas de transformação Modelo-Modelo e transformação Modelo-Texto. Na categoria Modelo-Modelo, a transformação pode ser distinguida entre as seguintes abordagens: manipulação direta; orientada a estrutura; operacional; baseada em *template*; relacional; baseada em grafos e híbrida. Na categoria Modelo-Texto, as transformações podem ser distinguidas entre as abordagens baseadas no padrão *Visitor* e baseadas em *templates* [5]. As transformações Modelo-Texto são aplicadas para gerar códigos em uma determinada linguagem a partir de um PSM [13].

A Figura 3 ilustra os conceitos básicos da transformação Modelo-Modelo. Para que ocorra a transformação entre modelos é preciso definir as regras de transformação do Transformador. Essas regras são baseadas no conhecimento das estruturas dos elementos dos metamodelos *Origem* e *Destino*. O Transformador recebe como entrada o *Origem*' Modelo e o transforma no *Destino*' Modelo.



Figura 3 – Definição de Regras de Transformação [5]

De acordo com os domínios dos metamodelos *Origem* e *Destino*, o Transformador pode viabilizar: i) novos modelos no mesmo domínio, a fim de se obter um grau maior de especificidade em relação ao modelo original, essa transformação é chamada de *endogenous* ou *rephrasings* e; ii) novos modelos em domínios diferentes, a fim de se obter reuso de modelos para criação de novos modelos em diferentes domínios, essa transformação é chamada de *exogenous* ou *translations* [15]. Uma aplicação pode ser gerada automaticamente de uma especificação escrita em uma linguagem textual ou gráfica de um determinado domínio de problemas [3].

Na próxima seção são apresentados detalhes da implementação do protótipo da MDA utilizando os conceitos abordados.

4. Protótipo da MDA

Baseado nos estudos e idéias apresentadas, foi implementada um protótipo da MDA na ferramenta *Multiple View CASE* (MVCASE) [18]. A MVCASE é uma ferramenta que suporta a modelagem de sistemas de *software* com a notação UML [2]. O sistema modelado pode ser especificado segundo quatro visões: Casos de Uso; Lógica; Componentes; Visão “*Deployment*” ou Implantação.

A Figura 4 mostra para os diferentes níveis de abstração da MDA, os componentes MDRCComp, UMLComp, CWMComp e UML2CWM, os quais foram implementados em Java e são responsáveis pela operacionalização da MDA na MVCASE.

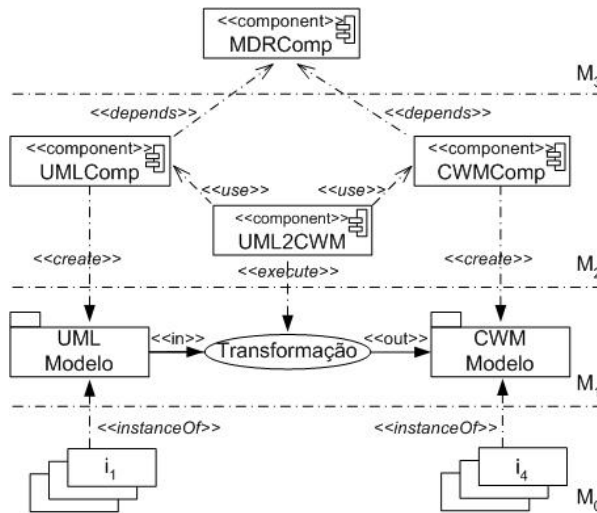


Figura 4 - Arquitetura do protótipo

4.1. Integrando as Linguagens nos Níveis M₃ e M₂

O componente MDRComp é uma implementação do MOF (M₃), do *Java Metadata Interface* (JMI) e do XMI fornecida gratuitamente pela *NetBeans Community* e chamada de *Metadata Repository* (MDR) [12]. O JMI implementado no MDR define o mapeamento de metamodelos baseados no MOF para a linguagem Java e define também um conjunto de interfaces reflexivas que podem ser usadas para acessar instâncias dos metamodelos. O XMI implementado no MDR possibilita a descrição dos metadados dos metamodelos em documentos XML.

Os componentes UMLComp e CWMComp são metamodelos (M₂) correspondentes aos domínios OO e BDOR. Esses metamodelos são instâncias do MOF e podem armazenar seus metadados no MDRComp.

Outro componente da arquitetura do protótipo é o UML2CWM, o qual é responsável pela transformação das instâncias do UMLComp em instâncias do CWMComp. As regras de transformação são especificadas no UML2CWM em linguagem Java e estabelecem o mapeamento entre os elementos do UMLComp e os elementos do CWMComp. A transformação é sempre executada de forma determinística e guiada por informações contidas nas instâncias do UMLComp [13], e dessa forma, o UML2CWM recupera as instâncias do UMLComp em uma determinada ordem pré-estabelecida e as mapeia para novas instâncias do CWMComp. Uma instância do UMLComp entra no UML2CWM, e uma instância do CWMComp sai, pois as regras de transformação definem apenas o mapeamento unidirecional da linguagem UMLComp para a linguagem CWMComp.

A Figura 5 ilustra por meio do *Modelo de Casos de Uso Transformar Modelos UML2CWM* o comportamento do componente UML2CWM. Têm-se um *Modelo UML* de entrada que é submetido ao caso de uso *Transformar Modelo UMLComp para Modelo CWMComp*. Esse caso de uso organiza o projeto de transformação e incorpora em uma dada seqüência o comportamento dos seguintes casos de uso: *Transformar Tipo de Dado*, *Transformar Classe*, *Transformar Associação* e *Transformar Generalização*. O caso de uso *Transformar UMLComp para CWMComp*, retorna para o Engenheiro uma instância do CWMComp, o *CWM Modelo*.

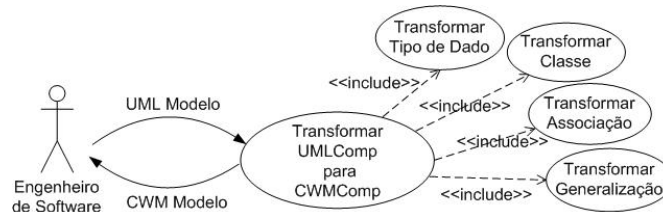


Figura 5 - Modelo de Casos de Uso Transformar Modelos UML2CWM

O modelo de casos de uso da Figura 5 descreve o comportamento do sistema para transformar as instâncias dos tipos *Data Type*, *Class*, *Association* e *Generalization* do UMLComp em instâncias dos tipos *SqlSimpleType*, *SqlDistinctType*, *SqlStructuredType*, *Table*, *Dependency*, *Association* e *Generalization* do CWMComp.

A Figura 6 mostra o *Modelo de Seqüência Transformar Tipo de Dado* que detalha o caso de uso *Transformar Tipo de Dado* do modelo de casos de uso da Figura 5. O UML2CWM recupera as instâncias do tipo *DataType* de UMLComp e as transforma em instâncias do tipo *SqlSimpleType* ou *SqlDistinctType* ou *SqlStructuredType* de CWMComp, desde que seja cumprida a restrição A ou B ou C *Constraints* detalhadas a seguir:

- *A Constraint: Se (dataType.getAtributos().size()==0):*

Uma instância do tipo *SqlSimpleType* do CWMComp é criada para cada instância do tipo *DataType* recuperada do UMLComp que não contenha nenhum atributo.

- *B Constraint: Senão, Se ((dataType.getAtributo().size()==1) e (dataType.getAtributo().equals(SqlSimpleType))):*

Uma instância do tipo *SqlDistinctType* do CWMComp é criada para cada instância do tipo *DataType* recuperada do UMLComp que contenha apenas um atributo e este seja do tipo *SqlSimpleType*.

- *C Constraint: Senão, Se (dataType.getAtributos().size(>1) ou Se ((dataType.getAtributos().size()==1) e ((dataType.getAtributos().equals(SqlDistinctType)) ou (dataType.getAtributos().equals(SqlStructuredType))))):*

Uma instância do tipo *SqlStructuredType* do CWMComp é criada para cada instância do tipo *DataType* recuperada do UMLComp que contenha mais de um atributo ou que contenha um único atributo do tipo *SqlDistinctType* ou *SqlStructureType*.

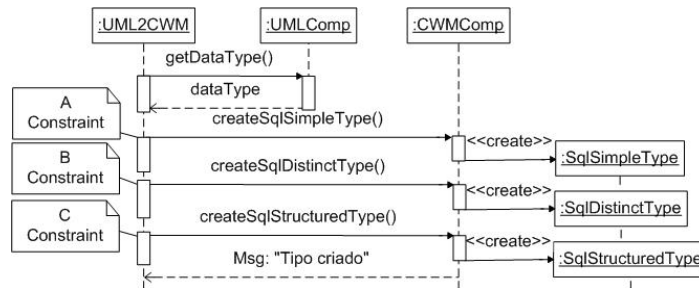


Figura 6 - Modelo de Seqüência Transformar Tipo de Dado

A Figura 7 mostra o *Modelo de Seqüência Transformar Classe* que detalha o caso de uso *Transformar Classe* do modelo de casos de uso da Figura 5. O UML2CWM recupera as instâncias *Class* do UMLComp e as transforma em instâncias *SqlStructuredType*, *Table* e *Dependency* do CWMComp.

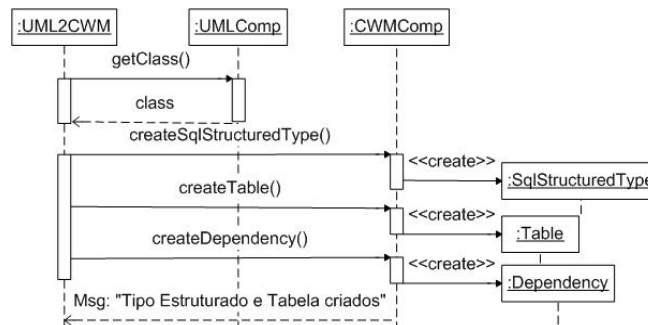


Figura 7 - Modelo de Seqüência Transformar Classe

A Figura 8 mostra o *Modelo de Seqüência Transformar Associação* que detalha o caso de uso *Transformar Associação* do modelo de casos de uso da Figura 5. O UML2CWM recupera as instâncias *Association* do UMLComp e as transforma em instâncias *Association*, *SqlStructuredType*, *Table* e *Dependency* do CWMComp, desde que seja cumprida a restrição A ou B e C *Constraints*:

- *A Constraint: Se ((association.cardinality().equals(1x1)) ou*

$(association.cardinality().equals(1xN))$:

Uma instância do tipo *Association* do CWMComp é criada para cada instância do tipo *Association* recuperada do UMLComp que contenha o valor da cardinalidade igual a *1x1* ou *1xN*

- *B Constraint: Senão:*

Caso não seja satisfeita a *A Constraint*, as instâncias do tipo *SqlStructuredType*, *Table*, *Dependency* e *Association* do CWMComp são criadas para cada instâncias recuperadas do UMLComp.

- *C Constraint: Se (association.getAggregation().equals(composite)):*

Atualizações nas instâncias *Association* e *Table* do CWMComp são executadas para cada instância do tipo *Association* recuperada do UMLComp que contenha o valor da agregação igual a *composite*.

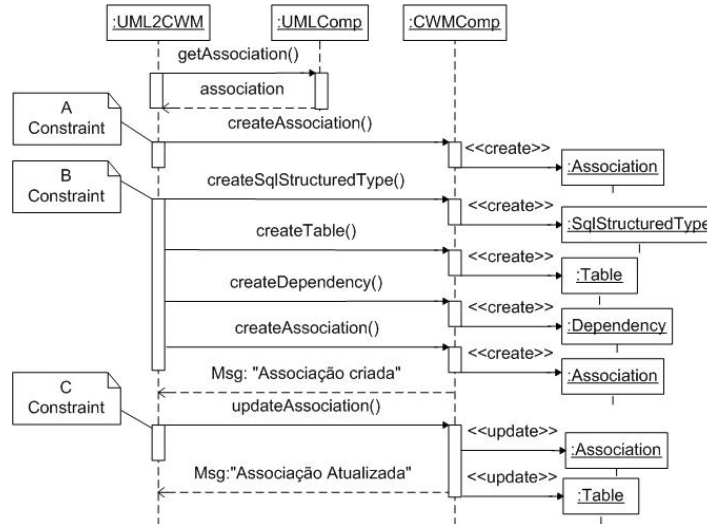


Figura 8 - Modelo de Seqüência Transformar Associação

A Figura 9 mostra o *Modelo de Seqüência Transformar Generalização* que detalha o caso de uso *Transformar Generalização* do modelo de casos de uso da Figura 5. O UML2CWM recupera as instâncias *Generalization* do UMLComp e as transforma em instâncias *Generalization* do CWMComp.

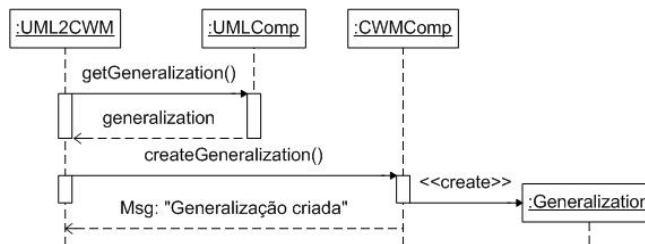


Figura 9 - Modelo de Seqüência: Transformar Generalização

A definição das regras de transformação Modelo-Modelo dependem do conhecimento prévio do domínio OO e do domínio de BDOR e também das estruturas dos elementos dos UMLComp e CWMComp.

4.2. Integrando as Linguagens no Nível M₁

Uma aplicação simples do domínio de vendas é utilizada para ilustrar a integração das linguagens no nível M₁, isto é, um modelo no domínio OO e sua transformação em um novo modelo no domínio de BDOR. Esse exemplo de aplicação é chamado de *Sale* e seu modelo no domínio OO é chamado *UML Modelo*, o qual foi especificado na MVCASE usando um editor gráfico orientado pela sintaxe e semântica da UML.

A Figura 10 ilustra o *UML Modelo* da *Sale*, esse modelo mostra as classes *Pessoa*, *Consumidor*, *Pedido*, *ItemPedido* e *Produto*. A semântica do modelo mostra que uma *Pessoa* pode ser um *Consumidor*, o qual pode estar associado a um ou mais *Pedido*. Cada *Pedido* é composto por um ou mais *ItemPedido* e pode estar associado a apenas um *Consumidor*. Cada *ItemPedido* está associado a um único *Produto* e faz parte de apenas um *Pedido*.

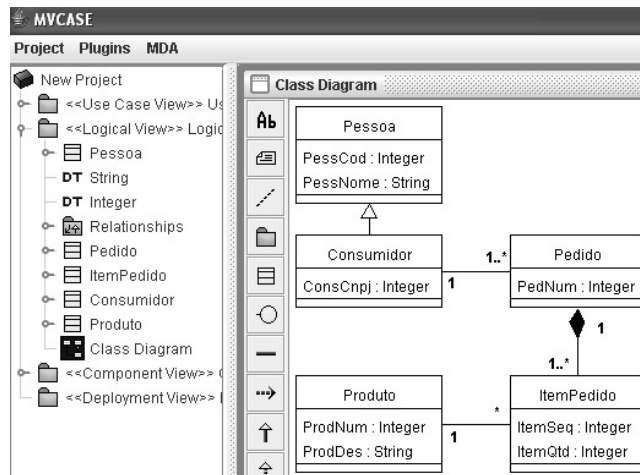


Figura 10 - UML Modelo da Sale

A Figura 11 ilustra o modelo no domínio de BDOR obtido a partir do *UML Modelo* da Figura 10 por meio das transformações executadas pelo UML2CWM. Esse modelo é chamado de *CWM Modelo*, o qual é está de acordo com as características de OO contidas na SQL. Os estereótipos utilizados no *CWM Modelo* ajudam a caracterizar a semântica de um modelo no domínio de BDOR [27]. O estereótipo *ObjectType* indica que a classe é um objeto, o *ObjectTable* indica que a classe é uma tabela e o *NestedTable* indica que a classe é uma tabela aninhada.

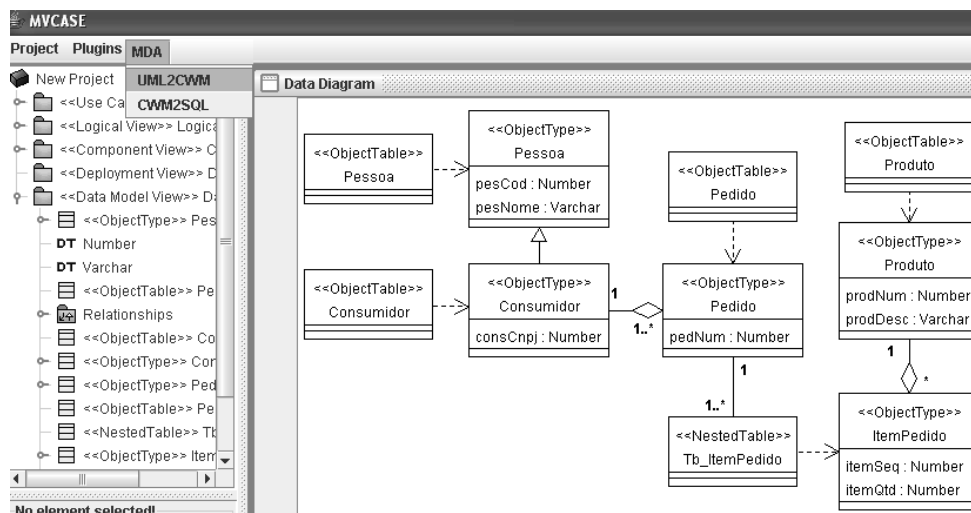


Figura 11 - CWM Modelo da Sale

Os relacionamentos de dependências entre classes que possuem o estereótipos *ObjectTable* e *ObjectType* indicam que uma tabela é criada a partir da estrutura de um objeto. Os relacionamentos de dependências entre classes que possuem os estereótipos *NestedTable* e *ObjectType* indicam que uma tabela aninhada é criada também a partir da estrutura de um objeto.

Os relacionamentos entre classes que possuem o estereótipo *ObjectType* indicam que a classe a qual a agregação esta ligada, possui um atributo que faz referência a outra classe do relacionamento em questão. Os relacionamentos entre classes que possuem os estereótipos *ObjectTable* e *NestedTable* indicam que um atributo em *ObjectTable* faz referência a uma tabela aninhada. Os relacionamentos de heranças entre classes que possuem o estereótipo *ObjectType* indicam que um objeto especializa o outro.

Para obter o *CWM Modelo* a partir do *UML Modelo* da *Sale*, o UML2CWM recupera instâncias de determinados tipos no *UML Modelo* e as transforma em instâncias de determinados tipos no *CWM Modelo*:

- i) Instâncias *DataType* transformadas em *SqlSimpleType*:

A instâncias Integer e String do tipo *DataType* no *UML Modelo* são transformadas nas correspondentes instâncias Number e Varchar do tipo *SqlSimpleType* no *CWM Modelo*.

ii) Instâncias *Class* transformadas em *Table*, *SqlStructuredType* e *Dependency*:

As instâncias *Pessoa*, *Consumidor*, *Pedido*, *ItemPedido* e *Produto* do tipo *Class* no *UML Modelo* são transformadas em instâncias no *CWM Modelo* do tipo *SqlStructuredType* com estereótipo *ObjectType*, *Table* com estereótipo *ObjectTable* e *Dependency*.

iii) Instâncias *Association* transformadas em *Association*:

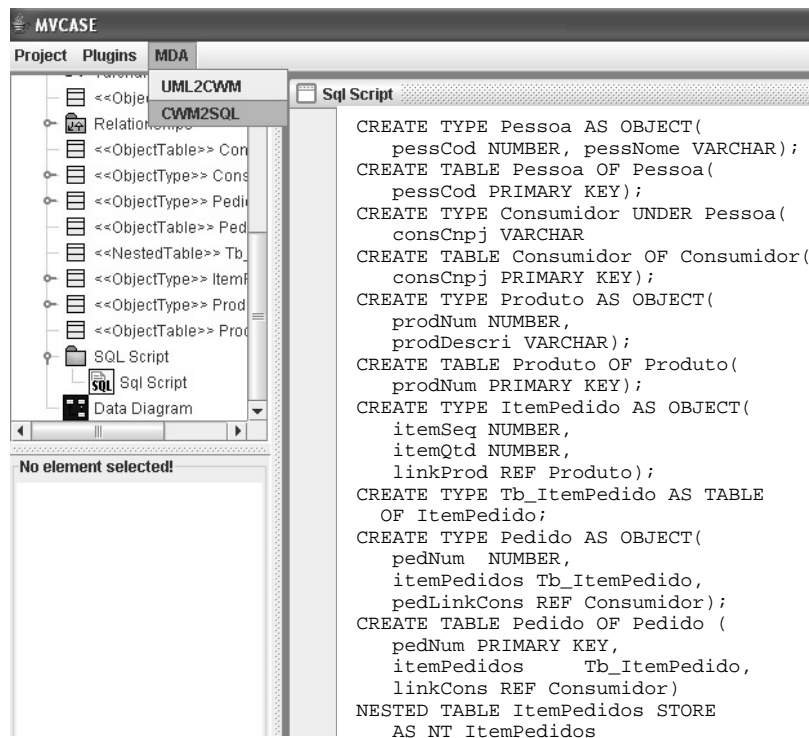
As instâncias do tipo *Association* no *UML Modelo* que possuem cardinalidade *1xN* são transformadas em instâncias do tipo *Association* no *CWM Modelo* com o valor de agregação para o lado com cardinalidade *N*. Para o caso da associação que possui em um dos lados a composição (*composite*) no *UML Modelo*, há uma restrição (Figura 8, *C Constraint*) na operação Transformar Associação que executa a atualização das instâncias *Association* e *Table* de *CWMComp*.

iv) Instâncias *Generalization* Transformadas em *Generalization*:

As instâncias do tipo *Generalization* no *UML Modelo* são transformadas em instâncias do tipo *Generalization* no *CWM Modelo*.

Num nível menos abstrato, podem-se obter códigos em SQL a partir do *CWM Modelo*, para isso é usado um componente gerador de códigos que recupera informações das instâncias *CWM Modelo* e as transforma em códigos em SQL.

A Figura 12 mostra os códigos em SQL obtidos a partir do *CWM Modelo* da Figura 11 com o auxílio do gerador de códigos. As instâncias do tipo *SqlStructuredType* que possuem estereótipos com o valor *ObjectType* e que não fazem parte como “filha” em um dos lados do relacionamento das instâncias *Generalization* no *CWM Modelos*, são mapeadas para o código “*Create Type instância.nome() As Object*”, essa sintaxe SQL cria um tipo de dado do tipo objeto no banco de dados, já as instâncias que fazem parte como “filhas” em um dos lados do relacionamento de uma instância *Generalization* no *CWM Modelo*, são mapeadas para o código “*Create Type instância.filha() Under instância.pai()*”. Por exemplo, a instância *Pessoa* origina o seguinte código “*Create Type Pessoa As Object*”. É o caso também das instâncias *Produto*, *ItemPedido* e *Pedido*. A instância *Consumidor* por fazer parte como “filha” em um relacionamento de uma instância *Generalization*, é mapeada para o seguinte código “*Create Type Consumidor UNDER Pessoa*”, indicando que o *Consumidor* é uma especialização de *Pessoa*.



```
CREATE TYPE Pessoa AS OBJECT(
    pessCod NUMBER, pessNome VARCHAR);
CREATE TABLE Pessoa OF Pessoa(
    pessCod PRIMARY KEY);
CREATE TYPE Consumidor UNDER Pessoa(
    consCnpj VARCHAR);
CREATE TABLE Consumidor OF Consumidor(
    consCnpj PRIMARY KEY);
CREATE TYPE Produto AS OBJECT(
    prodNum NUMBER,
    prodDescri VARCHAR);
CREATE TABLE Produto OF Produto(
    prodNum PRIMARY KEY);
CREATE TYPE ItemPedido AS OBJECT(
    itemSeq NUMBER,
    itemQtd NUMBER,
    linkProd REF Produto);
CREATE TYPE Tb_ItemPedido AS TABLE
OF ItemPedido;
CREATE TYPE Pedido AS OBJECT(
    pedNum NUMBER,
    itemPedidos Tb_ItemPedido,
    pedLinkCons REF Consumidor);
CREATE TABLE Pedido OF Pedido (
    pedNum PRIMARY KEY,
    itemPedidos Tb_ItemPedido,
    linkCons REF Consumidor)
NESTED TABLE ItemPedidos STORE
AS NT_ItemPedidos
```

Figura 12 – Código em SQL do CWM Modelo

As instâncias do tipo *Table* que possuem estereótipos com o valor *ObjectTable* e que possuem uma dependência para uma outra instância do tipo *SqlStructuredType* cujo valor de estereótipo seja *ObjectType*, são

mapeadas para o seguinte código “*Create Table instância.nome() Of dependênciaInstância.nome()*”, essa sintaxe SQL cria uma tabela com a mesma estrutura de um objeto no banco de dados. Por exemplo, a instância *Consumidor* origina o código “*Create Table Consumidor Of Consumidor*”, como é também o caso das instâncias *Pessoa*, *Pedido* e *Produto*.

As instâncias do tipo *Table* que possuem estereótipos com o valor *NestedTable* e que possuem uma dependência para uma outra instância do tipo *ObjectType* cujo estereótipo seja *ObjectType*, são mapeadas para o seguinte código “*Create Type instância.nome() As Table Of dependênciaInstância.nome()*”, essa sintaxe SQL cria uma tabela aninhada que pode ser referenciada por um atributo em uma outra tabela no banco de dados. Por exemplo, a instância *Tt_ItemPedido* origina o código “*Create Type Tb_ItemPedido As Table Of ItemPedido*”.

As instâncias do tipo *Association* que possuem em um dos lados o valor *aggregate* são mapeadas para um atributo que faz referência ao outro lado da associação de acordo com o seguinte código “*'link'+instanciaLadoSemAggregate.nome() Ref instanciaLadoSemAggregate.nome()*”, essa sintaxe SQL cria um atributo do tipo referência de outros objetos no banco de dados. Por exemplo, na instância do tipo *Association* que possui em um lado a instância *Produto* e no outro lado a instância *ItemPedido* com o valor *aggregate*, é criado um atributo em *ItemPedido* que referência a instância *Produto*, de acordo com o código “*linkProduto REF Produto*”.

As instâncias do tipo *Association* que relacionam duas outras instâncias dos tipos *SqlStructuredType* e *Table*, cujos estereótipos são *ObjectType* e *NestedTable*, são mapeadas para um atributo na instância de estereótipo *ObjectType* que faz referência a instância do lado de estereótipo *NestedTable* de acordo com o código “*instânciaTb.nomeSemTb()+ 's' instanciaTb.nome() Nested Table instânciaTb.nomeSemTb()+s Store As Nt_+instânciaTb.nomeSemTb()*”, essa sintaxe SQL cria um atributo do tipo tabela aninhada no banco de dados. Por exemplo, na instância *Association* que relaciona as instâncias *Pedido* e *Tb_ItemPedido*, um atributo que referencia *Tb_ItemPedido* é criado no lado *Pedido* com o seguinte código SQL “*ItemPedidos Tb_ItemPedido Nested Table ItemPedidos Store As Nt_ItemPedido*”.

O protótipo vem sendo testado desde o 2º. Semestre de 2006 por alunos de uma turma da disciplina Laboratório de Banco de Dados ministrada para o curso de Bacharelado em Ciência da Computação do Departamento de Computação da Universidade Federal de São Carlos. Dentre os vários modelos testados por essa turma, mostrou-se nesta seção um pequeno exemplo de uso com o objetivo de demonstrar a viabilidade operacional do protótipo.

Os testes estão indicando também novos rumos na implementação do protótipo a fim de flexibilizar a modelagem semântica de acordo com o gerenciador de banco de dados, através do uso de um editor gráfico orientado pela sintaxe e semântica e um transformador de acordo com as necessidades de mapeamento dos elementos dos metamodelos.

5. Trabalhos Correlatos

Diversas abordagens têm sido propostas para a transformação de modelos, dentre elas encontra-se: a *Query/Views/Transformations* (QVT) [20], que visa permitir a definição das regras de transformações unidirecionais por meio da abordagem declarativa e imperativa; a *Visual Automated Model* (VIATRA) [24], que visa criar os relacionamentos entre os elementos dos *Modelos Origem e Destino* por meio de grafos que os relacionam, aplicando regras de transformação de forma não determinística e a *MT Model Transformation Language* [23], que é uma derivação da QVT e permite transformações unidirecionais por meio da abordagem declarativa permitindo que códigos sejam embutidos as regras de transformação. Além dessas abordagens, foram encontradas diversas implementações em ferramentas existentes que se aproximam da abordagem deste trabalho, dentre elas: o projeto *Generative Modeling Technologies* (GMT) [9]; o *framework* AndroMDA [1] e; a ferramenta Rational Rose Data Modeler da IBM [10].

O GMT é um dos projetos da *Eclipse Community* que produz um conjunto de ferramentas para a *Model-Driven Engineering* (MDE). Uma dessas ferramentas faz uso da *Atlas Transformation Language* (ATL). A ATL é uma linguagem baseada no MOF e baseada numa sintaxe concreta para transformação de modelos, que ainda possibilita a combinação de linguagem declarativa e imperativa. As transformações dos modelos, que também devem estar de acordo com seus respectivos metamodelos e esses de acordo com o MOF, são descritas em ATL através de um conjunto de regras de transformações [17]. As regras de transformações são processadas pelo *ATL Development Tooling* (ADT) que funciona com o suporte da *Integrated Development Environment* (IDE) Eclipse. A ATL tem a limitação de não suportar a geração de código. O protótipo desse artigo faz a transformação de um *UML Modelo* para um *CWM Modelo* através de um componente que transforma de forma

simples as instâncias do repositório utilizando somente a máquina virtual Java, sem a necessidade da utilização de um IDE para suportar o protótipo.

O AndromDA é um *framework* extensível para a MDA que recebe como entrada um modelo UML em XMI e gera uma saída utilizando *templates* configuráveis específicos para plataformas pré-determinadas. Além dos *templates* prontos, como por exemplo, Hibernate, Struts, JSF e outros, é possível criar novos *templates* escritos em *Velocity Template Language* (VTL) de acordo com cada necessidade. Para a AndromDA suportar as características do protótipo implementado na MVCASE, seria necessário configurar um *template* para gerar um XMI de acordo com o *CWM Metamodelo*, além de configurar um outro *template* para gerar códigos SQL à partir de um *UML Modelo*, já que o AndromDA recebe como entrada somente *UML Modelo* em XMI. No caso do protótipo proposto neste artigo, não é necessário nenhuma configuração extra para gerar o XMI dos *CWM Modelos*, pois o MDR implementa o XMI, assim a geração de XMI é automática e de acordo com o *CWM Metamodelo*.

A ferramenta IBM Rational Rose Data Modeler permite a transformação entre modelos de objetos, modelos de dados e geração de código SQL. A IBM Rational Rose usa o MOF e o XMI como parte do *framework* de integração de ferramentas dirigidas a modelos, além do suporte ao *Eclipse Modeling Framework* (EMF). O EMF pode ser entendido como uma eficiente implementação Java de um subconjunto do MOF [7] que gerencia instâncias de metamodelos baseados no MOF. Essa ferramenta dá suporte a MDA com enfoque no desenvolvimento de *software* para Banco de Dados, mas não usa o *CWM Metamodelo* como metamodelo dos modelos de dados. O protótipo desse artigo, implementado na MVCASE, utiliza o *CWM Metamodelo* que possibilita a integração entre metadados de distintas ferramentas de desenvolvimento de *software* que utilizam o mesmo *CWM Metamodelo* definido pela OMG.

6. Conclusão

A principal contribuição deste trabalho compreende a integração das diferentes linguagens MOF, UML, CWM e XMI da OMG com a transformação de modelos do domínio OO para modelos do domínio de BDOR. A integração de um conjunto de linguagens padrões facilita a troca de informações entre ferramentas distintas que utilizam as mesmas linguagens ou que utilizam uma porção delas. O reuso de modelos de diferentes plataformas proporciona maior rapidez e qualidade no desenvolvimento de *software*.

A operacionalização da MDA na MVCASE teve sua viabilidade demonstrada através de exemplos executados em laboratório. Neste trabalho a operacionalização foi ilustrada através de um pequeno exemplo, o qual se partiu de um modelo estático em UML e efetuando transformações, obteve-se um modelo lógico de dados, e por fim seu mapeamento para códigos em SQL pode ser viabilizado com um gerador de códigos.

Como descrito na seção 3, os resultados da implementação do protótipo estão direcionando os próximos passos. Assim, a continuidade do trabalho compreende: um estudo maior de cada abstração que envolve a manutenção e refino dos modelos do domínio de BDOR de forma gráfica; o desenvolvimento de um transformador genérico o suficiente para permitir que o Engenheiro de *Software* possa fazer alterações nas regras de transformação; e a criação de um metamodelo para o domínio do rastreamento das instâncias do modelo *origem* para o modelo *destino*.

Referências

- [1] Andromda. Disponível em <http://www.andromda.org>. (Dezembro, 2006).
- [2] Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language User Guide (Object Technology). Addison Wesley. 2th Rev Edition. 2005.
- [3] Czarnecki, K. "Overview of Generative Software Development". *Unconventional Programming Paradigms (UPP)*, Mont Saint-Michel, France. (2004), pp. 313–328.
- [4] Czarnecki, K., Antkiewicz, M. Kim, C.H.P., Lau, S., Pietroszek, K. "Model-Driven Software Product Lines". *ACM SIGPLAN - Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA) – Poster Session*, (2005), pp. 126-127.
- [5] Czarnecki, K., Helsen, S. "Feature-Based Survey of Model Transformation Approaches". *IBM Systems Journal*. (2006), pp. 621-645.
- [6] Eisenberg, A., Melton, J., Kulkarni, K., Michels, J.E., Zemke, F. "SQL:2003 has been published". *ACM SIGMOD Record*. (2004), pp. 119-126.

- [7] The Eclipse Modeling Framework Overview - Eclipse Modeling Framework (EMF). Disponível em <http://www.eclipse.org/emf/docs/> (Outubro, 2006).
- [8] Frankel, D.S. “Model Driven Architecture: Applying MDA to Enterprise Computing”. Willey Press. Hoboken, EUA. 2003.
- [9] Generative Modeling Technologies (GMT). Eclipse Modeling Project (EMP). Disponível em <http://www.eclipse.org/gmt>, (Novembro, 2006).
- [10] IBM Rational Data Modeler offers a sophisticated visual modeling environment Rational Rose Data Modeler. Disponível em <http://www-306.ibm.com/software/awdtools/developer/datamodeler>, (Dezembro, 2006).
- [11] Kleppe, A., Warmer, J., Bast, W. MDA Explained, The Model-Driven Architecture: Practice and Promise. Addison Wesley. 2003.
- [12] Matula, M. "NetBeans Metadata Repository". *NetBeans Community*. Disponível em <http://mdr.netbeans.org/docs.html>, (Setembro, 2006).
- [13] The Model-Driven Architecture - Guide Version 1.0.1, OMG Document: omg/2003-06-01, (2006).
- [14] Melton, J. Advanced SQL: 1999 - Understanding Object-Relational and Other Advanced Features. Morgan Kaufmann, 1th edition. 2002.
- [15] Mens, T., Gorp, Van Gorp, P. “A Taxonomy of Model Transformation and Its Application to Graph Transformation” *Proceedings of the International Workshop on Graph and Model Transformation*. Estônia, (2005), pp. 7-23.
- [16] Meta Object Facility 1.4 (MOF) Specification. Object Management Group (OMG), (2002).
- [17] Model-to-Model Transformation (M2M). Eclipse Modeling Project (EMP). Disponível em <http://www.eclipse.org/proposals/m2m>, (Novembro, 2006).
- [18] Paiva, D.M.B., Lucrédio, D., Fortes, R.P.M. “MVCASE - including design rationale to help modeling in research projects.” *XX - Simpósio Brasileiro de Engenharia de Software (XX SBES)*. Florianópolis - SC – Brasil, (2006).
- [19] Poole, J., Chang, D., Tolbert, D., Mellor, D. Common Warehouse Metamodel – Developer’s Guide. Willey Publishing, Inc. 2003.
- [20] Query/Views/Transformations (QVT). OMG Document: omg/2005-11-01, (2005).
- [21] International Organization for Standardization (ISO) & American National Standards Institute (ANSI) - ISO/IEC JTC1/SC32 - ANSI ISO/IEC 9075-1:2003. ISO International Standard. Database Language (2003) - SQL - Parte 1: Framework (SQL/ Framework).
- [22] Stahl, T., Völter, M. “Model-Driven Software Development – Technology, Engineering, Management”. John Willey and Sons Ltda., England. 2006.
- [23] Tratt, L. “The MT Model Transformation Language”, *Proceedings of ACM Special Interest Group on Applied Computing (SIGAC) - Session: Model transformation*, Dijon, France, (2006), pp. 1296-1303.
- [24] Varró, D., Varró G., Pataricza, A. “Generic and Meta-Transformation for Model Transformation Engineering”, *Proceedings of the 7th International Conference on Unified Modeling Language*, Lisboa, Portugal, (2004), pp. 290-304.
- [25] XML Metadata Interchange 1.3 (XMI) Specification. Object Management Group (OMG), (2003).
- [26] Extensible Markup Language (XML). W3C Architecture Domain. Disponível em <http://www.w3.org/XML>, (Dezembro, 2006).
- [27] Zendulka, J. “Object-Relational Modeling in UML”, *Encyclopedia of Database Technologies and Applications*, Idea Group Publishing, Hershey, US. pp. 421-426. 2005.