Transformando Modelos da MDA com o apoio de Componentes de Software♦

Marco Antonio Pereira[1], Antonio Francisco do Prado[1], Mauro Biajiz[1], Valdirene Fontanette[1], Daniel Lucrédio[2]

[1]Universidade Federal de São Carlos, Departamento de Computação
Rod. Washington Luís, Km 235 - Caixa Postal 676 - Cep.13565-905 - São Carlos, SP - Brasil

[2]Universidade de São Paulo, Instituto de Ciências Matemáticas e de Computação
Av. Trabalhador São-carlense, 400, Centro - Caixa-Postal: 668 - Cep. 13560-970 - São Carlos, SP - Brasil

pemarco@gmail.com, {prado, mauro, valdirene}@dc.ufscar.br, lucredio@icmc.usp.br

Abstract. *This article presents the use of software components to supply the Model Drivel Architecture (MDA) in the MVCASE tool. MDA represents the software specifications from its modeling to implementation. In a lower abstract level there are Object-Oriented models supported by the UML component. Using transformation components it is possible to obtain the Object-Relational DataBase models, and to get through, obtain also the codes in Structed Query Language (SQL). A example illustrates the use of these components to instantiate and transform different platforms models and finally obtain their codes in SQL.*

Resumo. *Este artigo apresenta o uso de componentes de software para operacionalizar a Model Driven Architecture (MDA) na ferramenta MVCASE. A MDA representa as especificações do software desde a sua modelagem até a implementação. Em nível menos abstrato, têm-se os modelos Orientados a Objetos suportados pelo componente UML. Utilizando-se de componentes de transformação é possível obter os modelos de Banco de Dados Objeto-Relacional e por fim, códigos em Structured Query Language (SQL). Um exemplo de uso ilustra o uso dos componentes para instanciar e transformar modelos de plataformas diferentes e por fim, obter seus códigos em SQL.*

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

1. Introdução

Muitas organizações que desenvolvem *softwares* 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 níveis de abstração, a fim de prover o uso de modelos abstratos para se obter novos modelos em plataformas específicas.

Neste contexto encontra-se o *Model-Driven Development* (MDD) [Stahl and Völter 2006], 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 [Czarnecki *et al*. 2005]. O *Object Management Group* (OMG) propôs uma abordagem ao MDD chamada de *Model Driven Architecture* (MDA) [Mda 2003]. A MDA utiliza linguagens padronizadas pela OMG para representar esses artefatos de *software*s.

As linguagens utilizadas na MDA são denominadas de meta-metalinguagens, metalinguagens e linguagens de acordo com o nível de abstração em que são utilizadas na arquitetura. Essas linguagens podem ser integradas em uma ferramenta *Computer-Aided Software Engineering* (CASE) com o objetivo de suportar modelos de diferentes plataformas. Neste artigo, particular ênfase é dada ao desenvolvimento e integração dessas linguagens por meio de componentes de *software* a fim de estender as funcionalidades da ferramenta MVCASE. Essa ferramenta já suporta o desenvolvimento de modelos Orientados a Objetos (OO) e para que a mesma também suporte o desenvolvimento de modelos de Bancos de Dados Objeto-Relacional (BDOR) [Zendulka 2005] e que por fim, possa gerar códigos em *Structured Query Language* (SQL) [Sql 1999] uma extensão baseada em componentes é proposta neste artigo.

Os componentes responsáveis pelas transformações são chamados de *Componentes Transformadores de Modelos* e *Componentes Geradores de Códigos*. Esses tem suas regras de transformações definidas e implementadas, e eles são integrados à MVCASE por meio de suas interfaces. O *Componente Transformador de Modelos* tem a função de mapear os elementos de um modelo OO para elementos de um modelo BDOR. Finalmente, por meio do *Componentes Gerador de Códigos*, os códigos em SQL são obtidos a partir de modelos BDOR. O uso de componentes para operacionalizar a MDA é um fator importante para prover reuso dos *Componentes Transformadores de Modelos* e dos *Geradores de Códigos* em outras ferramentas que também implementem os conceitos da MDA e as linguagens padronizadas pela OMG. A MDA é uma tecnologia que vem evoluindo com o passar do tempo e fatores fundamentais para a adoção prática da MDA, como operacionalização e definição de transformações de 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) [Mof 2002], *Unified Modeling Language* (UML) [Omg 2004], *Common Warehouse Metamodel* (CWM) [Omg 2003] e *XML*

Metadata Interchange (XMI) [Xmi 2003] com os níveis de abstração da MDA; na seção 3 são apresentadas as definições das regras de transformação; na seção 4 é apresentado um protótipo da MDA implementado na MVCASE e um exemplo de uso, no qual se utiliza de componentes para transformar modelos OO em modelos BDOR e por fim, se obter os códigos em SQL; 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 [Mda 2003]. 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 UML; e ii) especificações para diferentes plataformas, como por exemplo, a obtenção de modelos de Bancos de Dados a partir de modelos 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.

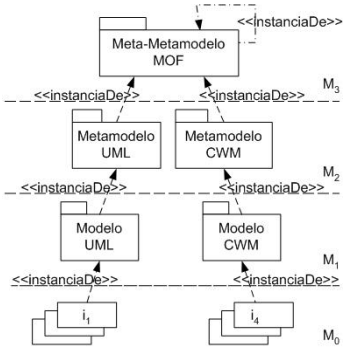

Figura 1 - Níveis de abstrações da MDA [MOF 2002]

No nível M_3, o mais abstrato, encontra-se o *Meta-Metamodelo MOF* (*Meta Object Facility* - MOF), que é uma instância do seu próprio meta-metamodelo. O *Meta-Metamodelo MOF* descreve uma meta-metalinguagem abstrata utilizada para descrever outras metalinguagens para diferentes domínios e também descreve um repositório de metadados para suportar metadados dos metamodelos baseados no próprio *Meta-Metamodelo MOF* [Mof 2002].

Os metamodelos no nível M_2, instâncias do *Meta-Metamodelo MOF*, são criados de acordo com os requisitos de um particular domínio do problema, como por exemplo, para o domínio OO, o *Metamodelo UML* descreve uma metalinguagem para especificar modelos nesse domínio [Omg 2004] e para o domínio *Data Warehouse* (DW) e *Business Intelligence* (BI), o *Metamodelo CWM (Common Warehouse Metamodel* -

CWM) descreve uma metalinguagem para especificar modelos para esse domínio [Poole *et al*. 2003].

Os modelos no nível M_1, o *Modelo UML* e o *Modelo CWM*, 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,..., 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) [Xmi 2003]. 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) [Xml 2006]. 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 [Kleppe *et al*. 2003].

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

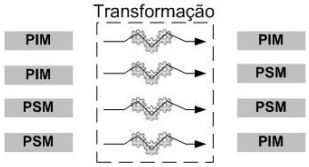

Figura 2 - Transformação de Modelos

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* [Gamma *et al.* 1995] e baseadas em *templates*

[Czarnecki e Helsen 2006]. As transformações *Modelo-Texto* são aplicadas para gerar códigos em uma determinada linguagem a partir de um PSM [Mda 2003].

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 [Czarnecki e Helsen 2006]

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* [Mens e Van Gorp. 2005]. Uma aplicação pode ser gerada automaticamente de uma especificação escrita em uma linguagem textual ou gráfica de um determinado domínio de problemas [Czarnecki 2004].

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

4. Protótipo da MDA na MVCASE

Baseado nos estudos e idéias apresentadas, foi implementada um protótipo da MDA na ferramenta *Multiple View* CASE (MVCASE) [Paiva *et al.* 2006]. A MVCASE é uma ferramenta que suporta a modelagem de sistemas de *software* com a notação UML [Booch *et al*. 2005]. O sistema modelado pode ser especificado segundo quatro visões: Casos de Uso; Lógica; Componentes; Visão "*Deployment*" ou Implantação.

O protótipo tem como objetivo operacionalizar os conceitos da MDA através do uso de componentes de *software*. Esses componentes transformam um sistema modelado na Visão Lógica da MVCASE (Modelo de Classes) em um sistema modelado para BDOR, e desse para códigos em SQL. Um componente é um conjunto de vários artefatos de *software* que podem ser independentemente desenvolvidos e ainda, pode ser composto para construir algo maior [D'Souza e Wills 1999].

A Figura 4 mostra os diferentes níveis de abstração da MDA e os componentes integrados à MVCASE. O componentes *MDRComp* e *UMLComp* já fazem parte da arquitetura da MVCASE. O *MDRComp* é responsável por armazenar os metadados dos componentes que dependem dele e suas instâncias correspondentes. O componente *UMLComp* prove o suporte aos modelos OO *(Modelos UML)*. O componente *CWMComp* é integrado à MVCASE com o objetivo de prover o suporte aos modelos BDOR *(Modelos CWM)*. O componente *UML2CWM* é um *Componente Transformador*

de Modelos que é integrado com o objetivo de prover o suporte a *Transformação Modelo-Modelo*, isto é, de *Modelos UML* para *Modelos CWM*. O *UML2CWM* é implementado de acordo com a abordagem de manipulação direta, na qual a lógica das regras de transformação e o escalonamento dessas regras são implementados diretamente no código do componente.

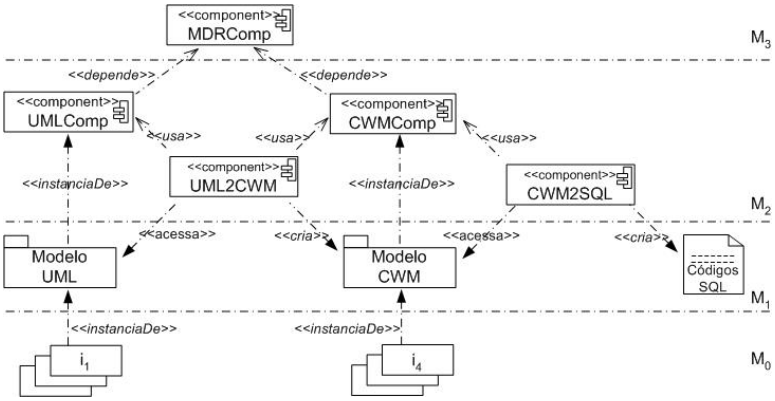

Figura 4 – Componentes que apóiam a MDA na MVCASE

O componente *CWM2SQL* é um *Componente Gerador de Códigos* que é integrado à MVCASE com o objetivo de prover o suporte a *Transformação Modelo-Texto*, isto é, de *Modelos CWM* para *Códigos SQL*. O CWM2SQL é implementado de acordo com a abordagem de manipulação direta, e suas regras de transformação consistem em navegar na estrutura interna dos elementos do *Modelo CWM*, coletando informações e as transformando em *Códigos SQL*.

4.1. Integrando Componentes

A Figura 5 mostra a integração desses componentes com a MVCASE, a qual é representada na figura por meio do pacote MVCASE. A MVCASE usa os componentes *MDRComp*, *CWMComp*, *CWM2SQL*, *CWM2UML*, *UMLComp* por meio de suas interfaces.

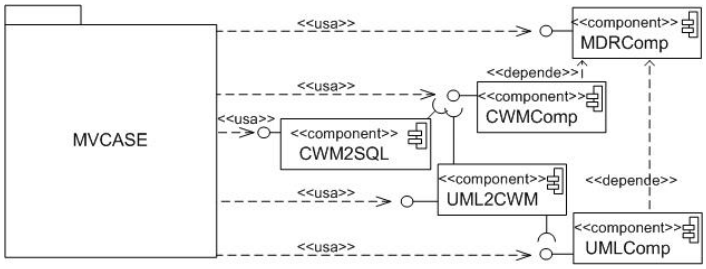

Figura 5 - Componentes integrados à MVCASE

O MDRComp é uma implementação do MOF, do JMI *(Java Metadata Interface)* e do XMI fornecida gratuitamente pela *NetBeans Community* e chamada de *Metadata Repository* (MDR) [Matula 2003]. O JMI implementado no *MDRComp* 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 *MDRComp* possibilita a descrição dos metadados dos metamodelos em documentos XML. O MOF

implementado possibilita que metadados de metamodelos baseados no MOF sejam armazenados no *MDRComp*.

O *UMLComp* é baseado no *Metamodelo UML* [Omg 2004] e o *CWMComp* é baseado no *Metamodelo CWM* [Omg 2003], ambos os metamodelos são baseados no MOF, assim, esses componentes dependem das implementações do *MDRComp* para armazenar seus metadados, instâncias e ler e escrever informações em XMI.

O *UML2CWM* acessa as interfaces do *UMLComp* e do *CWMComp*, isto possibilita a ele navegar por suas estruturas e coletar informações de suas instâncias, ou seja, dos *Modelos UML* e *CWM*, a fim de prover a transformação de forma determinística, unidirecional e guiada por informações contidas nos *Modelos UML* e *CWM* [Mda 2003].

O *SQL2CWM* acessa a interface do *CWMComp*, isto também possibilita a ele navegar por sua estrutura e coletar informações do *Modelo CWM*, a fim de também prover essa transformação de forma determinística e unidirecional, guiada por informações contidas nos *Modelos CWM* e *Códigos SQL*.

A Figura 6 ilustra por meio do *Modelo de Casos de Usos de Componentes de Transformação* o comportamento dos componentes *UML2CWM* e *CWM2SQL*. No caso do *UML2CWM*, tem-se um *Modelo UML* como entrada que é submetido ao caso de uso *Transformar Modelo UML para Modelo CWM*. Esse caso de uso organiza o projeto de transformação do *Componente Transformador de Modelos* e incorpora em uma dada seqüência o comportamento dos seguintes casos de uso: *Transformar Tipo de Dado*, *Transformar Classe*, *Transformar Atributo*, *Transformar Associação* e *Transformar Generalização*, os quais são responsáveis pela transformação de instâncias UML em instâncias CWM. Tem-se como saída do caso de uso *Transformar Modelo UML para Modelo CWM* um *Modelo CWM*.

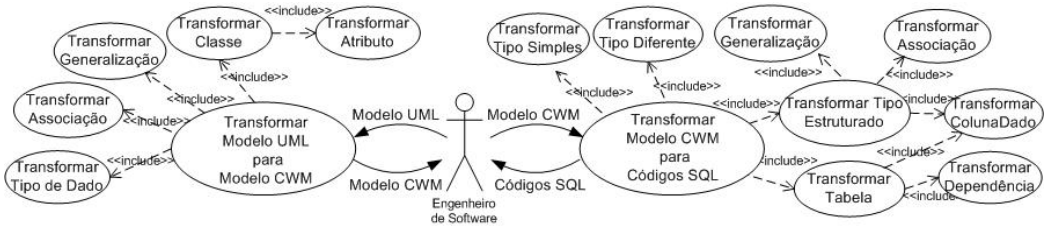

Figura 6 - Modelo de Casos de Usos de Componentes de Transformação

No caso do *CWM2SQL*, tem-se um *Modelo CWM* como entrada que é submetido ao caso de uso *Transformar Modelo CWM para Códigos SQL*. Esse caso de uso organiza o projeto de transformação do *Componente Gerador de Códigos* e incorpora em uma dada seqüência às operações *Transformar SQLSimpleType, Transformar SQLDistinctType, Transformar SQLStructuredType, Transformar Generalização, Transformar Associação, Transformar ColunaDado, Transformar Tabela* e *Transformar Dependência,* os quais são responsáveis pelo mapeamento de instâncias CWM para códigos SQL. Tem-se como saída do caso de uso *Transformar Modelo CWM para Códigos SQL* os *Códigos SQL* correspondentes ao *Modelo CWM* de entrada.

A Tabela 1 e a Tabela 2 sumarizam as transformações executadas pelos Casos de Uso da Figura 6.

Tabela 1 – Transformações executadas pelo UML2CWM

Instância UML	**Casos de Uso**	**Instância CWM**
DataType	Tipo de Dado	<cwmDataTypeList> <cwmDataTypeList> = SqlSimpleType | SqlDistinctType | SqlStructuredType
Class	Classe e Atributo	SqlStructuredType, Column, Table e Dependency
Association	Associação	Association
Generalization	Generalização	Generalization

Tabela 2 – Transformações executadas pelo CWM2SQL

Instância CWM	**Casos de Uso**	**Código SQL**
SqlSimpleType	Tipo Simples	<sqlSimpleTypeList> <sqlSimpleTypeList> = char | number | date | etc
SqlDistinctType	Tipo Distinto	Create Distinct Type *sqlDistinctTypeInstance* As <sqlSimpleTypeList>
SqlStructuredType, Column, Association e Generalization	Tipo Estruturado, ColunaDado, Associação e Generalização	Create Type *SqlStructuredTypeInstance* <elementList> <columnsList> <elementList> = As Object | As Table Of *sqlStructuredTypeInstance* | Varray [1..9] Of *sqlStructuredTypeInstance* | Under *sqlStructuredTypInstance* <columnsList> = *columnInstance* <dataTypeList> <dataTypeList> = *sqlSimpleTypeList* | *sqlDistinctTypeInstance* | <sqlStructuredTypeList> <sqlStructuredTypeList> = [*Ref*] *sqlStructuredTypeInstance*
Table e Dependency	Tabela, Dependência e ColunaDado	Create Table *tableInstance* Of *dependencyInstance* [*columnInstance* <typeList>] <typeList> = *PrimaryKey* | Nested Table Store As

A Tabela 3 mostra os estereótipos utilizados para modelar *Instâncias CWM*. Esses estereótipos ajudam a caracterizar a semântica de modelos BDOR na notação UML [Zendulka 2005].

Tabela 3 - Estereótipos de Instâncias CWM

Instância CWM	**Estereótipo**
SqlStructuredType	*ObjectType*
Table	<stereotypeList> <stereotypeList> = *ObjectTable* | *NestedTable*

4.2.Usando a MDA na MVCASE

Uma aplicação simples do domínio de vendas é utilizada para ilustrar a operacionalização dos componentes dos níveis M_3 e M_2 sobre modelos do nível M_1, isto é, constrói-se um modelo OO e a partir desse obtêm-se um modelo BDOR e por fim, seus códigos em SQL Essa aplicação é chamada de *Sale* e seu *Modelo UML* foi especificado na MVCASE usando um editor gráfico orientado pela sintaxe e semântica da UML.

A Figura 7 ilustra o *Modelo UML* da *Sale*, esse 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*.

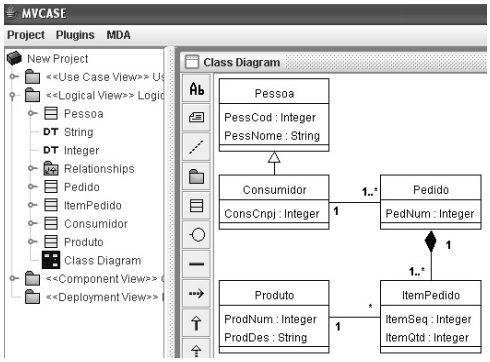

Figura 7 - Modelo UML da *Sale*

A Figura 8 ilustra o *Modelo CWM* obtido a partir do *Modelo UML* da Figura 7 por meio das transformações automáticas executadas pelo *UML2CWM*. O *Modelo CWM* obtido é um modelo BDOR da *Sale*, o qual está de acordo com as características OO contidas no padrão SQL [Sql 1999].

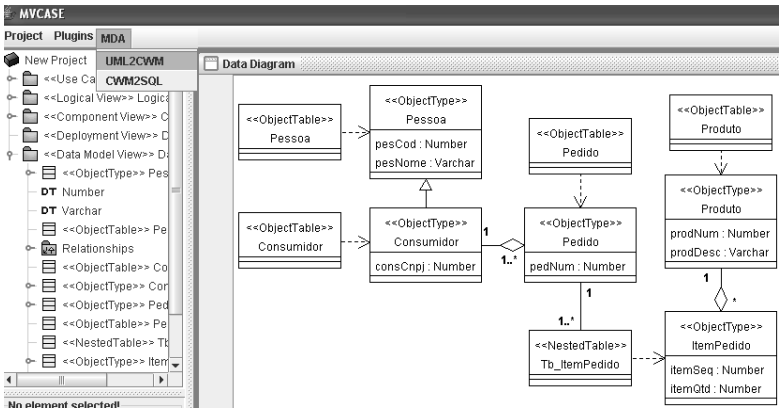

Figura 8 - Modelo CWM da *Sale*

Para obter o *Modelo CWM* a partir do *Modelo UML* da *Sale*, o *UML2CWM* recebe como entrada *Instâncias UML* e as transformam em *Instâncias CWM* de acordo com as transformações descritas na Tabela 1 adicionadas de estereótipos descritos na Tabela 3. Para o caso do *Modelo UML* da *Sale*, as seguintes regras foram utilizadas:

i) Tipo de Dado: instâncias *DataType* transformadas em *SqlSimpleType*, isto é, as instâncias Integer e String do tipo *DataType* no *Modelo UML* são transformadas nas correspondentes instâncias *Number* e *Varchar* do tipo *SqlSimpleType* no *Modelo CWM;*

ii) Classe e Atributo: instâncias *Class* transformadas em *Table*, *SqlStructuredType* e *Dependency*, isto é, as instâncias *Pessoa*, *Consumidor*, *Pedido*, *ItemPedido* e *Produto* do tipo *Class* no *Modelo UML* são transformadas em instâncias

no *Modelo CWM* do tipo *SqlStructuredType* com estereótipo *ObjectType*, *Table* com estereótipo *ObjectTable* e *Dependency;*

iii) Associação: instâncias *Association* transformadas em *Association*, isto é, as instâncias do tipo *Association* no *Modelo UML* que possuem cardinalidade "*1xN*" são transformadas em instâncias do tipo *Association* no *Modelo CWM* 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 *Modelo UML*, são modeladas com um tipo estereótipo do tipo *Nested Table*, executando ainda a atualização das instâncias *Association* e *Table* de *CWMComp;* e

iv) Generalização: instâncias *Generalization* transformadas em *Generalization*, isto é, as instâncias do tipo *Generalization* no *Modelo UML* são transformadas em instâncias do tipo *Generalization* no *Modelo CWM*.

A Figura 9 mostra os códigos em SQL obtidos a partir do *Modelo CWM* da Figura 8 com o auxílio dos *Componentes Geradores de Códigos*.

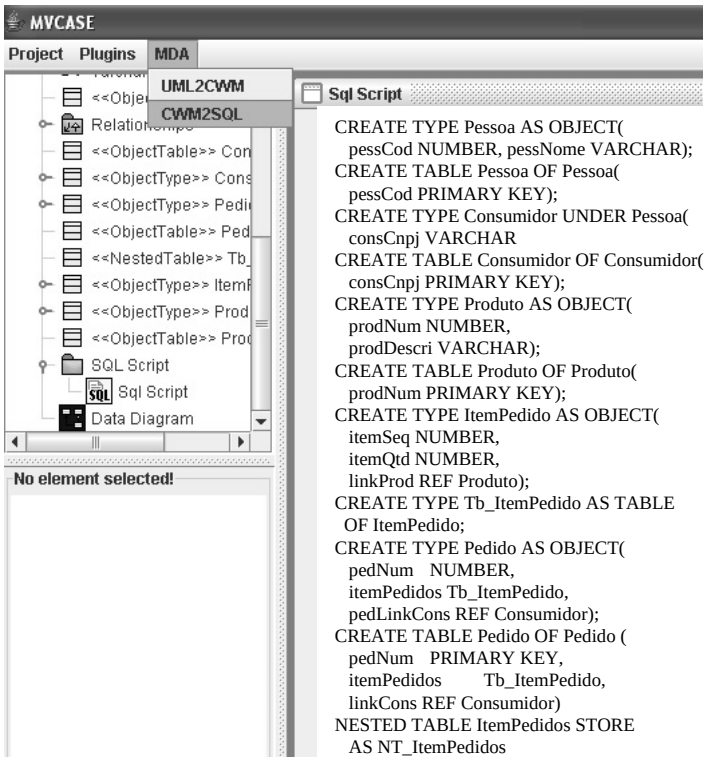

Figura 9 – Código em SQL

Para obter *Códigos SQL* a partir do *Modelo CWM* da *Sale*, o *CWM2SQL* recupera instâncias no *Modelo CWM* e as transformam, em *Códigos SQL* de acordo com as características descritas na Tabela 2. Para o caso do *Modelo CWM* da *Sale*, as seguintes regras foram utilizadas:

i) Tipo Simples: instâncias *SqlSimpleType* e *SqlDistinctType* são mapeadas para o código *"instancia.nome() instancia.type()";*

ii) Tipo Estruturado, ColunaDado, Associação e Generalização: instâncias *SqlStructuredType* que possuem estereótipos com o valor *ObjectType* e que não fazem parte como *"filha"* em um dos lados do relacionamento de instâncias *Generalization* no *Modelo CWM,* são mapeadas para o código *"Create Type instância.nome() As Object"*. Já as instâncias que fazem parte como *"filhas"* em um dos lados do relacionamento de uma instância *Generalization,* 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 "*Cre*ate *Type Consumidor UNDER Pessoa*", indicando que o *Consumidor* é uma especialização de *Pessoa;*

iii) Tabela, Dependência e ColunaDado: instâncias *Table* que possuem estereótipos com o valor *ObjectTable* e que possuem uma instância *Dependency* para uma outra instância *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;*

iv) Tipo Estruturado, ColunaDadoTabela, Associação e Generalização: instâncias do tipo *SqlStructuredType* 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 *Tb_ItemPedido* origina o código *"Create Type Tb_ItemPedido As Table Of ItemPedido";*

v) Tipo Estruturado, ColunaDadoTabela, Associação e Generalização: instâncias *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 instânciaLadoSemAggregate.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";* e

vi) Tipo Estruturado, ColunaDadoTabela, Associação e Generalização: 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' instânciaTb.nome() Nested Table instânciaTb.nomeSemTb()+s Store As Nt_+instânciaTb.nomeSemsTb()",* essa sintaxe SQL cria uma 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"*.

5. Trabalhos Correlatos

Diversas abordagens têm sido propostas para a transformação de modelos, dentre elas encontra-se: a *Query/Views/Transformations* (QVT) [Qvt 2005], que visa permitir a definição das regras de transformações unidirecionais por meio da abordagem declarativa e imperativa; a *MOF to Text Transformation* [M2T], que define uma abordagem baseada em *Templates* para especificar as regras de transformação entre modelos baseados no MOF e texto; a *Visual Automated Model* (VIATRA) [Varró *et al*. 2004], 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* [Tratt 2006], 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 deste trabalho, dentre elas: o projeto *Generative Modeling Technologies* (GMT) [Gmt 2006]; e o *framework* AndroMDA [Andromda 2006].

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 Modelo-Modelo* através da combinação de linguagem declarativa e imperativa. As regras de transformações são descritas em ATL [M2m 2006] e são processadas pelo ATL *Development Tooling* (ADT), o qual funciona com o suporte da *Integrated Development Environment* (IDE) Eclipse. Os componentes de transformação apresentados neste artigo possuem dependência da máquina virtual Java e de outros componentes de metamodelos.

O AndroMDA destaca-se pela sua estabilidade e relativa maturidade. Ele é um *framework* extensível para a MDA que recebe como entrada um 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. No AndroMDA, a geração de códigos é dirigida por estereótipos especificados no modelo UML, servindo como guias para os *templates.* Portanto, os modelos UML especificados precisam ser mais elaborados e completos. Já o protótipo proposto, necessita apenas de um modelo UML simples, sem a necessidade de estereótipos para guiar a transformação. Assim, o Engenheiro de *Software* precisa se concentrar apenas nos aspectos relacionados ao domínio do problema, deixando que questões relacionadas as características particulares de cada plataforma sejam adicionadas automaticamente na transformações.

A implementação proposta diferencia-se ainda dos trabalhos citados porque permite que os *Componentes de Transformação de Modelos* e de *Geração de Códigos*, isto é, os componentes que possibilitam a transformação de modelos OO em modelos BDOR e códigos SQL, sejam reutilizados por outras ferramentas que utilizam o MDR [Matula 2003] como repositório de metadados e que utilizam os metamodelos UML [Omg 2004] e CWM [Omg 2003].

6. Conclusão

A principal contribuição deste trabalho compreende a integração das diferentes linguagens MOF, UML, CWM e XMI da OMG com os componentes que provêem a Transformação Modelo-Modelo e Modelo-Texto, isto é, de modelos OO para modelos BDOR e posteriormente, códigos em SQL. Essas transformações são viabilizadas por meio de componentes que podem ser reutilizados em outras ferramentas de desenvolvimento de *software*. A transformação de modelos é uma contribuição importante, visto que possibilita o reuso de informações entre plataformas diferentes, proporcionando maior rapidez e qualidade no desenvolvimento de *software* além de não exigir do Engenheiro de *Software* nenhum conhecimento específico na área de modelagem de BDOR.

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 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 através do uso de um editor gráfico orientado pela sintaxe e semântica; e a criação de um metamodelo para o domínio do rastreamento das instâncias dos Metamodelos *Origem* para o *Destino.*

Referências

Andromda (2006). AndroMDA. Disponível em http://www.andromda.org, Dezembro.

Booch, G., Rumbaugh, J., Jacobson, I. (2005) The Unified Modeling Language User Guide (Object Technology). Addison Wesley. 2th Rev Edition.

Czarnecki, K. (2004) "Overview of Generative Software Development". *Unconventional Programming Paradigms (UPP)*, Mont Saint-Michel, France. pp. 313–328.

Czarnecki, K., Antkiewicz, M. Kim, C.H.P., Lau, S., Pietroszek, K. (2005). "Model-Driven Software Product Lines". *ACM SIGPLAN - Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA) – Poster Session.* pp. 126-127.

Czarnecki, K., Helsen, S. (2006) "Feature-Based Survey of Model Transformation Approaches". *IBM Systems Journal.* pp. 621-645.

D'Souza, D. F., Wills, A. C., (1999). Objects, Components and Frameworks with UML, *The Cataliysis Approach. Addison-Wesley. USA.*

Gamma, E., Helm, R., Johnson, R. (1995). *Design Patterns: Reusable Object-Oriented Software.* Addison-Wesley.

Gmt (2006). Generative Modeling Technologies (GMT). Eclipse Modeling Project (EMP). Disponível em http://www.eclipse.org/gmt, Novembro.

Kleppe, A., Warmer, J., Bast, W. (2003) MDA Explained, The Model-Driven Architecture: Practice and Promise. Addison Wesley.

Matula, M. (2003). "NetBeans Metadata Repository". *NetBeans Community*. Disponível em http://mdr.netbeans.org/docs.html, Setembro/2006.

Mda (2003). The Model-Driven Architecture - Guide Version 1.0.1, OMG Document: omg/2003-06-01.

Mens, T., Gorp, Van Gorp, P. (2005) "A Taxonomy of Model Transformation and Its Application to Graph Transformation" *Proccedings of the International Workshop on Graph and Model Transformation.* Estônia. pp. 7-23.

Mof (2002). Meta Object Facility 1.4 (MOF) Specification. Object Management Group (OMG). Document formal/2002-04-03.

M2m (2006). Model-to-Model Transformation (M2M). Eclipse Modeling Project (EMP). Disponível em http://www.eclipse.org/proposals/m2m, Novembro.

M2t (2006). Mof to Text Transformation. Specification. Object Management Group (OMG). Document: omg/2006-11-01.

Omg (2004). Unified Modeling Language (UML) Specification, version 1.4. *Object Management Group* (OMG). Document formal/04-07-02.

Omg (2003). Common Warehouse Metamodel (CWM) Version 1.1. *Object Management Group* (OMG). Document formal/2003-03-02.

Paiva, D.M.B., Lucrédio, D., Fortes, R.P.M. (2006) "MVCASE - including design rationale to help modeling in research projects." *XX - Simpósio Brasileiro de Engenharia de Software (XX SBES) - Sessão de Ferramentas.* Florianópolis - SC - Brasil.

Poole, J., Chang, D., Tolbert, D., Mellor, D. (2003) Common Warehouse Metamodel - Developer's Guide. Willey Publishing, Inc.

Qvt (2005). Query/Views/Transformations (QVT). OMG Document: omg/2005-11-01.

Sql (1999). International Organization for Standardization (ISO) & American National Standards Institute (ANSI) - ISO/IEC JTC1/SC32 - ANSI ISO/IEC 9075-2:1999. ISO International Standard. Database Language - SQL - Parte 2: Fundation (SQL-Foundation), 1999.

Stahl, T., Völter, M. (2006) "Model-Driven Software Development – Technology, Enginnering, Management". John Willey and Sons ltda., England.

Tratt, L. (2006). "The MT Model Transformation Language", *Proceedings of ACM Special Interest Group on Applied Computing (SIGAC) - Session: Model transformation*, Dijon, France. pp. 1296-1303.

Varró, D., Varró G., Pataricza, A. (2004) "Generic and Meta-Transformation for Model Transformation Engineering", *Proceedings of the 7th International Conference on Unified Modeling Language,* Lisboa, Portugal. pp. 290-304.

Xmi (2003). XML Metadata Interchange 1.3 (XMI) Specification. Object Management Group (OMG). Document formal/03-05-01.

Xml (2006). Extensible Markup Language (XML). W3C Architecture Domain. Disponível em http://www.w3.org/XML, Dezembro.

Zendulka, J. (2005). "Object-Relational Modeling in UML", Encyclopedia of Database Technologies and Applications, Idea Group Publishing, Hershey, US. pp. 421-426