Universidade Federal de São Carlos
Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

Transformação de Modelos Orientados a Objetos em Modelos de Banco de Dados Objeto-Relacional

MARCO ANTONIO PEREIRA

São Carlos
Dezembro/2007

UNIVERSIDADE FEDERAL DE SÃO CARLOS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

"Transformação de Modelos Orientados a Objetos em Modelos de Banco de Dados Objeto-Relacional"

MARCO ANTONIO PEREIRA

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Antonio Francisco do Prado
(Orientador – DC/UFSCar)

Prof. Dr. Mauro Biajiz
(Co-Orientador – DC/UFSCar)

Profa. Dra. Marilde Teresinha Prados Santos
(DC/UFSCar)

Prof. Dr. Carlos Roberto Valêncio
(IBILCE/UNESP)

São Carlos - SP

Dezembro/2007

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

P436tm

Pereira, Marco Antonio.
Transformação de modelos orientados a objetos em modelos de banco de dados objeto-relacional / Marco Antonio Pereira. -- São Carlos : UFSCar, 2008.
82 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2007.

1. Engenharia de software. 2. UML. 3. MDA. 4. Banco de dados objeto-relacional. 5. Transformação de modelos. 6. SQL (Linguagem de programação de computador). I. Título.

CDD: 005.1 (20ª)

Agradecimentos

Em especial, agradeço a Deus, aos meus pais e a minha irmã pelo auxílio direto e indireto na construção de um caráter inquieto e um senso crítico que me trouxeram até este ponto. Aqui cabem também meus sinceros agradecimentos a minha companheira de coração, Nathália, pelo amor que me conforta em todos os momentos e por trazer junto dela meus sogros, que tanto me motivaram nesta conquista.

Meu muito obrigado aos professores Antonio Francisco do Prado e Mauro Biajiz, pelas orientações de caráter pessoal e acadêmico em relação ao "ser pesquisador", ao objeto "pesquisa" e às correções intelectuais que culminaram neste trabalho. Um agradecimento especial pela paciência e pelo empenho desses dois distintos professores.

Aos meus novos e verdadeiros amigos, Val e Edson, pelos quais tenho muita estima e carinho. Mais uma vez, muito obrigado pelo companheirismo, pela co-autoria nesta pesquisa e como não haveria de faltar, pelas boas horas de boteco.

Obrigado a todos os meus amigos e colegas do mestrado que proporcionaram boas risadas na hora do "café dos gordos", em especial aos titulares Amanda, Cris e Laia.

Obrigado também a todo pessoal do Departamento de Computação da UFSCar e aos funcionários que conheci aqui.

Sei que posso não dar conta de expressar meus sinceros agradecimentos a muitos e tantos adorados familiares e amigos, sem os quais eu não chegaria aqui.

Obrigado.

"– Gato de Cheshire... Poderia me dizer, por favor, qual é o caminho que devo tomar pra sair daqui?

– Isso depende muito de pra onde você quer ir – respondeu o Gato.

– Não me importo muito pra onde... – retrucou Alice.

– Neste caso, não importa muito o caminho a ser tomado – disse o Gato.

– ... contanto que dê em algum lugar – acrescentou Alice.

– É claro que isso acontecerá – disse o Gato – desde que você caminhe bastante."

(Lewis Carroll, *Alice no país das maravilhas,* apud René Dubos, *O despertar da razão,* 1972, p. 165).

Resumo

Em geral, os sistemas de *software* implementados com base nos conceitos Orientados a Objetos (OO) persistem suas informações em Banco de Dados Relacional (BDR) e mais recentemente em Banco de Dados Objeto Relacional (BDOR), os quais suportam conceitos do paradigma OO. Portanto, é comum a utilização de Modelos OO para especificação dos requisitos de um sistema de *software* cujas informações são persistidas em BDOR. Assim, a transformação de Modelos OO para Modelos de Banco de Dados tem sido objeto de inúmeras pesquisas, particularmente no caso de Modelos BDOR. Motivados em pesquisar a transformação de modelos OO em Modelos de BDOR e em Códigos *Structured Query Language* (SQL), este trabalho investigou uma abordagem baseada na idéia do desenvolvimento dirigido por modelos do *Object Management Group* (OMG), a *Model Driven Architecture* (MDA), e viabilizou-a em forma de protótipo construído como extensão da ferramenta *Multiple Views Case* (MVCASE). Como principal resultado desta pesquisa tem-se a aplicação de uma abordagem que auxilia o Engenheiro de *Software* no desenvolvimento de Modelos OO e sua transformação para Modelos de BDOR e conseqüente geração de Códigos SQL.

Abstract

In general, software systems implemented based on Object Oriented (OO) concepts persist their information in Relational Database (RDB) and more recently in Object Relational Database (ORDB), which support the OO paradigm concepts. Therefore, it is common to use OO Models for software system requirements specification whose information are persisted in ORDB. Thus, the transformation of OO Models into Database Models has been an issue of several researches, particularly in the case of ORDB Models. Motivated in researching the transformation of OO models in ORDB Models and in Codes Structured Query Language (SQL), this work investigated an approach based on the idea of model driven development of Object Management Group (OMG), Model Driven Architecture (MDA), and offered it in a prototype form built as extension of the Multiple Views CASE (MVCASE) tool. As main result of this research we have the application of an approach that helps the Software Engineer in the development of OO Models and its transformation into ORDB Models and consequent generation of SQL Codes.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AE – Análise Estruturada

ANSI – American National Standards Institute

BI – Business Intelligence

CASE – Computer-Aided Software Engineering

CWM – Common Warehouse Metamodel

DER – Diagrama Entidade-Relacionamento

DFD – Diagrama de Fluxo de Dados

DTD – Document Type Definition

GOES – Grupo de Engenharia de Software

BDR – Banco de Dados Relacional

BDOO – Banco de Dados Orientados a Objeto

BDOR – Banco de Dados Objeto Relacional

CIM – Computation Independent Model

DDL – Data Definition Language

DML – Data Definition Language

DW – Data Warehouse

EMF – Eclipse Modeling Framework

GMT – Generative Modeling Technologies

IBM – International Business Machines

IDL – Interface Definition Language

IEC – International Electrotechnical Commission

ISO – International Organization for Standardization

JMI – Java Metadata Interface

MDA – Model Driven Architecture

MDD – Model-Driven Development

MDR – Metadata Repository

MOF – Meta Object Facility

MVCASE – Multiple Views Case

OCL – Object Constraint Language

OID – Object Identifiers

OLAP – On-Line Analytical Processing

OMG – Object Management Group

OMT – Object Modeling Technique

OO – Orientado a Objeto

PIM – Plataform Independent Model

PSM – Plataform Specific Model

SGBD – Sistema de Gerenciamento de Banco de Dados

SGBDOO – Sistema de Gerenciamento de Banco de Dados Orientados a Objetos

SGBDOR – Sistema de Gerenciamento de Banco de Dados Objeto Relacional

SGBDR – Sistema de Gerenciamento de Banco de Dados Relacional

SQL – Structured Query Language

TI – Tecnologia da Informação

UDT – User-Defined Types

UFSCar – Universidade Federal de São Carlos

UML – Unified Modelling Language

VTL – Velocity Template Language

W3C – World Wide Web Consortium

XMI – XML Metadata Interchange

XML – EXtensible Markup Language

Sumário

Capítulo 1

Introdução

O contexto em que os sistemas de *software* são desenvolvidos está estreitamente ligado a quase cinco décadas de evolução dos sistemas computadorizados. Inicialmente, toda a codificação desses sistemas era realizada em linguagens de baixo nível, muito próximas às linguagens de máquina, com pouca expressividade e de difícil manutenção. Com o passar do tempo esses sistemas evoluíram e se tornaram mais complexos tornando-se necessária a criação de novas linguagens e paradigmas de programação, como as linguagens orientadas a objeto que possibilitam a abstração em alto nível, facilitando assim a programação e aumentando a manutenibilidade desses sistemas [Mellor *et al.* 2004].

Paralelamente, linguagens de modelagem foram surgindo, tornando-se assim possível a construção de modelos para especificar os sistemas de *software*. No princípio esses modelos permitem a manipulação e visualização das especificações de um sistema durante seu desenvolvimento, possibilitando uma melhor avaliação dos requisitos antes deles serem implementados. Inicialmente, esses modelos eram utilizados para facilitar a comunicação entre as equipes de desenvolvimento e documentar o processo de desenvolvimento, porém, com o passar do tempo, começaram a fazer parte não só do processo de desenvolvimento, mas também do próprio sistema de *software,* surgindo assim, o paradigma do desenvolvimento dirigido por modelos, o qual se utiliza de modelos em diferentes níveis de abstração para gerar o código em

linguagem de programação, assim, modelos começaram a ser considerados artefatos de *software* que podiam ser transformados em outros modelos e códigos em linguagens de programação [CZARNECKI *et al.* 2005]. A transformação de modelos é o principal conceito para se obter os benefícios do desenvolvimento dirigido por modelos, é responsável por mapear os modelos abstratos para modelos menos abstratos e, por conseguinte, mapeá-los para uma determinada linguagem de programação [OMG 2003Mda].

Segundo [KLEPPE *et al.* 2003], os benefícios obtidos com o uso do desenvolvimento dirigido por modelos podem ser resumidos em:

- **Produtividade**: o desenvolvedor concentra seus esforços no desenvolvimento de um modelo em alto nível de abstração. Com a aplicação da transformação de modelos é possível obter modelos menos abstratos que contenham detalhes técnicos, os quais são especificados nas regras de mapeamento e adicionados automaticamente durante o processo de transformação, aumentando a produtividade no desenvolvimento dos sistemas de *software;*
- **Portabilidade**: um único modelo abstrato pode ser totalmente portável para múltiplos modelos menos abstratos;
- **Interoperabilidade**: os modelos menos abstratos podem ser relacionados para formar um sistema completo, como por exemplo, códigos em linguagem Java que acessam um Banco de Dados Relacional, quando ambos são gerados a partir de um único modelo abstrato; e
- **Manutenção e documentação**: a manutenção é feita diretamente nos modelos abstratos, gerando-se novamente os modelos menos abstratos e os seus respectivos códigos, assim, os modelos são a exata representação do código e servem também como documentação dos artefatos de *software.*

Outro aspecto importante do processo de desenvolvimento de sistemas de *software* é o da persistência das informações. Grande parte dos sistemas necessita armazenar suas

informações para posteriormente recuperá-las, e para tal, o meio mais comumente utilizado são os bancos de dados. No contexto de sistemas computacionais, um banco de dados é um repositório que armazena uma coleção de arquivos de dados computadorizados [DATE 2003]. Um Banco de Dados Relacional (BDR) é baseado no Modelo Relacional [CODD 1970].

Os Sistemas de Gerenciamento de Banco de Dados (SGBD) são *softwares* que tratam todo o acesso aos dados de um banco de dados, logo, os SGBDs que tratam o acesso à BDRs são chamados de SGBD Relacional (SGBDR). Os SGBDR suportam uma coleção limitada de tipos fixos e, acredita-se que sejam insuficientes para capturar os problemas dos sistemas de *software* atuais, tais como sistemas multimídias, geográficos, espaciais, médicos, projetos auxiliados por computador, etc. Assim, para tentar solucionar esses problemas e alinharem-se às novas necessidades surgiram os Bancos de Dados Orientados a Objetos (BDOO), que possuem semântica rica para a modelagem dos problemas atuais e possibilitam a análise e o projeto com foco no problema [ÖZSU 1996].

Visto que o Modelo Relacional pode ser estendido sem comprometer seus princípios fundamentais [DARWEN e DATE 1995, DATE 2003], os conceitos da Orientação a Objetos (OO) podem ser suportados por esse modelo. Um BDR baseado nessa extensão é chamado de Banco de Dados Relacional Estendido (BDRE), porém, com o objetivo de enfatizar as características OO acrescentadas ao Modelo Relacional, ele é geralmente chamado de Banco de Dados Objeto Relacional (BDOR) e, por conseguinte, um SGBD que trata do acesso aos dados de um BDOR é chamado de Sistema Gerenciador de Banco de Dados Objeto Relacional (SGBDOR) [EISENBERG e MELTON 1999].

Uma vantagem dos BDORs em relação aos BDRs é o fato dos primeiros permitirem o acréscimo de tipos de dados que podem ser armazenados e, em relação aos BDOOs, é o fato deles serem considerados como tecnologias "maduras", pois têm sido a base para um grande número de sistemas de *software* ao redor do mundo.

O Engenheiro de *Software* pode utilizar ferramentas de engenharia de *software* assistida por computador *(Computer-Aided Software Engineering* - CASE) para auxiliá-lo nas atividades do processo de desenvolvimento de sistemas de *software*. Em [CRONHOLM 1995] é mostrado um estudo que buscou identificar os principais motivos que levam fabricantes de sistemas de *software* a investirem em ferramentas CASE. Segundo a pesquisa, as ferramentas CASE podem ajudar em dois pontos:

- **Aumentar a produtividade**: a ferramenta CASE pode tornar o processo de desenvolvimento mais simples e fácil de ser seguido, além de tornar o processo de documentação mais eficaz; e
- **Aumentar a qualidade do sistema de *software* produzido**: o uso de uma ferramenta CASE pode tornar o processo de desenvolvimento mais gerenciável, com conseqüente aumento da qualidade do sistema de *software* produzido.

Assim, motivados pela idéia de prover mecanismos para auxiliar o Engenheiro de *Software* na modelagem das informações dos sistemas atuais, este trabalho investigou uma abordagem de desenvolvimento dirigido por modelo, aplicou-a para o desenvolvimento desses sistemas que necessitam armazenar suas informações em BDOR e disponibilizou-a em uma ferramenta CASE.

1.1 Objetivo e Escopo

O objetivo deste projeto de pesquisa foi fornecer um estudo de uma abordagem dirigida por modelos enfatizando o desenvolvimento de sistemas de *software* do domínio de BDOR e implementá-lo em uma ferramenta CASE a fim de auxiliar o Engenheiro de *Software* na transformação de Modelos OO em Modelos BDOR e Códigos *Structured Query Language* (SQL).

Para viabilizar essa idéia, a ferramenta MVCASE [LUCRÉDIO 2005] foi estendida para incorporar os conceitos da abordagem *Model Driven Architecture* (MDA) [OMG 2003Mda]. Para tanto, fez-se necessário delimitar o escopo, visto que a transformação de modelos e os BDOR são alvos de inúmeras linhas de pesquisa. Assim, podem-se listar as seguintes características tratadas:

- Quanto às características das linguagens utilizadas podem-se citar:
  - **Linguagens padronizadas**: a utilização de linguagens padronizadas a fim de facilitar o intercâmbio de informações entre ferramentas de modelagem de sistemas de *software;* e
  - **Notação gráfica padronizada**: notação gráfica padronizada para facilitar o entendimento semântico dos modelos entre os membros de equipes de desenvolvimento.
- Quanto às características de transformação de modelos podem-se citar:
  - **Transformações unidirecionais**: a transformação se dá em um único sentido, de Modelos OO para Modelos BDOR, e por fim, para Códigos SQL;
- Quanto às características semânticas dos modelos podem-se citar:
  - **Escopo das características semânticas dos Modelos OO**: definiu-se um escopo de abrangência sobre os conceitos OO, os quais são melhores explicados no capítulo 2:
    - Classes;
    - Atributos;
    - Generalização simples; e
    - Associação, agregação e composição com as seguintes multiplicidades possíveis: um pra um, um pra muitos e muitos pra muitos.

- **Escopo das características semânticas dos Modelos BDOR**: com base no escopo dos Modelos OO, foram definidas as características semânticas de um Modelo BDOR, as quais são melhores explicadas no capítulo 2:
  - Tipos definidos pelo usuário;
  - Tabelas baseadas em tipos estruturados;
  - Tabelas aninhadas;
  - Colunas;
  - Generalização simples; e
  - Relacionamentos binários do tipo *referências* com as seguintes multiplicidades possíveis: um pra um e um pra muitos.

- E por fim, quanto à metodologia de desenvolvimento pode-se citar:
  - **Metodologia**: desenvolvimento de artefatos de *software* reutilizáveis, isto é, componentes de *software* que podem ser reutilizados em outros ambientes de desenvolvimento de sistemas de *software*. Um componente é um pacote coerente de artefatos de software que pode ser independentemente desenvolvido e entregue como unidade, e ainda pode ser composto, sem mudança, com outros componentes para construir algo maior [D'SOUZA e WILLS 1999].

Na seção a seguir tem-se a estrutura desta dissertação.

1.2 Estrutura da Dissertação

Este primeiro capítulo apresentou o contexto, o objetivo e o escopo desta dissertação de mestrado.

No capítulo 2 abordam-se os principais conceitos relacionados ao trabalho de pesquisa apresentado nesta dissertação. Neste capítulo discorre-se sobre o cenário atual do desenvolvimento dirigido por modelos, apresentando a abordagem MDA [OMG 2003Mda] juntamente com uma descrição das linguagens *Meta Object Facility* (MOF) [OMG 2002Mof], *Unified Modeling Language* (UML) [OMG 2007Uml], *Common Warehouse Metamodel* (CWM) [OMG 2003CWM], *XML Metadata Interchange* (XMI) [OMG 2005Xmi] e *Structured Query Language* (SQL) [ISO/IEC 1999]. E ainda, neste capítulo são abordados os conceitos de transformação de modelos, a ferramenta MVCASE e suas principais funcionalidades.

No capítulo 3 é apresentada em detalhes a abordagem deste trabalho de pesquisa, o qual contém as definições das transformações de Modelos OO para BDOR e Códigos SQL, além da implantação dessas transformações na MVCASE, a fim de oferecer o suporte ao processo de desenvolvimento de sistemas de *software* do domínio BDOR a partir de Modelos OO.

O capítulo 4 apresenta um estudo de caso, juntamente com os resultados para avaliar o protótipo.

Finalmente, no capítulo 5 são apresentados os trabalhos relacionados, considerações finais e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

No contexto do desenvolvimento de sistemas de *software* dirigido por modelos, encontra-se o *Model-Driven Development* (MDD) [VÖLTER e STAHL 2006], o qual tem como foco os modelos que representam a abstração do mundo real. O MDD ilustra a idéia do desenvolvimento de sistemas de *software* em um nível mais abstrato, mantendo os detalhes da implementação separados de suas especificações [MELLOR *et al.* 2004]. Com esse propósito, os Engenheiros de *Software* podem concentrar seus esforços na construção de modelos que representam os conceitos e funcionalidades de um sistema de *software,* dando maior enfoque apenas nos requisitos do negócio a fim de modelar as funcionalidades desse sistema. Um sistema é um conjunto de princípios coordenados entre si de maneira a formar um todo. Um sistema de *software* pode ser entendido como uma combinação de diversas partes que juntas produzem um determinado resultado [OMG 2003Mda].

Modelos podem ser utilizados para especificar sistemas de *software.* Um modelo é uma representação simplificada, com o objetivo de observação, manipulação e entendimento sobre algum conceito [MELLOR *et al.* 2004]. Seguem três outras definições de modelos:

- A especificação *Unified Modeling Language* (UML) [OMG 2007Uml] diz que modelo é captação de uma visão de um sistema físico, é uma abstração desse sistema com certo propósito. Esse propósito determina o que está incluído no modelo e qual sua relevância. Assim, o modelo descreve

completamente aqueles aspectos do sistema físico que são relevantes para o propósito do modelo em um apropriado nível de detalhe;

- Os autores em [KLEPPE *et al.* 2003] definem modelo como a descrição de um sistema escrito em uma linguagem bem definida. Uma linguagem bem definida é uma linguagem com sintaxe e semântica bem definidas; e
- No contexto do desenvolvimento de sistemas de *software,* modelos são criados com o objetivo de descrever ou especificar a relação de um sistema com o seu ambiente em um determinado propósito [OMG 2003Mda].

Em resumo, modelos devem estar habilitados a responder questões no lugar de um determinado sistema. Conforme ilustra a Figura 1, os modelos representam um sistema de *software,* que por sua vez representam um sistema. Um sistema é composto por seus requisitos, funcionalidades e plataforma. Os requisitos são as características pertinentes a determinadas funcionalidades que compõem o sistema e que provêem seu funcionamento sobre uma plataforma. A plataforma é um conjunto de subsistemas e tecnologias que disponibilizam um coerente conjunto de funcionalidades a um sistema sem que este se preocupe com os detalhes de implementação desta [OMG 2003MDA].

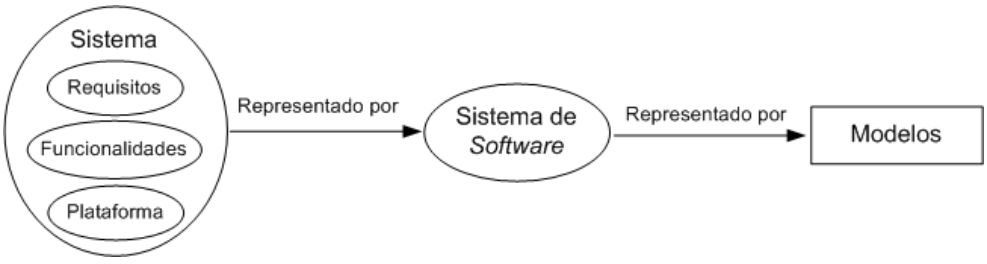


Figura 1 – Modelos e Sistemas

Dentro deste contexto, a *Model Driven Architecture* (MDA) [OMG 2003Mda] é a abordagem do *Object Management Group* (OMG) [OMG 2006Omg] ao MDD. A seguir é discutida essa abordagem.

2.1 *Model Driven Architecture (MDA)*

A MDA fundamenta-se na idéia da separação entre as especificações de um sistema de *software* e os detalhes de sua implementação. Os modelos que descrevem esse sistema são especificados em diferentes níveis de abstração e utilizados juntos a fim de se obter a partir destes uma implementação [MELLOR *et al.* 2004].

A Figura 2 apresenta uma taxonomia para diversos tipos de modelos utilizados no desenvolvimento com a MDA [FRANKEL 2003]. Os modelos concretos são aqueles que realmente são construídos no processo de desenvolvimento de sistemas de *software,* já os modelos abstratos são apenas para classificação na taxonomia.

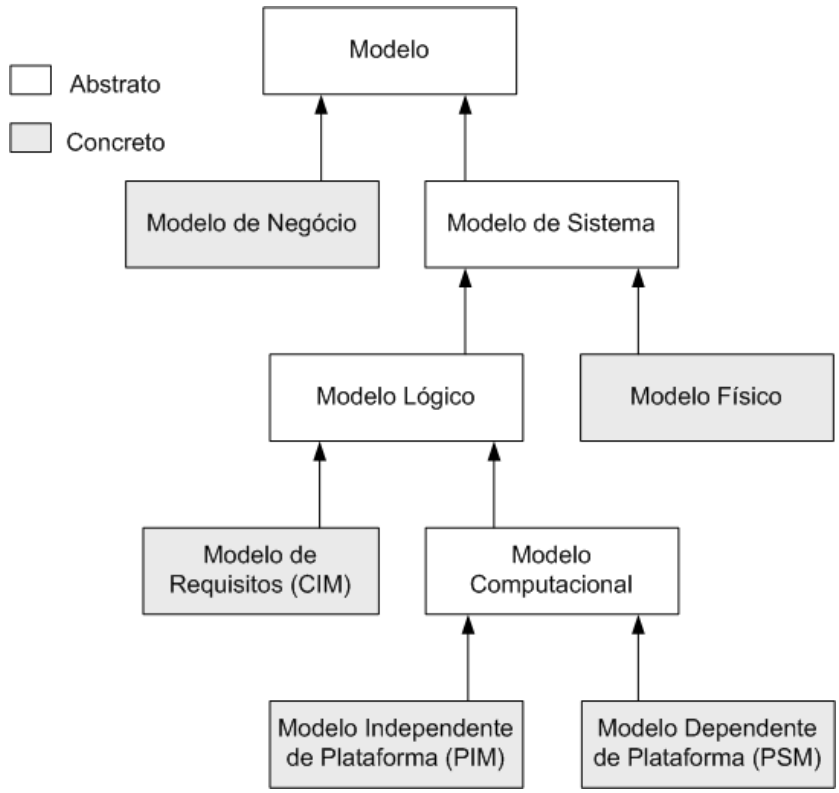


Figura 2 – Taxonomia de Modelos [FRANKEL 2003]

Esses modelos são os seguintes:

- **Modelo de Negócio**: descreve os aspectos à respeito do funcionamento das regras de um determinado negócio;

- **Modelo de Sistema**: descreve os aspectos do negócio que são automatizados em um sistema de *software;*
- **Modelo Lógico**: descreve os elementos, relacionamentos e comportamento para atingir um determinado propósito de acordo com os aspectos lógicos;
- **Modelo Físico**: descreve os artefatos físicos usados durante a execução de um sistema, como por exemplo, os arquivos executáveis, o código fonte, o processador, etc;
- **Modelo de Requisitos**: descreve uma visão geral independente de seus detalhes computacionais, ou seja, descreve os requisitos, geralmente de maneira descritiva e em linguagem natural, que devem ser atendidos para um modelo lógico possuir determinada semântica sobre um domínio de problemas. Esse modelo é a representação mais abstrata de um sistema e é conhecido também por Modelo Computacionalmente Independente *(Computation Independent Model* - CIM);
- **Modelo Computacional**: também descreve um modelo lógico, mas são modelos que podem ser entendidos no âmbito computacional;
- **Modelo Independente de Plataforma** ***(Plataform Independent Model*** - PIM): descreve, com base nas características de uma determinada plataforma, os aspectos que são independentes dessa plataforma, e por isto, o conceito de PIM é considerado relativo; e
- **Modelo Dependente de Plataforma** ***(Plataform Specific Model*** - PSM): descreve os aspectos que caracterizam os detalhes de como os recursos e serviços presentes na plataforma são utilizados no modelo do sistema.

Os PSMs são modelos menos abstratos que os PIMs, e por isso, podem ser obtidos a partir do refinamento desses. A especificação da transformação de um PIM em um PSM é

realizada através de mapeamentos que devem ser automatizado o tanto quanto possível [KLEPPE 2003]. Podem-se utilizar mapeamentos não apenas para caracterizar os conceitos de uma determinada plataforma sobre um PIM, mas também para qualquer transformação entre modelos.

Conforme ilustra a Figura 3, o processo de utilização de uma abordagem MDA no desenvolvimento de sistemas de *software* consiste basicamente na criação do PIM utilizando-se de alguma linguagem de modelagem, na execução de transformações sobre esse PIM para a obtenção do PSM, e por fim, na geração de códigos na linguagem de programação da plataforma específica [FRANKEL 2003]. Um sistema de *software* pode ser gerado automaticamente a partir de uma especificação escrita em uma linguagem textual ou gráfica de um determinado domínio de problemas [CZARNECKI *et al.* 2005].

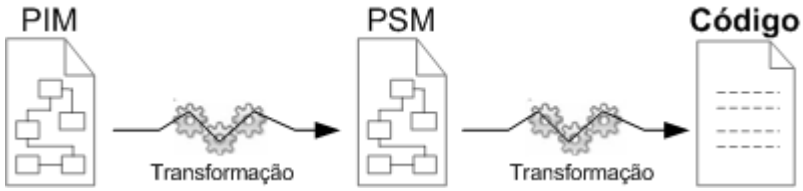


Figura 3 – Utilização da MDA [FRANKEL 2003]

Outros conceitos da MDA são detalhados nos próximos itens desta subseção.

2.1.1 *Arquitetura*

A MDA é organizada na forma de um *framework*, composto por vários conceitos e padrões. As linguagens utilizadas na MDA são linguagens padronizadas pelo OMG para especificar os artefatos de *software* nos diferentes níveis de abstração da arquitetura [OMG 2003Mda].

A Figura 4 ilustra os diferentes níveis de abstrações (M_0, M_1, M_2 e M_3) da MDA. A meta-metalinguagem é descrita pelo meta-metamodelo, as metalinguagens são descritas pelos seus metamodelos, e as linguagens são descritas pelos modelos. Um modelo é especificado com base no formalismo da linguagem descrita pelo seu metamodelo, assim como um metamodelo é

especificado com base no formalismo da linguagem descrita pelo seu meta-metamodelo [FRANKEL 2003].

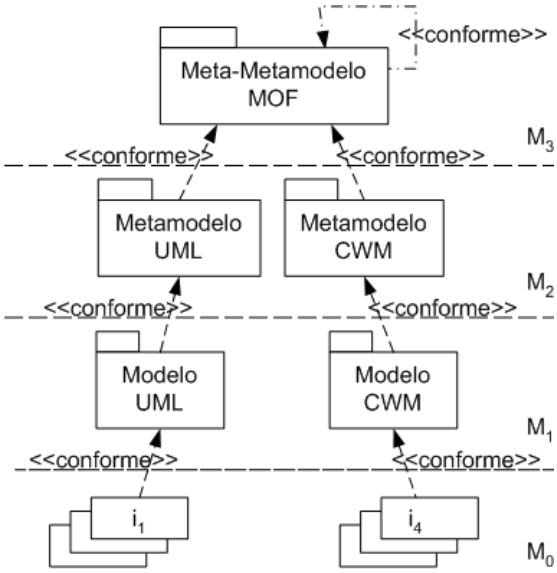


Figura 4 – Níveis de Abstrações da MDA [OMG 2002Mof]

No nível M_3, o mais abstrato, encontra-se o meta-metamodelo *Meta Object Facility* (MOF) [OMG 2002Mof], que é um modelo conforme seu próprio meta-metamodelo. O MOF descreve uma meta-metalinguagem abstrata utilizada para especificar outras metalinguagens para diferentes domínios e também descreve um repositório de metadados para suportar metamodelos baseados no próprio MOF [MATULA 2006].

Os metamodelos no nível M_2 são especificados de acordo com a linguagem descrita pelo MOF, também descrevem as propriedades de uma plataforma particular [MELLOR *et al.* 2004], como por exemplo, o Metamodelo UML que descreve uma metalinguagem para especificar modelos do domínio OO, e *Common Warehouse Metamodel* (CWM) [OMG 2003Cwm] que descreve uma metalinguagem para especificar modelos dos domínios *Data Warehouse* (DW) e *Business Intelligence* (BI). O metamodelo descreve formalmente o modelo, a sintaxe e a semântica compreendida para manipulação dos elementos desse modelo.

Os modelos no nível M_1, o *Modelo UML* e o *Modelo CWM*, são modelos em conformidade com seus correspondentes metamodelos do nível M_2. Finalmente no nível 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. O nível M_0 é o menos abstrato da arquitetura, ele representa os objetos do "mundo real".

Em resumo, um meta-metamodelo está para um metamodelo assim como um metamodelo está para um modelo e um modelo está para os objetos do "mundo real".

2.1.2 *Transformação de Modelos*

A transformação de modelos é um dos principais focos da MDA, pois apóia o desenvolvimento rápido de sistemas de *software* através da realização de transformações de modelos e também obtenção de códigos em linguagem de programação [OMG 2003Mda].

A Figura 5 ilustra a idéia das possíveis transformações de modelos com relação às suas plataformas. Um PIM pode ser transformado em novos PIMs, os quais podem possuir características especificas, mas essas não o classificam com dependência de uma plataforma, e novos PSMs, os quais devem possuir características específicas de uma determinada plataforma. Um PSM pode ser transformado em novos PSMs e novos PIMs. PSMs gerados a partir de um PSM são refinamentos do PSM original. PIMs gerados a partir de um PSM são modelos que substituem as características de uma plataforma específica para características de uma plataforma independente. Códigos em uma determinada linguagem de uma determinada plataforma podem ser obtidos a partir de PSMs. PIMs, PSMs e códigos são diferentes representações do mesmo sistema de *software*.

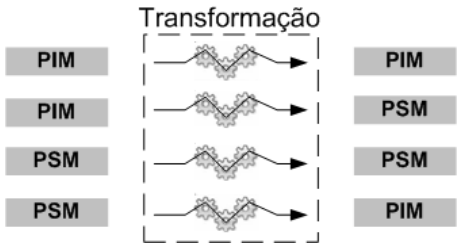


Figura 5 – Transformação de Modelos [OMG 2003Mda]

A transformação é a geração de um modelo-destino a partir de um modelo-origem. A transformação é definida por um conjunto de regras de mapeamentos que juntas descrevem como uma linguagem no modelo-origem pode ser mapeada em uma ou mais linguagens no modelo-destino [KLEPPE *et al.* 2003]. Os mapeamentos podem ser especificados de acordo com as seguinte abordagens [OMG 2003Mda]:

- **Mapeamento baseado em metamodelo**: são definidos de acordo com os tipos dos elementos do metamodelo, isto é, todos os elementos de um modelo são transformados de acordo com seu respectivo tipo de elemento no metamodelo;
- **Mapeamento baseado em instância**: são definidos com base nos elementos do modelo, isto é, os elementos de um modelo que direcionam a transformação; e
- **Mapeamento baseado em metamodelo e instância**: essa abordagem contempla as outras duas abordagens, isto é, a transformação é direcionada tanto pelos elementos de um modelo quanto pelos tipos dos elementos de seu metamodelo.

A MDA indica, em idéias gerais, quatro formas de transformação de modelos [OMG 2003Mda], são elas:

- **Transformação manual**: decisões são tomadas durante a fase de desenvolvimento de um projeto, a fim de que o mesmo proporcione um PIM e um PSM que é posteriormente mapeado para código. A MDA contribui, neste caso, estabelecendo os níveis de abstração de cada modelo;
- **Transformação de PIM preparado com perfil *(profile)*:** um PIM é modelado de acordo com um perfil UML independente de plataforma. Então, esse PIM é transformado em um PSM que usa um segundo perfil UML dependente de

plataforma. O perfil é um mecanismo que permite estender a UML para diferentes propósitos [OMG 2007Uml];

- **Transformação utilizando padrões e marcação**: os elementos de um PIM são marcados e identificados de acordo com padrões especificados no mapeamento, produzindo assim um PSM; e
- **Transformação automática**: existem contextos em que um PIM contém toda a informação necessária para a implementação, isto é, ele é computacionalmente completo e não necessita de marcações e perfis UML para gerar o código. Então, é possível realizar a transformação automática sem a necessidade de interferência do Engenheiro de *Software*.

As transformações podem ser chamadas de transformações **Modelo-Modelo** e **Modelo-Texto**. A Figura 6 ilustra os conceitos básicos da transformação **Modelo-Modelo**. Para que ocorra a transformação entre modelos é preciso definir as **Regras de Mapeamento** responsáveis pela transformação. Essas regras são baseadas no conhecimento das estruturas dos elementos do **Metamodelo Origem** e do **Metamodelo Destino**. A **Transformação** se dá de acordo com as regas de mapeamento definidas. Assim, um **Modelo Origem'** é transformado em um **Modelo Destino'**.

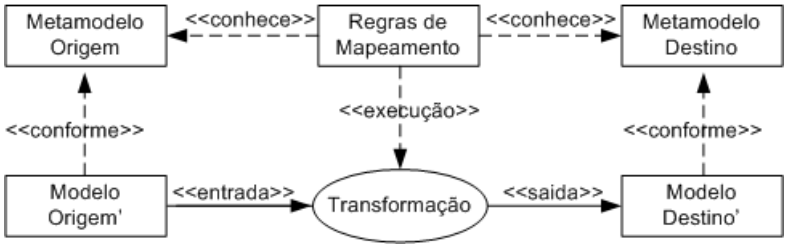


Figura 6 – Definição de Regras de Transformação [CZARNECKI e HELSEN 2006]

A especificação das regras do mapeamento pode ser descrita em linguagem declarativa, imperativa, ou uma combinação entre os dois tipos, resultando em uma linguagem híbrida. Uma linguagem declarativa descreve relacionamentos entre elementos do Metamodelo

Origem e elementos do Metamodelo Destino, e um executor dessa linguagem, também chamado de compilador, aplica algum algoritmo sobre essas relações para produzir um resultado [FOLDOC 2006]. Uma linguagem imperativa permite a especificação do comportamento de um sistema computacional. Uma implementação imperativa pode recuperar os elementos de um Modelo Origem' baseado em seu Metamodelo Origem e criar os elementos em um Modelo Destino' baseado na estrutura de seu Metamodelo Destino.

De acordo com os domínios dos metamodelos Origem e Destino, a transformação viabiliza: 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 novos modelos para novos domínios, essa transformação é chamada de *exogenous* ou *translations* [MENS e VAN GORP 2005].

O exemplo mais comum encontrado de transformação **Modelo-Texto** é a geração de códigos em linguagem de programação a partir de um modelo de entrada. Assim como a transformação Modelo-Modelo, a Modelo-Texto também pode ser especificada por meio de linguagem declarativa, imperativa ou hibrida. A diferença se dá quando essa transformação é especificada em linguagem declarativa, pois os modelos baseados no metamodelo são posteriormente serializados em forma de arquivo.

A seguir é apresentado o Modelo OO e a linguagem de modelagem UML para representar as especificações de PIMs.

2.2 Modelo Orientado a Objetos e a *Unified Modelling Language* (UML)

O modelo de objetos foi introduzido inicialmente para representar modelos de aplicações codificadas em linguagem de programação, entretanto, tem sido utilizado na representação do conhecimento, modelagem de sistemas de *software,* modelagem de banco de

dados, entre outros, firmando-se como linguagem dominante, sendo largamente utilizada pela comunidade de desenvolvimento de sistemas de *software*. Os modelos OO possuem várias vantagens [JACOBSON 1992]:

- Os sistemas de *software* possuem semântica de fácil entendimento por representar objetos do modelo como objetos do mundo real;
- Um modelo abstrato pode ser refinado para conter os conceitos pertinentes a sua implementação; e
- Os modelos OO permitem dar mais enfoque à estrutura de um sistema do que às suas funções.

Os principais conceitos da orientação a objetos usados em modelos OO são [JACOBSON 1992]:

- **Objetos**: são abstrações do mundo real. Cada objeto do mundo real pode ser representado computacionalmente por um objeto. São independentes uns dos outros, isto é, dois objetos com os mesmos atributos são objetos diferentes. São caracterizados por possuírem atributos e operações (comportamentos ou métodos);
- **Classes**: um grupo de objetos que contém as mesmas características é representado por uma classe. As classes representam como os objetos são estruturados internamente por meio de atributos e comportamentos;
- **Instâncias**: os objetos criados a partir de uma classe são chamados de instâncias. O comportamento e os atributos das instâncias são definidos pela estrutura de suas respectivas classes; e
- **Polimorfismo**: classes diferentes podem implementar a mesma operação de maneiras diferentes. Isto oferece uma dinâmica no comportamento dos

objetos que implementam suas operações. Cada objeto sabe qual é a sua classe e pode executar seus comportamentos apropriados;

Os modelos OO também possuem conceitos de relacionamentos, os quais estabelecem as conexões entre as classes. Em modelagem OO os relacionamentos mais importantes são [BOOCH *et al.* 2005]:

- **Dependência**: esse é um relacionamento semântico entre duas classes, no qual uma classe dependente pode ser afetada quando há mudanças na sua classe fornecedora. Também podem ser criadas dependências entre pacotes, anotações, etc;
- **Generalização**: é o relacionamento entre uma classe generalizada e uma classe especializada. Instâncias de uma classe especializada podem ser utilizadas em qualquer lugar em que as instâncias de uma classe generalizada são utilizadas. Uma classe especializada é subclasse de uma classe generalizada, a qual pode ser chamada de superclasse. O conceito de generalização simples é utilizado quando uma subclasse que tem apenas uma única superclasse, e generalização múltipla é quando uma subclasse têm mais do que uma superclasse;
- **Associação**: é um relacionamento estrutural entre classes que relaciona os objetos de uma classe com os objetos de outras classes. Esse relacionamento representa as classes conectadas no mesmo nível conceitual;
- **Agregação**: é um relacionamento "todo/parte" entre classes, no qual uma classe representa um elemento maior, o "todo", formado por itens menores, as "partes"; e
- **Composição**: esse também é um relacionamento "todo/parte", mas diferencia-se da agregação por vincular o tempo de vida das partes atrelado ao todo.

Com o advento da orientação a objetos, surgiram métodos alternativos de análise e projeto. Muitos desenvolvedores que utilizavam esses métodos tiveram dificuldades em encontrar uma linguagem de modelagem capaz de atender às suas necessidades. Assim, surgiu a UML, uma linguagem de modelagem que deriva de três métodos: o *Object Modeling Technique* (OMT) de Rumbaugh [RUMBAUGH *et al.* 1990], o *método* de Booch [BOOCH 1993] e o *Object-Oriented Software Engineering* (OOSE) de Jacobson [JACOBSON 1992], e que posteriormente foi aceita como um padrão do OMG para modelagem de sistemas de *software* OO [OMG 2007Uml]. A UML é muito rica, tanto no escopo, suportando a modelagem de uma grande variedade de sistemas de *software*, quanto na profundidade, suportando modelos com abstrações genéricas e modelos com detalhes de implementação.

Conforme ilustra a Figura 7, a UML é composta por vários diagramas, os quais se dividem em Diagramas Estruturais e Diagramas Comportamentais [OMG 2007Uml]. Cada diagrama possui sua representatividade para a especificação de um sistema de *software* [BOOCH *et al.* 2005].

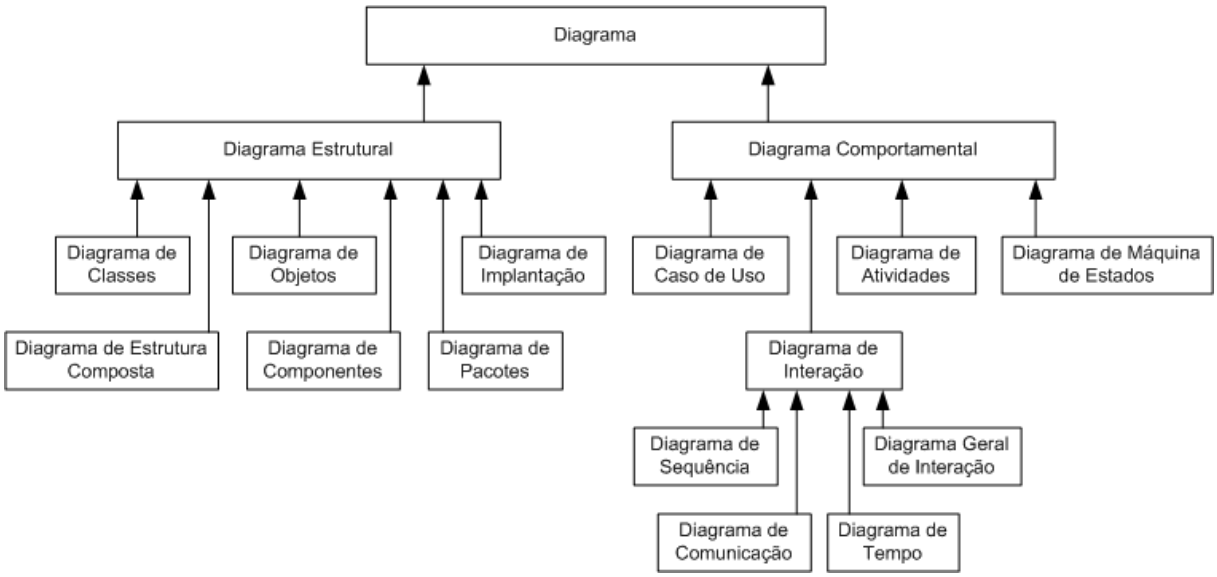


Figura 7 – Diagramas da UML [OMG 2007Uml]

O **Diagrama Estrutural** suporta a modelagem estrutural de sistemas de *software*. Os elementos desse diagrama representam conceitos significativos para um sistema e, incluem uma

abstração do mundo real representada em modelos de objetos. O Diagrama Estrutural é composto pelos seguintes diagramas:

- **Classes**: mostra as classes que compõem o sistema e as relações entre elas;
- **Objetos**: mostra os objetos (instâncias das classes) e as relações entre eles;
- **Implantação**: mostra a configuração sobre o qual o sistema é instalado;
- **Estrutura Composta**: mostra a composição dos elementos interconectados para atingir algum objetivo comum;
- **Componentes**: mostra a organização dos componentes; e
- **Diagrama de Pacotes**: mostra a organização dos pacotes do sistema.

O **Diagrama Comportamental** suporta a modelagem do comportamento dinâmico de sistemas de *software*. Esse comportamento dinâmico descreve uma série de mudanças que o sistema pode sofrer no decorrer do tempo. O Diagrama Comportamental é composto pelos seguintes diagramas:

- **Caso de Uso**: mostra como o sistema vai interagir com o usuário;
- **Atividades**: mostra como um objeto ou subsistema realiza uma operação;
- **Máquina de Estados**: mostra como um objeto ou subsistema realiza uma operação, mas o foco está nos eventos e ações que essa operação produz. Esse diagrama corresponde ao Diagrama de Estado da versão anterior da UML [OMG 2005UML];
- **Seqüência**: mostra interações entre os vários elementos do sistema;
- **Geral de Interação**: é uma variação do Diagrama de Atividades, que fornece uma visão geral do fluxo de informação;
- **Comunicação**: é uma variação do Diagrama de Seqüência, mas a ênfase é no tempo em que as interações ocorrem. Esse diagrama corresponde ao

diagrama de Colaboração da versão anterior, cujo nome também foi modificado; e

- **Tempo**: descreve a mudança no estado de uma instância e seu papel durante o tempo.

Os diferentes diagramas servem a diferentes propósitos durante o processo de desenvolvimento de sistemas de *software*. Em geral, o diagrama de classes é o diagrama mais comumente encontrado na modelagem de sistemas de *software* OO e tem um papel central por se tratar de um diagrama que suporta a modelagem da estrutura desses sistemas. Esse diagrama pode ser utilizado como PIM pra modelagem de sistemas em alto nível de abstração.

O OMG sugere que extensões a notação UML seja criadas a fim de oferecer suporte aos mais variados tipos de modelagem [OMG 2005Uml]. Assim, a UML oferece mecanismos de extensão chamados de UML *Profiles* [OMG 2007Profile], que permitem que estereótipos, valores marcados *(tagged values)* e *Object Constraint Language* (OCL) [OMG 2006Ocl] sejam utilizados para especificar as características específicas de um determinado domínio.

Apesar desse apoio à extensão, a UML pode não ser suficiente para especificar todos os conceitos necessários na modelagem, especialmente no que diz respeito aos PSMs. Assim, de acordo com as necessidades da modelagem uma nova linguagem de modelagem pode ser definida através da especificação de um novo metamodelo.

A próxima seção mostra as características do padrão SQL3 e o CWM como mecanismo para representar as especificações de PSMs do domínio de BDOR.

2.3 A *Structured Query Language* (SQL) e o *Common Warehouse Metamodel* (CWM)

A *Structured Query Language* (SQL) é uma combinação de, pelo menos duas, sub-linguagens, a *Data Definition Language* (DDL) e a *Data Manipulation Language* (DML). A

primeira dá suporte à definição e a segunda permite a manipulação dos dados [DATE 2003]. Ela foi originalmente desenvolvida pela *International Business Machines* (IBM) [IBM 2007] para processar dados contidos em bases de dados em computadores *mainframes*, e hoje é aceita por diversos gerenciadores existentes no mercado [DATE 2003].

Devido à alta adesão da linguagem, sua padronização foi estabelecida em 1986 pelo *American National Standards Institute* (ANSI) [ANSI 2005]. Desde então, o padrão SQL foi adotado formalmente pela *International Organization for Standardization* (ISO) [ISO 2005], principal controladora de padrões mundial, e pela *International Electrotechnical Commission* (IEC) [IEC 2005], a qual desenvolve padrões para a área de Tecnologia da Informação (TI). ISO e IEC juntaram seus esforços para formar um comitê a fim de especificar padrões para cada área de TI [EISENBERG e MELTON 1998]. Uma vez definidos os padrões para determinada área de TI, esses são absorvidos de maneira independente pela ISO e pelo IEC [GALLAGHER 1994].

Depois de 1986 o padrão foi revisado em:

- 1989 (SQL-89 ou SQL1);
- 1992 (SQL-92 ou SQL2) adiciona novas funcionalidades à versão SQL1;
- 1999 (SQL-99 ou SQL3) adiciona características OO à versão SQL2; e
- 2003 (SQL-2003) adiciona referências para novas partes à versão SQL3, como por exemplo, a parte que trata de dados no formato XML.

A especificação do padrão SQL define a estrutura e as operações básicas sobre os dados através da definição da sintaxe e semântica da linguagem para:

- Especificação e modificação da estrutura e integridade de dados;
- Declaração e invocação de operações sobre dados; e
- Declaração de *procedures*, que são chamadas de métodos ou rotinas que são executadas em SGBDs.

O padrão SQL é composto por dois conjuntos de características [MELTON e SIMON 2001]:

- ***Core**:* são as características mínimas que os gerenciadores devem suportar para que estejam de acordo com o padrão; e
- ***Non-Core**:* são as características adicionais, ou seja, não obrigatórias.

A orientação a objetos contida no padrão SQL3 é um conceito central para tratar tipos de dados não convencionais, proporcionando o suporte apropriado ao armazenamento de informações dos sistemas de *software* atuais [DATE 2003]. Os tipos de dados contidos no padrão SQL3 fazem parte de subseções do *Core* e podem ser divididos em três principais classes de tipos [ISO/IEC 1999, PARDEDE *et al.* 2003]:

- *Pré-Definidos;*
- *Construídos; e*
- *User-Defined Types (UDT).*

Os tipos de dados da classe de tipos **Pré-Definidos** são atômicos, os quais são tipos que não podem ser compostos por valores de outros tipos. Os tipos pré-definidos podem ser classificados em: numéricos, caracteres, *boolean*, datas *(datetime),* períodos de intervalo de tempo *(interval),* XMl, entre outros.

Os tipos de dados da classe de tipos **Construídos** podem ser atômicos ou compostos. Os tipos atômicos podem ser somente tipos *Reference* (REF), os quais são ponteiros lógicos utilizados em relacionamentos de elementos de Modelos BDOR. Os tipos compostos podem ser classificados em coleção e linha *(row).* Uma coleção é um valor composto por um ou mais elementos de um mesmo tipo de dado. Uma linha é uma seqüência "par-valor" chamada de campos. Cada campo é composto por um nome e um tipo associado a esse nome [DATE 2003].

Os tipos de dados da classe de tipos **UDT** podem ser **Distintos** *(Distinct)* ou **Estruturados**, e podem ser usados em qualquer lugar onde um tipo pré-definido é usado. Um

tipo **Distinto** é um tipo baseado em um tipo pré-definido. Um tipo **Estruturado** consiste em um número de atributos e comportamentos. Cada atributo de um tipo estruturado pode ser dos tipos de dados das classes de tipos Pré-Definidos, Construídos ou UDT. Um valor de um atributo em um tipo estruturado é dito como encapsulado, isto é, esse valor só pode ser acessado por meio de operações que o retornam.

Um tipo **Estruturado** pode ser definido como um subtipo de outro tipo estruturado, o qual é seu supertipo, esse conceito é conhecido como generalização em OO. Um subtipo herda a estrutura e o comportamento de seu supertipo e também pode possuir seus próprios atributos e comportamentos. Um supertipo pode ser usado em qualquer lugar onde um subtipo é usado, esse conceito é conhecido como substitutabilidade *(substitutability)*.

O conceito de generalização também é suportado por relações, as quais também podem ser chamadas de tabelas. Uma ou mais tabelas podem ser criadas a partir de um tipo estruturado e para cada atributo do tipo estruturado tem-se uma coluna na tabela. Essas tabelas são chamadas de tabelas com forte estrutura de tipos [DATE 2003]. Uma tabela com forte estrutura de tipos de um subtipo estruturado pode ser sub-tabela de outra tabela com forte estrutura de tipos baseada em um supertipo, isto é, uma sub-tabela de uma super-tabela baseada em tipos estruturados. O padrão SQL3 suporta ainda os conceitos de *Object Identifiers* (OID), polimorfismo [GALLAGHER 1994].

O padrão SQL3 [ISO/IEC 1999] é suportado pelo *Common Warehouse Metamodel* (CWM) [OMG 2003Cwm], o qual é um metamodelo para a especificação de modelos dos domínios DW e BI. Conforme mostra a Figura 8, o CWM pode ser representado em 5 camadas Cada uma das camadas contém um conjunto de metamodelos. Cada metamodelo é composto por pacotes que são responsáveis por cobrir um conjunto de funcionalidades específicas do domínio DW e BI [OMG 2003Cwm]. Os nomes das camadas e dos metamodelos estão em inglês devido à padronização de nomenclaturas definidas pelo OMG.

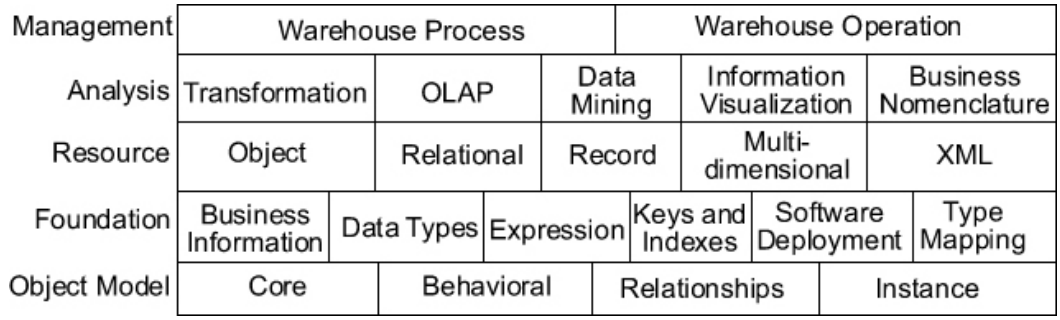


Figura 8 – Metamodelo CWM [OMG 2003Cwm]

A camada ***Object Model*** contém metamodelos para criar e descrever classes de modelos. Essa camada é considerada a camada base do CWM e é formada por um subconjunto da UML, incluindo somente as características que são necessárias para criar e descrever o CWM [POOLE 2003]. Essa camada é composta pelos seguintes metamodelos:

- ***Core:*** contêm as classes básicas usadas por todos os outros metamodelos de sua camada e das camadas superiores. Este metamodelo é a base da infra-estrutura necessária para o armazenamento de dados não orientados a objetos, como bases de dados relacionais e registros de arquivos sem excluir os conceitos da orientação a objetos.
- ***Behavioral:*** utiliza o *Core* como base e, possui modelos que descrevem o comportamento de tipos do CWM. Esse metamodelo é responsável por registrar a invocação e definição dos métodos, freqüentemente encontrados em sistemas de *software* OO e *procedures* encontradas em Sistemas de Gerenciamento de Banco de Dados (SGBD).
- ***Relationships:*** utiliza o *Core* como base e, define o relacionamento básico entre os elementos do modelo, como associações entre relações e colunas.
- ***Instance:*** utiliza o *Core* como base e, possui as classes e associações que descrevem as instâncias ou valores possíveis para as instâncias no metamodelo.

A camada ***Foundation*** depende da camada *Object Model* e estende as suas funcionalidades. Aquela é a camada mais baixa que possui metamodelos específicos ao domínio DW e BI, contendo uma coleção de metamodelos que representam conceitos e estruturas que são compartilhados por outros metamodelos das camadas superiores. A camada ***Foundation*** é composta pelos seguintes metamodelos:

- ***Data Types:*** suporta a definição de construções de modelos que podem ser utilizados para criar tipos de dados específicos conforme necessário.
- ***Type Mapping:*** suporta a criação de modelos de mapeamento entre sistemas não similares, para garantir a interoperabilidade entre eles.
- ***Keys and Indexes:*** suporta a especificação de instâncias, como por exemplo, chaves e campos de ordenação, conceitos encontrados em SGBD Relacionais (SGBDR).
- ***Business Information:*** suporta a especificação de elementos para a modelagem básica de informação de negócios pertinentes ao domínio DW e BI.
- ***Software Deployment:*** suporta a modelagem de aplicações orientadas a componentes e seu funcionamento através de uma rede de sistemas distribuídos.
- ***Expressions:*** suporta a especificação de elementos usados para construir estruturas de expressões, como árvores de expressões.

A camada ***Resource*** contém metamodelos que suportam vários aspectos para disponibilizar elementos de modelagem para o manuseio de recursos de dados. Os metamodelos desta camada utilizam as inferiores. A camada *Resource* é composta pelos seguintes metamodelos:

- ***Relational:*** é baseado no padrão SQL [ISO/IEC 1999] e usa construtores da camada *Object Model* para suportar as extensões OO do padrão SQL.
- ***Record:*** suporta a especificação de elementos do modelo com conceitos de registros e estruturas de dados.
- ***Multidimensional:*** suporta a modelagem de uma representação genérica de elementos de banco de dados multidimensionais.
- ***XML:*** suporta a especificação de elementos do modelo para representar recursos XML.
- ***Object:*** suporta as características orientadas a objetos para banco de dados que contemplam conceitos OO.

A camada ***Analisys*** contém metamodelos adicionais para modelagem de dados orientada a análise de informações do domínio DW e BI. Essa camada é composta pelos seguintes metamodelos:

- ***Data Mining:*** suporta a especificação de elementos do modelo utilizados em diversas ferramentas de mineração de dados *(Data Mining)*, como por exemplo, ferramentas de extração de padrões e ferramentas de tendência de informações baseadas em dados históricos.
- ***Business Nomenclature:*** suporta a modelagem de elementos do modelo para a construção de nomenclaturas relativas a termos do negócio pertinentes ao domínio DW e BI.
- ***Information Visualization:*** suporta a especificação de metadados relevantes para a construção de elementos para ferramentas de visualização e relatório avançados.
- ***Transformation:*** suporta a representação de elementos usados na transformação de metadados entre as fontes de dados e os seus destinos.

- *OLAP:* suporta a modelagem dos conceitos comuns dos sistemas de processamento analítico *on-line (On-Line Analytical Processing* - OLAP), como por exemplo, visão de cubos e visualização de informações em diferentes níveis de granularidade.

A camada *Management* suporta metamodelos genéricos para entendimento do ambiente e processos do domínio DW e BI. Essa camada é composta pelos seguintes metamodelos:

- *Warehouse Process:* suporta a descrição de elementos do modelo para processos envolvendo extração, transformação e carga de dados entre a fonte e destino.
- *Warehouse Operation:* suporta a especificação de elementos do modelo para definições específicas, como rotinas e tarefas agendadas.

A arquitetura do CWM implica em metamodelos com dependências mínimas entre eles a fim de prover soluções modulares à aplicações do domínio DW e BI [OMG 2003Cwm]. Devido ao metamodelo *Relational* suportar as características OO do padrão SQL3, este trabalho de pesquisa o adota com o objetivo de suportar o desenvolvimento de Modelos BDOR.

A Figura 9 ilustra as dependências desse metamodelo em relação aos metamodelos das camadas *Object Model* e *Foundation* do CWM. As camadas e os metamodelos do CWM são representados por meio de pacotes. O metamodelo *Relational,* contido na camada *Resource,* depende diretamente dos metamodelos *Core, Behavioral, Relationships* e *Instance* contidos no pacote *Object Model* e dos metamodelos *Data Type* e *Keys and Indexes* contido no pacote *Foundation.* Ele estende as funcionalidades da camada *Object Model* para descrever os conceitos OO contidos no padrão SQL3 [ISO/IEC 1999] e também estende a camada *Foundation* para suportar o uso dos conceitos de chave-primária, chave estrangeira e índices.

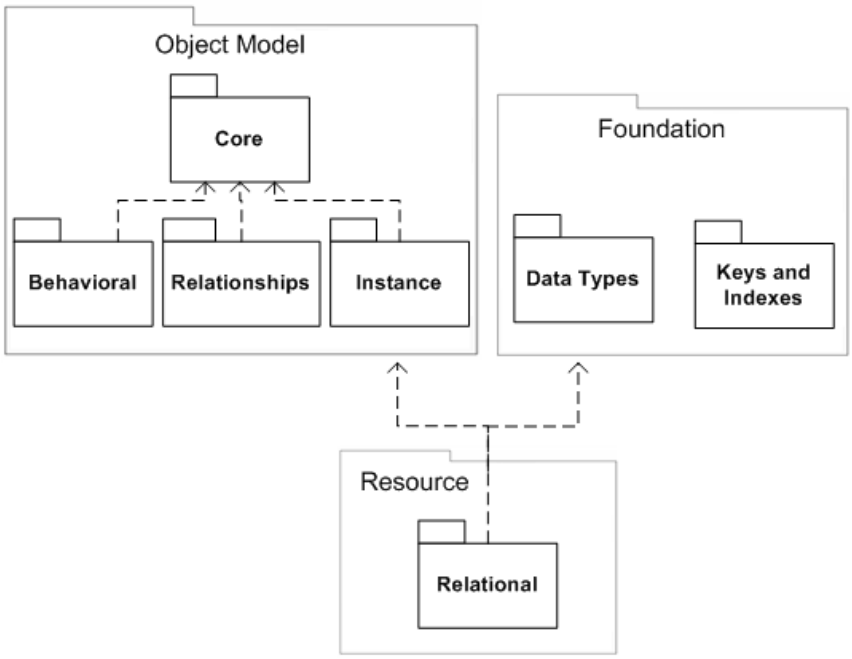


Figura 9 – Dependências do Pacote *Relational* [OMG 2003Cwm]

O **Anexo A** mostra os modelos de classes do metamodelo *Relational* e dos metamodelos das camadas *Object Model* e *Foundation* apresentados na Figura 9.

A seguir é apresentado o padrão XMI devido a sua adoção pelo OMG como mecanismo para descrever os metadados de metamodelos baseados no MOF.

2.4 XML *Metadata Interchange* (XMI)

O padrão XML *Metadata Interchange* (XMI) [OMG 2005Xmi] surgiu como um mecanismo para facilitar a interoperabilidade entre vários tipos de ferramentas de desenvolvimento e repositórios através de metadados codificados em *EXtensible Markup Language* (XML) [XML 2006]. O XML é um padrão adotado pelo *World Wide Web Consortium* (W3C) [W3C 2006], e tem o objetivo de prover, de maneira simples e independente de plataforma, a interoperabilidade através do uso de cadeias de textos dotadas de descrições dos dados contidos nele. O XMI utiliza a *Document Type Definition* (DTD), a qual é baseada em

especificações de metamodelos para validar os documentos XML. A DTD define um conjunto de regras para estruturar os elementos que podem ser descritos nos documentos XML.

O XMI descreve como gerar linguagens adequadas para descrever metamodelos baseados no MOF e como codificar os metadados desses metamodelos em XML. Assim, meta-metamodelos, metamodelos e modelos podem ser descritos em XML A especificação XMI é um conjunto de regras que normativa a geração de XML a partir do MOF [OMG 2002Mof]. Desde que o MOF foi adotado pelo OMG com o objetivo de ser utilizado para especificar metamodelos, o XMI tem sido adotado para descrever os metadados dos metamodelos baseados no MOF.

O padrão XMI oferece algumas vantagens [OMG 2005Xmi], tais como:

- **Interoperabilidade**: em um ambiente de desenvolvimento, diferentes ferramentas podem ser usadas para documentação, modelagem e implementação. Na maioria dos casos, cada ferramenta trabalha com um tipo específico de formato. Se as ferramentas produzirem e recuperarem dados no formato XMI, elas poderão trocar esses dados entre si sem necessitar de mecanismos específicos de transformação;
- **Cooperação**: a informação exportada por uma ferramenta pode ser interpretada de forma errada quando importada por outra ferramenta. Se ambas utilizam o mesmo metamodelo, toda informação transferida pode se utilizada. Assim, a utilização de repositórios e metamodelos baseados no MOF podem facilitar a cooperação entre ferramentas que utilizam o XMI como forma de troca e complementação de informação;
- **Trabalho em ambientes interconectados**: em ambiente de trabalho cooperativo, onde as estações estão conectadas por uma rede local com limite de largura de banda, frequentemente é necessária a transferência de informações entre as estações. Utilizando XMI, estas informações são representadas como seqüência de caracteres e dessa forma, podem

facilmente ser transferidas entre redes distintas ou mesmo passando por *firewalls;* e

- **Reutilização**: a diversidade de ferramentas utilizadas pelo Engenheiro de *Software* pode variar consideravelmente de acordo com o cliente e os projetos desenvolvidos. Através do uso do padrão XMI, os projetos desenvolvidos para um determinado cliente podem ser facilmente utilizados em outro cliente com um conjunto de ferramentas específicas desse cliente.

Assim, as ferramentas podem ser escolhidas de acordo com as vantagens providas por cada uma, desconsiderando o fato de falta de interoperabilidade e cooperação de informações entre as ferramentas escolhidas.

A seguir são apresentadas ferramentas CASE com ênfase na MVCASE devido sua adoção nesta pesquisa.

2.5 Ferramenta MVCASE

Existem diferentes tipos de *Computer-Aided Software Engineering* (CASE) e essas podem compreender desde a fase de identificação de requisitos até as fases de implementação e testes. As CASEs podem auxiliar o Engenheiro de *Software* na criação de modelos de diversos aspectos de um sistema de *software*, como por exemplo, os modelos comportamento, de arquitetura e estrutura de sistemas de *software* e também de banco de dados. Segundo [WERNECK 2006], a taxonomia de CASE pode ser classificada em: Engenharia da Informação; Gerenciamento e Modelagem de Processo; Planejamento de Projeto; Análise de Risco; Gerenciamento de Projeto; Auditoria de Requisitos *(Tracing);* Gerenciamento e Métricas; Documentação; Sistemas de *Software;* Controle da Qualidade; Gerenciamento de Banco de Dados; Gerenciamento de Configuração de *Software;* Análise Estática; Análise Dinâmica; Gerenciamento de Testes; Testes Cliente-Servidor; e Reengenharia.

As atividades do processo de desenvolvimento de sistemas de *software* podem ser encontradas em ferramentas distintas. Cada ferramenta pode gerar seu específico metadado de acordo com seu específico metamodelo, assim metamodelos distintos podem dificultar a reutilização e integração de metadados entre as ferramentas. Ferramentas ainda podem necessitar de mecanismos intermediários para validar os metadados de outras ferramentas a fim de proporcionar a compatibilidade entre distintos metamodelos. Esses mecanismos intermediários são chamados de *middewares,* os quais podem tratar a complexidade do processo de integração de informações entre ferramentas distintas. As CASEs podem oferecer a completude do processo de desenvolvimento de sistemas de *software* através do intercâmbio de informações entre ferramentas distintas [PRESSMAN 2004].

As ferramentas CASEs disponíveis até a década de 90 suportavam técnicas da Análise Estruturada (AE) [GANE 1979]. A partir dessa década, grande parte dessas ferramentas passou a utilizar a UML como linguagem de desenvolvimento de sistemas de *software* OO. Entre as diversas CASEs disponíveis podem-se citar:

- **CaseStudio**[1]: disponibiliza o Diagrama Entidade-Relacionamento (DER) [CHEN 1976] e o Diagrama de Fluxo de Dados (DFD) da AE. Suporta a modelagem de seus diagramas de forma visual e interativa, e ainda possibilita a geração de códigos SQL;
- **Dr.Case**[2]: possibilita modelagem segundo o DER e DFD de forma gráfica com visualização da visão lógica e física;
- **Visio Enterprise Architect**[3]: disponibiliza uma grande quantidade de diagramas e gráficos. Suporta a modelagem baseada em diagramas da AE e da UML;

[1] http://www.casestudio.com
[2] http://www.squadra.com.br
[3] http://www.microsoft.com/office/visio

- **Together Enterprise Architect**[4]: é uma ferramenta de projeto de *software* que auxilia o Engenheiro de *Software* na geração de código, trabalho cooperativo entre outras atividades do processo de desenvolvimento de sistemas de *software*. Suporta a modelagem segundo a UML e DER, além de muitos outros diagramas específicos da ferramenta. Possui funcionalidades para suportar a geração de códigos SQL. Também suporta engenharia reversa de banco de dados;
- **IBM Rational Rose**[5]: inicialmente concebida pelos criadores da UML. Suporta recursos de modelagem na notação UML, configuração de requisitos, trabalho cooperativo, engenharia reversa, entre outros; e
- **ArgoUML**[6]: suporta a modelagem baseada em diagramas UML e possui mecanismos para gerar códigos e engenharia reversa. A versão comercializada da ferramenta é a GentleWare Poseidon UML[7].

Além dessas ferramentas têm-se outras, como a MVCASE, a qual é utilizada neste trabalho de pesquisa. A ferramenta MVCASE vem sendo desenvolvida desde 1997 por alunos do Grupo de Engenharia de Software (GOES) dos programas de pós-graduação e graduação do Departamento de Computação da Universidade Federal de São Carlos (UFSCar). Ela vem auxiliando no andamento de diversos trabalhos de pesquisa, conforme pode ser visto em [PAIVA *et al.* 2006], [GARCIA *et al.* 2004] [LUCRÉDIO 2005], [MORAES 2004], [ASSÁO 2003] [LUCRÉDIO *et al.* 2003], [LUCRÉDIO *et al.* 2003a], [ALMEIDA *et al* 2002], [ALMEIDA *et al* 2002a], [NOVAIS 2002], [PRADO e LUCRÉDIO 2001], [PRADO e LUCRÉDIO 2000] e [BARRÉRE 1999].

[4] http://www.borland.com/us/products/together
[5] http://www.ibm.com/rational
[6] http://argouml.tigris.org
[7] http://gentleware.com

A MVCASE é uma CASE que suporta a modelagem de sistemas de *software* de acordo com a notação UML através de suas interfaces gráficas que facilitam o seu entendimento e utilização de seus diagramas em diferentes níveis de abstração.

O Engenheiro de *Software* é auxiliado de forma textual e gráfica pela MVCASE em atividades relativas à análise, projeto, implementação e implantação de sistemas de *software* OO. O sistema modelado com a MVCASE pode ser especificado segundo quatro visões: Visão de Casos de Uso; Visão Lógica; Visão de Componentes; Visão "*Deployment*" ou Implantação.

A visão de Casos de Uso mostra uma visão externa da interação do usuário com as ações que esse pode executar no sistema e a uma visão do comportamento desse sistema diante dessa interação. Essa visão utiliza-se de diferentes técnicas de representação dos casos de uso, destacando-se o diagrama de casos de uso e o diagrama de seqüência, ambos da UML [BOOCH *et al.* 2005].

A Visão Lógica oferece uma visão estrutural das classes e relacionamentos do sistema por meio do diagrama de classes, e uma visão dinâmica por meio do diagrama de estados [BOOCH *et al.* 2005].

A Visão de Componentes fornece uma visão estática que mostra os componentes e seus relacionamentos por meio do diagrama de componentes [BOOCH *et al.* 2005].

A Visão de Implantação oferece uma visão estática do ambiente ao qual o sistema é implantado, mostrando um conjunto de elementos processadores e seus relacionamentos [BOOCH *et al.* 2005].

A Figura 10 mostra o enquadramento da MVCASE em relação à classificação de CASEs proposta por Fuggetta [FUGGETTA 1993]. A MVCASE apóia o planejamento e modelagem do negócio, análise e projeto e também a programação, servindo para criar modelos complexos de negócio a partir dos requisitos do sistema e permitindo o refinamento manual desses modelos a fim de direcioná-los para modelos de projeto mais próximos da linguagem de programação e da plataforma tecnológica escolhida pelo Engenheiro de *Software* [LUCRÉDIO 2005].

Classes de CASE	
Planejamento e modelagem de negócio	MVCASE
Análise e projeto	
Programação	
Desenvolvimento de interface com o usuário	
Verificação e validação	
Manutenção e engenharia reversa	
Gerenciamento de configuração	
Gerenciamento de projetos	

Figura 10 – Classificação da MVCASE [LUCRÉDIO 2005]

A arquitetura da MVCASE é baseada em um repositório que armazena os metadados de metamodelos baseados no MOF em XMI. Visto que o XMI é utilizado como mecanismo facilitador de transferência de informações entre ferramentas de desenvolvimento de sistemas de *software,* uma ferramenta que o utiliza deve disponibilizar interfaces de acesso as informações contidas nele, além de métodos para escrever e ler XMI. O MOF contém a definição dessas interfaces em *Interface Definition Language* (IDL) [OMG 2002Mof], a qual pode ser mapeada para qualquer linguagem. A MVCASE usa a *Java Metadata Interface* (JMI) [DIRCKZE 2002] que é mapeada da IDL para linguagem Java. Assim, ferramentas escritas em Java, como é o caso da MVCASE, podem acessar os dados e metadados do MOF de maneira padronizada.

A especificação JMI [DIRCKZE 2002] permite a interoperabilidade em nível de linguagem de programação para a manipulação de modelos e metamodelos baseados no MOF. Através dessa especificação, é possível, por exemplo, navegar e manipular os elementos de Modelos UML de forma programática na linguagem Java [MATULA 2006]. Essa especificação define regras padronizadas que permitem a geração de interfaces escritas em Java para a manipulação de qualquer modelo descrito em qualquer linguagem baseada no MOF. Isto permite o desenvolvimento de ferramentas de modelagem capazes de manipular os modelos por meio da interação da linguagem Java.

A Figura 11 ilustra a arquitetura da MVCASE. O *Metadata Repository* (MDR) [MATULA 2006] é o repositório de metadados fornecido gratuitamente pela *NetBeans Community*[8], esse repositório implementa os padrões MOF, JMI e XMI. 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 carregados no MDR. O XMI implementado no MDR possibilita a descrição dos metadados dos metamodelos em documentos XML.

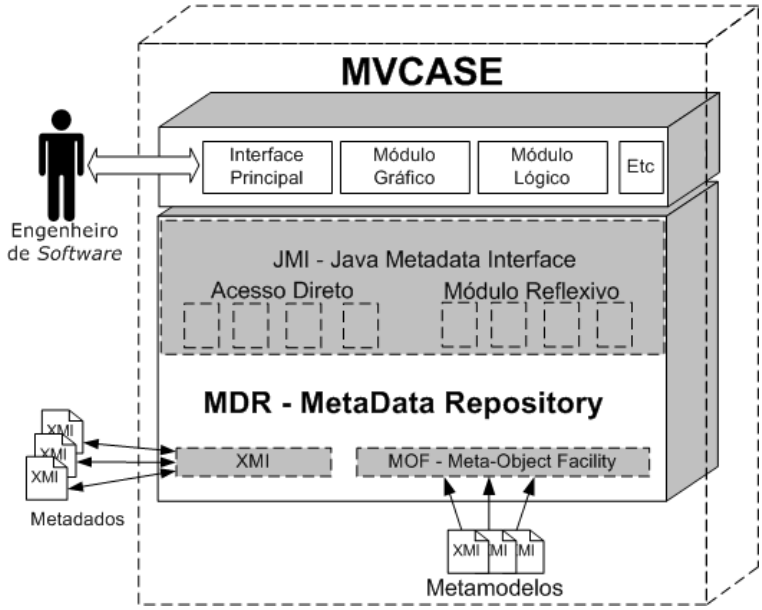


Figura 11 – Arquitetura da MVCASE [LUCRÉDIO 2005]

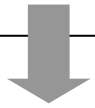
O XMI de cada metamodelo é armazenado no MDR, o qual é responsável por ler e escrever os metadados dos modelos, além de gerar JMI correspondente a cada metamodelo carregado nele. As diferentes funcionalidades da MVCASE (**Interface Principal**, **Módulo Gráfico**, **Módulo Lógico**, **etc.**) acessam os dados através das interfaces padronizadas pelo JMI (**Acesso Direto e Módulo Reflexivo**). O **Módulo Reflexivo** permite acesso aos dados por meio de métodos reflexivos, que fazem introspecção sobre os objetos armazenados no MDR a fim de descobrir seus métodos e atributos. O **Acesso Direto** é especificado por meio do acesso às interfaces geradas pelo MDR a partir de cada metamodelo XMI carregado no repositório.

[8] http://mdr.netbeans.org

A Figura 12 ilustra uma interface e seu correspondente metamodelo em XMI. Uma classe de nome *Table* e seus atributos estão especificados no metamodelo CWM. Para cada classe do metamodelo, o MDR gera uma interface em linguagem Java e métodos específicos para manipular suas instâncias. A implementação dessa interface é feita pelo MDR no momento em que o metamodelo XMI é carregado no MDR. Assim, aplicações escritas em Java podem acessar os dados armazenados no MDR por meio de acesso direto, através das interfaces especificadas no JMI, ou por meio do módulo reflexivo do JMI, fazendo introspecção aos objetos armazenados no MDR.

Trecho da Especificação do metamodelo CWM

```
<Model:Class xmi.id = 'a364CDE8501D3' name = 'Table'
annotation = 'A
      materialized NamedColumnSet.' isRoot = 'false' isLeaf =
'false'
      isAbstract = 'false' visibility = 'public_vis'>
      <Model:Namespace.contents>
        <Model:Attribute xmi.id = 'a377A994F0294' name =
'isTemporary'>
            …
        </Model:Attribute>
        <Model:Attribute xmi.id = 'a379CE39B024A' name =
'isSystem'>
            …
        </Model:Attribute>
      </Model:Namespace.contents>
</Model:Class>
…
```



Interface JMI correspondente

```
public interface TableClass extends javax.jmi.reflect.RefClass
{
      public Table createTable();
      public Table createTable(java.lang.String name,
            core.VisibilityKind visibility, boolean
isTemporary,
            Boolean isSystem, boolean isAbstract
                  java.lang.String temporaryScope,);
}
```

Figura 12 – Especificação XMI e correspondente JMI

A ferramenta MVCASE é totalmente implementada em Java e possui sua versão atual disponível para uso da comunidade de *software* [MVCASE 2005]. Dado o conhecimento sobre sua

arquitetura e códigos de implementação, a MVCASE foi utilizada como principal mecanismo para realizar as transformações de Modelo OO para Modelos BDOR e Códigos SQL.

A seguir são apresentados os trabalhos correlatos.

2.6 Trabalhos Correlatos

Foram encontradas diversas implementações existentes que se aproximam da abordagem desta pesquisa, dentre elas: o *framework* AndroMDA [ANDROMDA 2006]; a ferramenta Rational Rose Data Modeler da IBM [IBM 2006]; o protótipo Er2Cwm [FARPINYO e SENIVONGSE 2003] e um trabalho de pesquisa descrito em [VARA *et al.* 2007].

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. No caso do protótipo proposto nesta pesquisa, a transformação de Modelos UML para Modelos BDOR se dá de forma semi-automática, sem a necessidade de configurações extras.

A ferramenta IBM Rational Rose Data Modeler permite a transformação entre modelos de objetos, modelos de dados e geração de códigos SQL. A IBM Rational Rose usa o MOF e o XMI como parte do *framework* de integração de ferramentas dirigidas por modelos, além do suporte ao *Eclipse Modeling Framework* (EMF). O EMF é uma implementação Java de um subconjunto do MOF [EMF 2006] que gerencia os metamodelos. Essa ferramenta dá suporte a MDA com enfoque no desenvolvimento de sistemas de *software* para Banco de Dados, porém não utiliza o metamodelo CWM, o qual é um padrão proposto para facilitar o intercâmbio e a integração de informações entre ferramentas distintas.

Em [FARPINYO e SENIVONGSE 2003], os autores apresentam um protótipo baseado no metamodelo CWM, o ER2CWM. Esse protótipo permite a criação e manipulação de forma gráfica de modelos de dados relacionais que utilizam o DER. Porém, esse protótipo não suporta a modelagem de tipos de dados complexos e também não utiliza a notação gráfica UML.

O trabalho descrito em [VARA *et al.* 2007] apresentada uma proposta de metodologia de desenvolvimento de sistemas de *software* dirigido por modelos para o domínio BDOR. Os modelos são representados em UML de acordo com perfis (*profiles*) que suportam a semântica dos Modelos BDOR para representar as características OO desses modelos. A transformação entre PIM e PSM é especificada por meio de regras de mapeamentos representadas diagramaticamente. Porém, dentro da metodologia proposta, é implementado um metamodelo próprio para representar a estrutura dos modelos BDOR, isto é, a metodologia não utiliza o CWM como metamodelo para suportar os modelos BDOR.

A seguir são apresentadas as considerações deste capítulo.

2.7 Considerações

Considerando os conceitos e idéias apresentadas neste capítulo, algumas questões podem ser consideradas para o desenvolvimento desta pesquisa que objetiva transformar Modelos OO em Modelos BDOR e conseqüente obtenção dos respectivos Códigos SQL3:

- Como auxiliar o Engenheiro de *Software* no processo de desenvolvimento de sistemas de *software* do domínio BDOR?
- Como reaproveitar modelos abstratos que caracterizam independência de plataforma para que possam gerar novos modelos para uma determinada plataforma de BDOR?

A MDA é uma abordagem para o desenvolvimento dirigido por modelos que consiste da obtenção de modelos dependentes de plataforma e códigos a partir de um modelo abstrato.

Essa abordagem é baseada em padrões definidos pelo OMG, entre os quais podem-se citar UML, que suporta os modelos OO, CWM, que suporta os modelos BDOR, e o XMI, utilizado como mecanismo de intercâmbio de informações dos modelos em documentos XML.

Neste capítulo, também foram abordadas as ferramentas CASEs, as quais podem proporcionar o auxílio ao Engenheiro de *Software* para executar as tarefas em um processo de desenvolvimento de sistemas de *software*

Assim, os conceitos discutidos neste capítulo servem como base para o desenvolvimento deste trabalho de pesquisa, o qual é detalhado no próximo capítulo.

Capítulo 3

Transformação de Modelos Orientados a Objetos em Modelos de Banco de Dados Objeto Relacional

Nos capítulos anteriores foram apresentados os principais conceitos envolvidos nesta pesquisa, que fornecem o embasamento teórico para o desenvolvimento de sistemas de *software* do domínio BDOR utilizando a abordagem do desenvolvimento dirigido por modelos do OMG, a MDA.

Neste capítulo é apresentada a transformação de Modelos OO em Modelos BDOR e Códigos SQL3. Dos estudos realizados, identificaram-se os principais requisitos que orientaram essa transformação. Para facilitar o entendimento, esses requisitos foram agrupados em: **Domínio do Problema** e **Transformação de Modelos**.

Com relação ao **Domínio do Problema** consideraram-se os seguintes requisitos:

- **Usar linguagens padronizadas**: linguagens padronizadas é um fator considerado neste trabalho devido a maturidade que um padrão está relacionado, sua utilização em inúmeros cenários e a facilidade de entendimento de um problema pelo uso comum dessas linguagens entre equipes distintas de desenvolvimento; e

- **Ser genérico**: possibilitar uma possível extensão das linguagens utilizadas sem comprometer a ferramenta como um todo.

Com relação à **Transformação de Modelos** consideram-se:

- **Transformar Modelo OO em Modelo BDOR**: possibilitar a obtenção de Modelos BDOR a partir de Modelos OO;
- **Transformar Modelo BDOR em Código SQL3**: possibilitar a obtenção de códigos SQL3 a partir de modelos BDOR; e
- **Transformar de forma mais automática possível**: viabilizar as transformações de maneira a não necessitar o conhecimento do domínio BDOR pelo Engenheiro de *Software*.

Outro ponto considerado nesta pesquisa foi possibilitar o reuso dos artefatos de *software* responsáveis pelas transformações de modelos. Assim, outras ferramentas de modelagem de sistemas de *software* podem reutilizar esses artefatos para proporcionar transformação de Modelo OO para o Modelo BDOR e Códigos SQL em seus próprios ambientes de desenvolvimento.

A Figura 13 ilustra a visão geral da arquitetura da abordagem, na qual o Engenheiro de *Software* se utiliza da ferramenta MVCASE para construir Modelos OO de acordo com as seguintes visões disponíveis pela ferramenta: Modelos de Casos de Uso, Modelos de Classes, Modelo de Seqüência e Modelos de Componentes. O Engenheiro de *Software* especifica no modelo de classes a estrutura do sistema modelado, o qual deve ter suas funcionalidades preservadas quando suportadas por uma determinada plataforma. Assim, um modelo de classes pode ser utilizado para representar um PIM, este modelo é chamado a partir deste ponto simplesmente de **Modelo UML**.

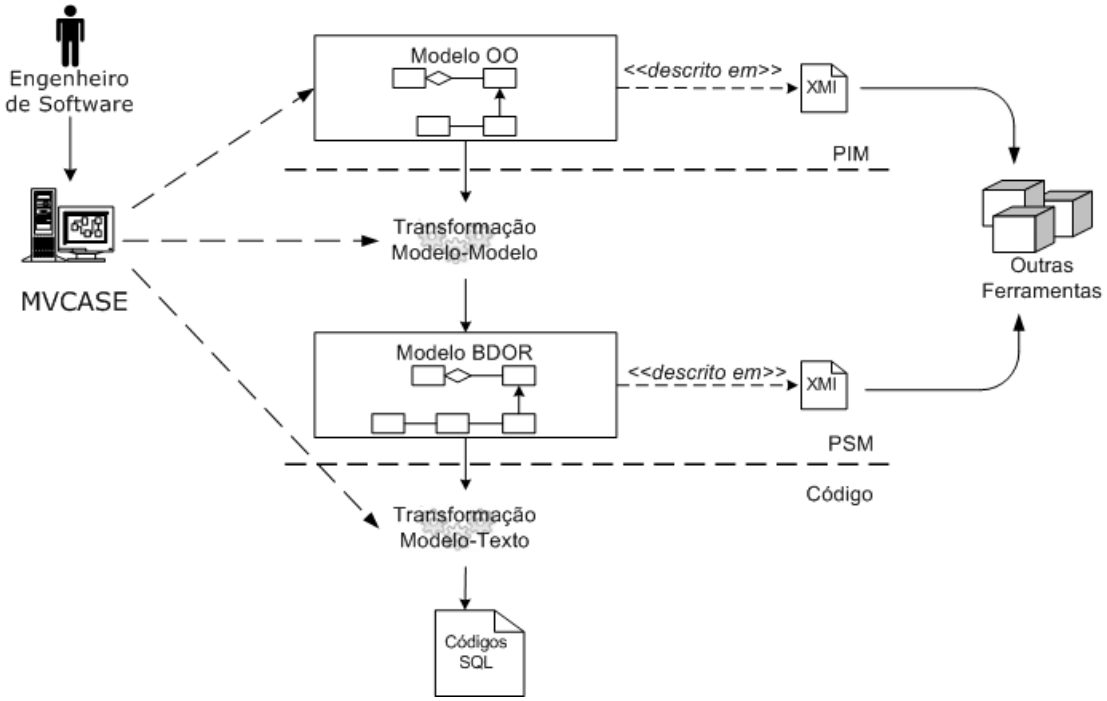


Figura 13 – Visão Geral da Arquitetura da Abordagem

Na fase de projeto são acrescentadas características não funcionais ao Modelo UML, isto é, características que dependem de uma determinada plataforma. Assim, o Engenheiro de *Software* utiliza a **Transformação Modelo-Modelo** para gerar um **Modelo BDOR** a partir de um Modelo UML. Um Modelo BDOR que contêm características do domínio BDOR pode ser utilizado para representar um PSM. Finalmente, esse Engenheiro de *Software* utiliza a **Transformação Modelo-Texto** para gerar os Códigos SQL3 a partir de um Modelo BDOR.

Os **Modelos UML e BDOR** são descritos em XMI, os quais podem ser utilizados por outras ferramentas para diferentes finalidades. Os metadados desses modelos são padronizados de acordo com a linguagem de seus correspondentes metamodelos, ou seja, os metadados do **Modelo UML** são descritos de acordo com o metamodelo UML e os metadados do **Modelo BDOR** são descritos de acordo com o metamodelo CWM.

Baseado nessas idéias e nos requisitos identificados, a seguir é apresentada a transformação de Modelos OO em Modelos BDOR e Códigos SQL3.

3.1 Transformação de Modelos OO em Modelos BDOR

Com o objetivo de satisfazer os requisitos citados anteriormente, a Figura 14 mostra as três etapas do processo de desenvolvimento para estender a MVCASE com recursos para suportar a transformação de Modelos OO em Modelos BDOR e a geração de Códigos SQL3.

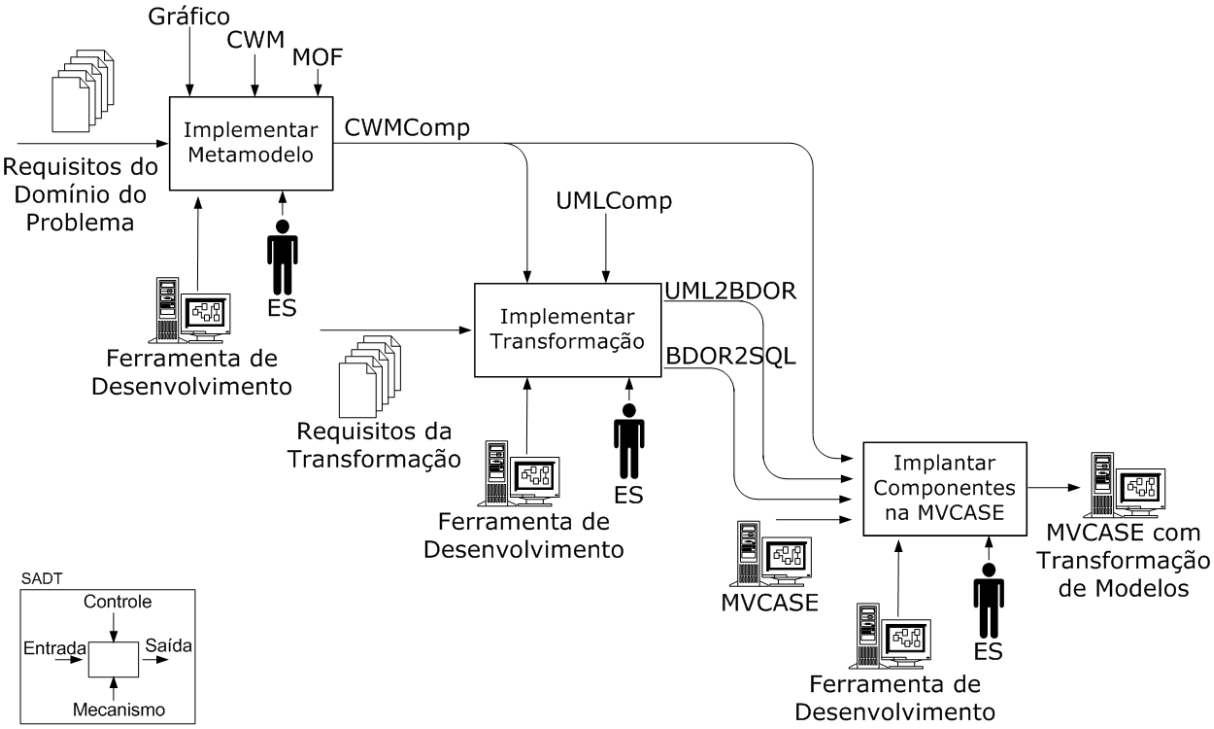


Figura 14 – Desenvolvimento da Solução

Na primeira etapa, **Implementar Metamodelo**, o **Engenheiro de *Software***, com base nos requisitos do **Domínio do Problema** e com o auxílio de uma **Ferramenta de Desenvolvimento** de modelos de sistemas de *software*, implementa um metamodelo de acordo com os metamodelos **Gráfico** (ver **Anexo B** para maiores informações sobre o metamodelo Gráfico), **CWM** e **MOF**. A saída desta etapa é um artefato de *software* que recebe o nome de **CWMComp**.

A segunda etapa, **Implementar Transformação**, compreende a implementação das transformações de modo a satisfazer os **Requisitos da Transformação de Modelos** de acordo com o **CWMComp** e o **UMLComp**, o qual é a implementação do metamodelo UML [OMG

2005Uml]. Essa etapa tem como saída os componentes responsáveis pelas transformações de Modelos UML em Modelos BDOR e em Códigos SQL3. Esses componentes de transformação são chamados de **UML2BDRO** e **BDRO2SQL**.

A terceira etapa, **Implantar Componentes na MVCASE**, compreende a integração dos componentes produzidos na primeira e segunda etapa do desenvolvimento à ferramenta MVCASE. Assim, a extensão da ferramenta disponibiliza mecanismos para auxiliar o Engenheiro de *Software* na transformação de Modelos UML em Modelos BDOR e geração de Códigos SQL3.

Seguem-se as apresentações mais detalhadas de cada etapa realizada para estender a MVCASE, tornando-a adequada para suportar o desenvolvimento de sistemas de *software* por meio da transformação de Modelos UML em Modelos BDOR e Códigos SQL3.

3.1.1 Implementar Metamodelo

Na abordagem MDA os Modelos UML e BDOR são baseados em metamodelos que são baseados no MOF, então se utilizou o CWM para possibilitar a modelagem das características semânticas dos Modelos BDOR de acordo com os conceitos OO do padrão SQL3, e se utilizou um metamodelo Gráfico [LUCRÉDIO 2005] para suportar as características gráficas dos modelos na MVCASE, como por exemplo, posição em tela, relacionamento entre os elementos gráficos, entre outras, são responsabilidade do metamodelo gráfico.

Com base na especificação CWM e no metamodelo Gráfico, a etapa **Implementar Metamodelo** considera como entrada os **Requisitos do Domínio do Problema**, os quais nortearam o desenvolvimento do artefato de *software* resultante desta etapa. Assim, com o auxílio da MVCASE ou de outra ferramenta CASE que suporte o padrão XMI, esses metamodelos são desenvolvidos e em seguida, com o apoio da ferramenta UML2MOF [UML2MOF 2006], são transformados em metamodelos de acordo com MOF. Esses metamodelos são submetidos à

ferramenta NetBeans[9] versão 3.5, a qual possui um módulo responsável por gerar as interfaces de acesso aos metamodelos por meio da linguagem Java. Portanto, essa etapa gerou o **CWMComp**, o qual representa os artefatos dos metamodelos Gráfico e CWM e suas respectivas interfaces. Esse artefato é integrado à MVCASE para possibilitar o desenvolvimento de Modelos BDOR.

A notação UML foi estendida com o objetivo de representar graficamente os aspectos semânticos dos Modelos BDOR. Os Modelos BDOR são instanciados por meio do CWM e representados graficamente por meio de modelos de classes da UML. Assim, têm-se um conjunto de extensões definidas, por meio de estereótipos, conforme ilustrado na Tabela 1.

Tabela 1 – Estereótipos

Estereótipo	Elemento do metamodelo UML
<<ObjectType>>	Class
<<ObjectTable>>	Class
<<NestedTable>>	Class

Os estereótipos são utilizados conforme a necessidade semântica de se representar elementos de um Modelo BDOR. O *ObjectType* é utilizado em elementos que representam um tipo estruturado, o *ObjectTable* é utilizado em elementos que representam uma tabela com forte estrutura de tipos e o *NestedTable* é utilizado em elementos que representam uma tabela aninhada [ZENDULKA 2005]. Elementos da notação gráfica do diagrama de classes da UML também foram adaptados para suportar a semântica dos Modelos BDOR, estes podem ser vistos na Tabela 2.

Tabela 2 – Elementos Gráficos da UML [ZENDULKA 2005]

Elemento gráfico da UML	Semântica BDOR
Dependência	Indica uma tabela com forte estrutura de tipos dependendo de um tipo estruturado
Associação	Indica um relacionamento de referência (REF) entre tipos estruturados
Agregação	Idem Associação
Composição	Indica um tipo estruturado que contém uma tabela aninhada
Generalização	Indica uma especialização entre tipos estruturados

[9] http://www.netbeans.org

Assim, foram utilizadas linguagens padronizadas para suportar Modelos BDOR de acordo com o padrão SQL3, foi utilizada a representação gráfica do diagrama de classes da UML para representar os Modelos BDOR e ainda manteve-se a generalidade da ferramenta MVCASE pelo uso de metamodelos baseados no padrão XMI.

3.1.2 Implementar Transformação

Com base nos Requisitos da Transformação, a etapa **Implementar Transformação** pode ser subdividida em duas novas etapas: Implementar Transformação de Modelo UML em Modelo BDOR e Implementar Transformação de Modelo BDOR em Código SQL3. Cada nova etapa gerou um artefato de *software,* chamados de **UML2BDOR**, responsável pela transformação de Modelo UML em Modelo BDOR, e o **BDOR2SQL**, responsável pela transformação Modelo BDOR em Códigos SQL3.

Dentre as várias formas de realizar a transformação de um Modelo UML em um Modelo BDOR, tem-se a forma automática e guiada pelos tipos dos elementos dos metamodelos, na qual, as regras de mapeamento que fazem a transformação baseiam-se na estrutura dos elementos dos metamodelos para transformar as instâncias de cada um dos elementos do UMLComp e do CWMComp.

Para facilitar o entendimento sobre o conceito de instâncias dos elementos do UMLComp e do CWMComp, a Figura 15 ilustra um Modelo UML que contém uma classe *Pessoa* e um atributo *nome* do tipo *String,* cujos elementos instanciados do UMLComp são *Class, Attribute* e *DataType.* A figura ainda mostra um Modelo BDOR análogo ao Modelo UML, cujos elementos instanciados do CWMComp são *SQLStructuredType, Collumn, SQLSimpleType, Stereotype* e *Dependency.*

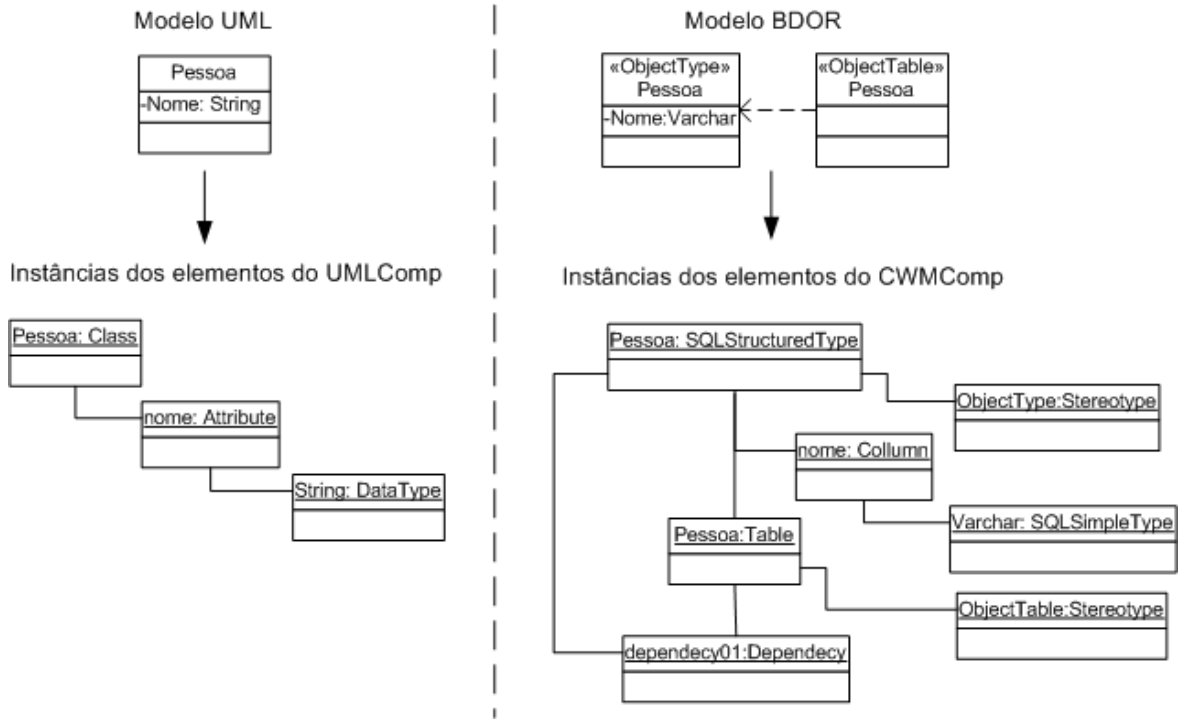


Figura 15 – Modelos e Instâncias dos Elementos dos Metamodelos

Considerando o reuso dos artefatos de transformação em outras ferramentas de desenvolvimento de sistemas de *software,* a transformação de Modelos UML em Modelos BDOR foi implementada como componente de *software*. Esse componente, chamado de **UML2BDOR,** foi construído conforme as regras de mapeamento apresentadas na Tabela 3.

Tabela 3 – Regras de mapeamento de Modelo UML para Modelos BDOR

Elementos do UMLComp	Elementos do CWMComp
DataType	<cwmDataTypeList> <cwmDataTypeList> = SqlSimpleType | SqlDistinctType | SqlStructuredType
Class e Attribute	SqlStructuredType, Column, Table e Dependency
Association	Association
Generalization	Generalization

Conforme a Tabela 3, uma instância do elemento *DataType* do metamodelo UML é transformada em uma instância do elemento *SqlSimpleType* ou do elemento *SqlDistinctType* ou do elemento *SqlStructuredType* do CWMComp. Uma instância do elemento *Class* do UMLComp e suas instâncias do elemento *Attribute* relacionadas são transformadas em instâncias dos

elementos *SqlStructuredType, Column, Table* e *Dependency* do CWMComp. Uma instância do elemento *Association* do UMLComp é transformada em uma ou duas instâncias do elemento *Association* do CWMComp dependendo da multiplicidade relacionada à instância do elemento do UMLComp. Uma instância do elemento *Generalization* do UMLComp é transformada em uma instância do elemento *Generalization* do CWMComp.

A Figura 16 ilustra num Modelo de Casos de Uso o comportamento do Componente **UML2BDOR**, que tem um Modelo UML como entrada e um Modelo BDOR como saída. Esse caso de uso representa o processo de transformação na seguinte seqüência: **Transformar Tipo de Dado**, **Transformar Classe**, **Transformar Atributo**, **Transformar Associação** e **Transformar Generalização**. Esse particionamento do caso de uso Transformar Modelo UML para Modelo BDOR foi baseado nas regras de mapeamento que fazem transformação de instâncias dos elementos do UMLComp em instâncias dos elementos do CWMComp.

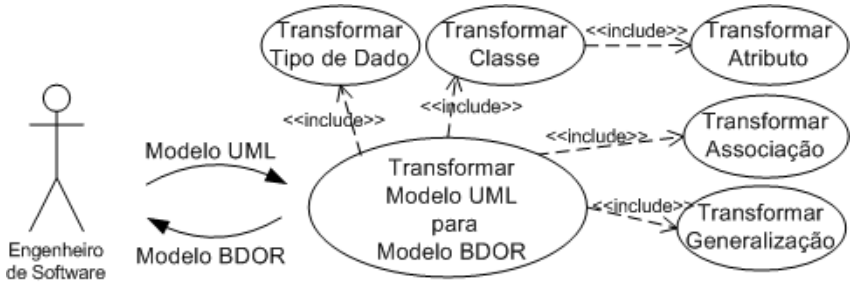


Figura 16 – Modelo de Casos de Uso: Transformar Modelo UML para Modelo BDOR

A seguir são apresentadas as especificações desses casos de uso através de modelos de seqüência da UML. A Figura 17 ilustra a seqüência do comportamento normal do caso de uso *Transformar Tipo de Dado*.

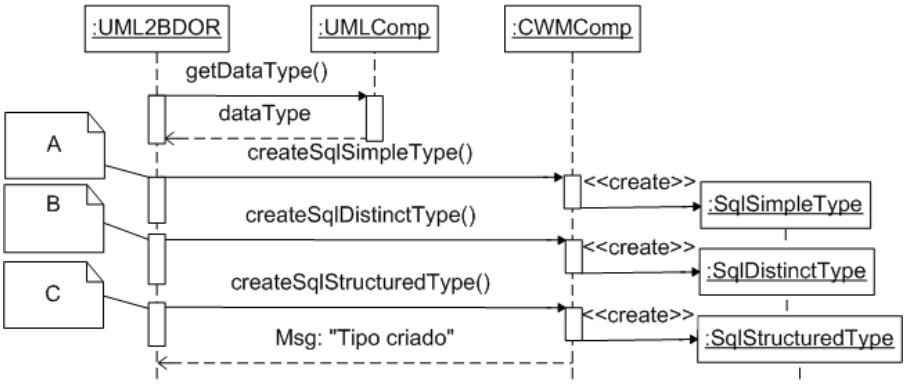


Figura 17 – Modelo de Seqüência Transformar Tipo de Dado

Uma instância que representa um tipo de dado no Modelo UML é transformada em uma instância que representa um tipo de dado pré-definido, construído ou UDT no Modelo BDOR. Os tipos construídos são tipos utilizados em campos que fazem relacionamentos (REF) com outros tipos estruturados, logo não se aplicam a este caso particular. Assim, o UML2BDOR recupera as instâncias do tipo *DataType* do UMLComp e as transforma em uma instância do tipo *SqlSimpleType* ou *SqlDistinctType* ou *SqlStructuredType* do CWMComp, desde que sejam cumpridas as restrições A ou B ou C*:*

- **Restrição A**: para cada instância do tipo *DataType* que não contenha relação nenhuma com instâncias do tipo *Attribute,* ambas do UMLComp, é criada uma instância do tipo *SqlSimpleType* do CWMComp.
- **Restrição B**: para cada instância do tipo *DataType* que contenha uma relação com uma instância do tipo *Attribute,* ambas do UMLComp, é criada uma instância do tipo *SqlDistinctType* do CWMComp.
- **Restrição C**: para cada instância recuperada do UMLComp do tipo *DataType* que contenha uma relação com mais do que uma instância do tipo *Attribute,* ou que contenha uma ou mais relações com instâncias do tipo *Attribute* transformadas em instâncias do tipo *SqlDistinctType* ou *SqlStructuredType* do CWMComp, é criada uma instância do tipo *SqlStructuredType* do CWMComp.

A Figura 18 mostra o modelo de seqüência do curso normal do caso de uso *Transformar Classe*. O UML2BDOR recupera as instâncias do tipo *Class* do UMLComp e as transformam em instâncias dos tipos *SqlStructuredType, Table* e *Dependency* do CWMComp.

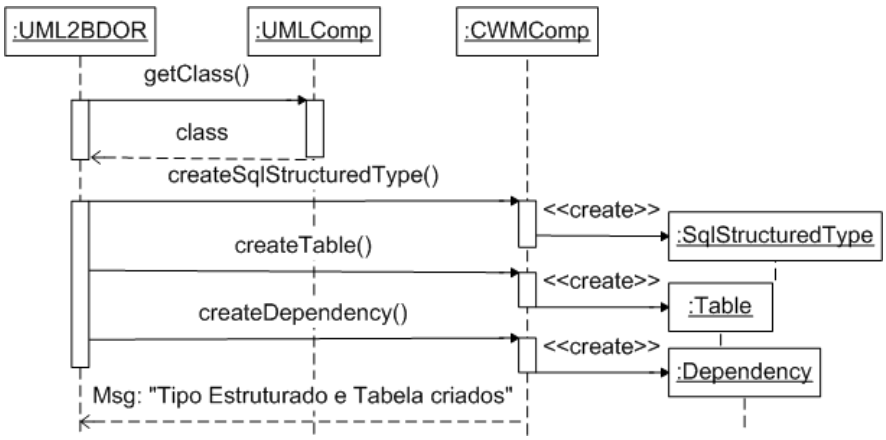


Figura 18 – Modelo de Seqüência Transformar Classe

A Figura 19 mostra o *Modelo de Seqüência Transformar Associação* que detalha o curso normal do caso de uso *Transformar Associação*. O UML2BDOR recupera as instâncias do tipo *Association* do UMLComp e as transformam em instâncias dos tipos *Association, SqlStructuredType, Table* e *Dependency* do CWMComp, desde que sejam cumpridas as restrições A ou B e C:

- **Restrição A**: uma instância do tipo *Association* do CWMComp é criada para cada instância do tipo *Association* recuperada do UMLComp que contenha o valor da multiplicidade igual a *um pra um* ou *um pra muitos*
- **Restrição B**: caso não seja satisfeita a *Restrição A* e a multiplicidade seja *um pra muitos*, instâncias dos tipos *SqlStructuredType, Table, Dependency* e *Association* do CWMComp são criadas para cada instância do tipo *Association* recuperadas do UMLComp.
- **Restrição C**: atualizações nas instâncias do tipo *Association* (mudança de valor de um de seus atributos para o valor *composite*) e *Table* (mudança do valor de seu estereótipo para *NestedTable*) do CWMComp são executadas

para cada instância do tipo *Association* recuperada do UMLComp que contenha um atributo com valor igual a *composite,* indicando dessa forma, uma tabela aninhada.

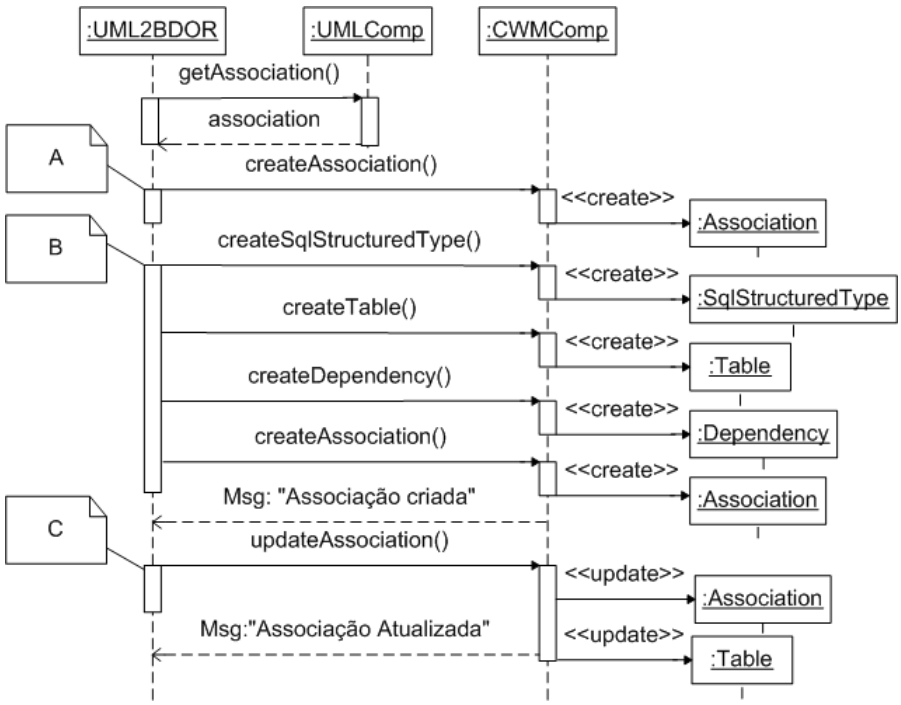


Figura 19 – Modelo de Seqüência Transformar Associação

A Figura 20 mostra o *Modelo de Seqüência Transformar Generalização* que detalha o curso normal do caso de uso *Transformar* Generalização. O UML2BDOR recupera as instâncias do tipo *Generalization* do UMLComp e as transformam em instâncias do tipo *Generalization* do CWMComp. Essa transformação de generalizações desconsidera a possível variância semântica que pode ser representada em Modelos UML por meio de restrições especificadas entre classes generalizadas e especializadas, como por exemplo, disjunção, sobreposição, etc. Assim, também não se considera essas possibilidades de restrições semânticas nos Modelos BDOR.

A definição das regras de transformação do Modelo UML para o Modelo BDOR foram baseadas no conhecimento do domínio OO e do domínio de BDOR e também das estruturas dos elementos dos componentes UMLComp e CWMComp.

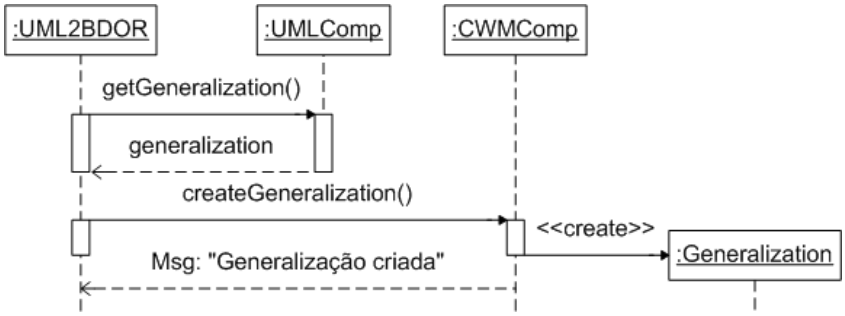


Figura 20 – Modelo de Seqüência Transformar Generalização

A transformação de Modelos BDOR para Códigos SQL3 também se dá de forma automática e guiada por elementos do metamodelo. Essa transformação também é implementada na forma de componente de *software*. Esse componente é chamado de **BDOR2SQL** e é implementado de acordo com a abordagem de manipulação direta, na qual suas regras de transformação consistem em navegar na estrutura interna dos elementos do Modelo BDOR coletando informações e as transformando em Códigos SQL3. A Figura 21 mostra as classes e relacionamentos internos desse componente.

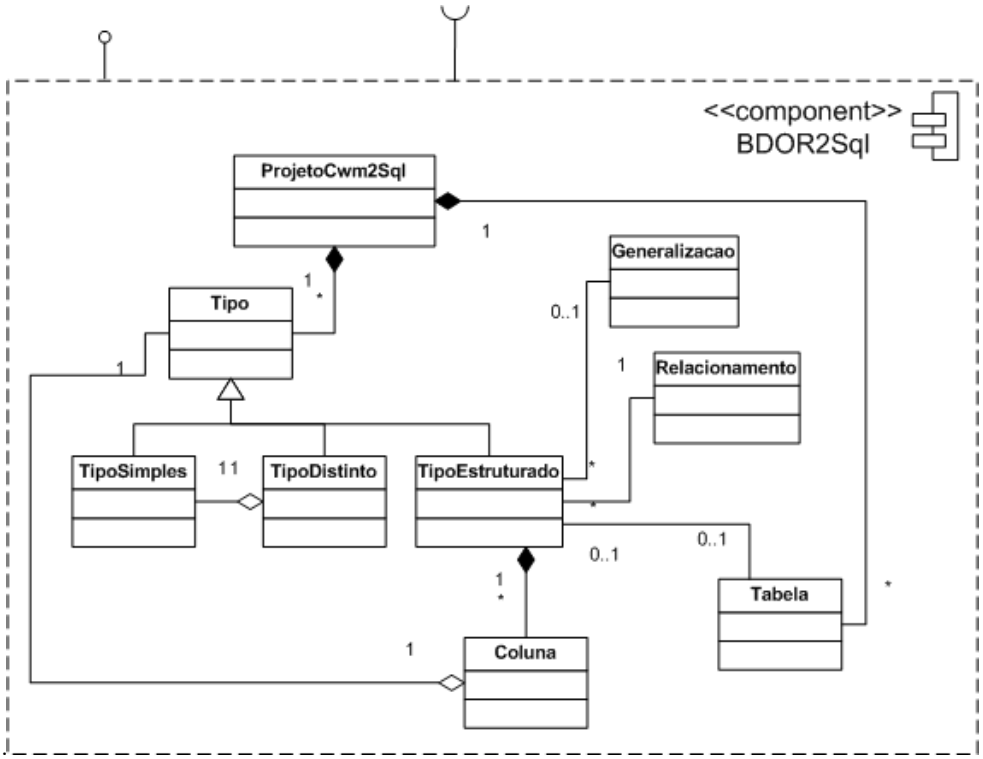


Figura 21 – Componente BDOR2SQL

A classe **ProjetoBDOR2SQL** é responsável pela organização da transformação, a qual é executada na seguinte ordem: tipos pré-definidos, tipos distintos, tipos estruturados, colunas, tabelas dependentes dos tipos estruturados, associações, agregações, composições e generalizações.

As regras de mapeamento responsáveis pela transformação de Modelos BDOR em Códigos SQL3 são mostradas na Tabela 4.

Tabela 4 – Regras de mapeamento de Modelos BDOR para Códigos SQL

Elemento do CWMCom	**Código SQL3**
SqlSimpleType	<sqlSimpleTypeList> <sqlSimpleTypeList> = varchar| number| date| etc
SqlDistinctType	Create Distinct Type *sqlDistinctTypeInstance* As <sqlSimpleTypeList>
SqlStructuredType, Column, Association (associação e agregação) e Generalization	Create Type *SqlStructuredTypeInstance* <elementList> <columnsList> [NOT FINAL] <elementList> = As Object| As Table Of *sqlStructuredTypeInstance*| Under *sqlStructuredTypInstance* <columnsList> = *columnInstance* <dataTypeList> <dataTypeList> = *sqlSimpleTypeList*| *sqlDistinctTypeInstance*| <sqlStructuredTypeList> <sqlStructuredTypeList> = [*Ref*] *sqlStructuredTypeInstance*
Table, Dependency e Associação (composição)	Create Table *tableInstance* Of *dependencyInstance* [*columnInstance* <typeList>] <typeList> = Nested Table *columnInstance* Store As "NT_"+*columnInstance*

Visto que cada SGBD pode implementar um dialeto SQL próprio baseado nas especificações do padrão SQL [ISO/IEC 1999], optou-se por escolher o dialeto SQL3 utilizado pelo SGBDOR Oracle 10G, o qual é compatível com a especificação *Core* do padrão SQL3 [ORACLE 2005].

A seguir é mostrada a implantação dos componentes desenvolvidos nas etapas Implementar Metamodelo e Implementar Transformação à ferramenta MVCASE.

3.1.3 Implantar Componentes na MVCASE

Nesta etapa, tornou-se operacional o suporte para a transformação de Modelos UML em Modelos de BDOR na ferramenta MVCASE. A decisão pelo uso da MVCASE foi devido principalmente ao fato de ser uma ferramenta que vem sendo utilizada em diferentes pesquisas no GOES, cujo código é livre, e possui uma arquitetura adequada para viabilizar a operacionalização das transformações.

A Figura 22 mostra a integração dos artefatos criados nas duas primeiras etapas à arquitetura da MVCASE. Esses artefatos são representados na figura por meio dos símbolos de componentes de *software* da notação UML.

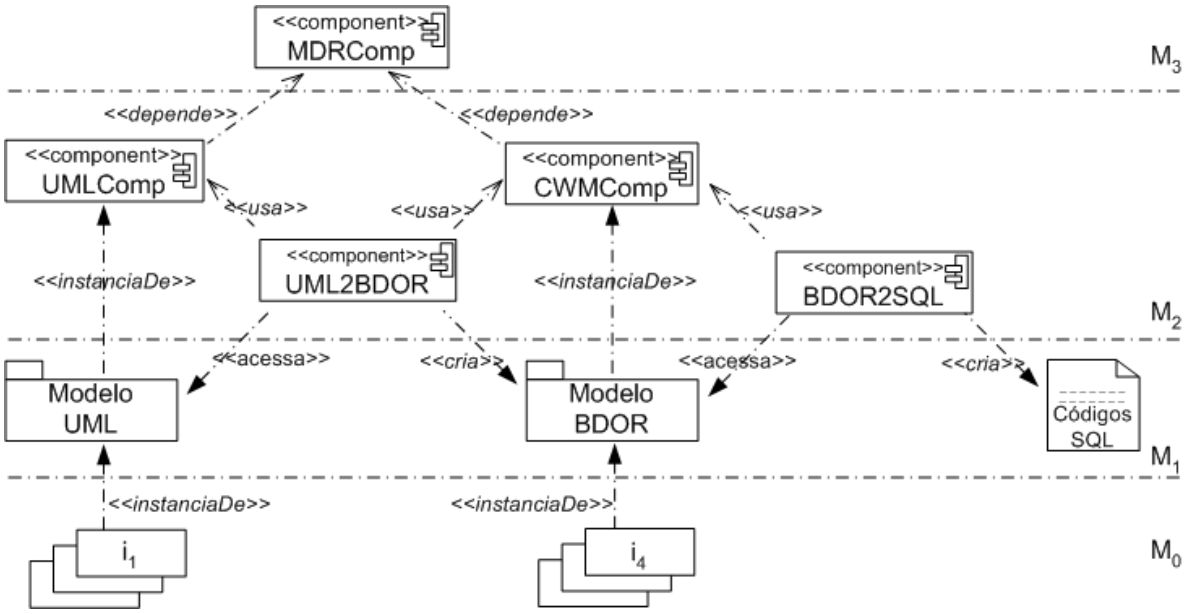


Figura 22 – Arquitetura da MVCASE e Componentes que Apóiam a Transformação de Modelos

Os 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** provê o suporte aos Modelos UML. O componente **CWMComp**, que é resultado da primeira etapa de desenvolvimento da extensão, é integrado à MVCASE com o objetivo de prover o suporte aos Modelos BDOR.

O componente **UML2BDOR** é integrado com o objetivo de oferecer o suporte à Transformação de Modelos UML para Modelos BDOR. O componente **BDOR2SQL** é integrado à MVCASE com o objetivo de oferecer o suporte à transformação de Modelos BDOR para Códigos SQL3.

Os componentes **CWMComp**, **UML2BDOR e BDOR2SQL** são implantados na arquitetura da MVCASE, a qual é estendida para suportar a transformação de Modelos UML em Modelos BDOR e geração de Códigos SQL3. A extensão da MVCASE envolveu as seguintes alterações:

- **Suporte ao CWMComp**: foi adicionado à MVCASE um módulo para criação de instâncias lógicas de Modelos BDOR. Assim, o CWMComp [OMG 2003Cwm] foi incorporado à arquitetura da ferramenta a fim de possibilitar a instanciação de elementos desse metamodelo com o objetivo de representar Modelos BDOR; e
- **Componentes de transformação**: a MVCASE foi alterada com o intuito de incorporar os componentes de transformação, a fim de disponibilizar um ambiente gráfico capaz de guiar o Engenheiro de *Software* na abordagem do desenvolvimento dirigido por modelos.

A Figura 23 mostra outra visão da integração dos componentes utilizados pela ferramenta MVCASE. A MVCASE acessa o componente MDRComp a fim de instanciar o repositório de metadados para os Modelos UML e BDOR, os quais são baseados nos componentes UMLComp e CWMComp. O componente BDOR2SQL é utilizado pela MVCASE para transformar um Modelo UML baseado no UMLComp em um Modelo BDOR baseado no CWMComp. O mesmo ocorre com o BDOR2SQL, que é utilizado pela MVCASE para transformar um Modelo BDOR em Códigos SQL3.

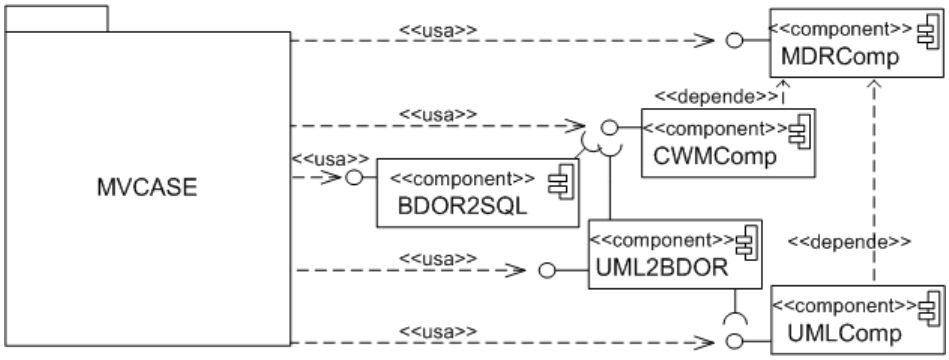


Figura 23 – Componentes Integrados à MVCASE

A Figura 24 mostra a janela principal da MVCASE que disponibiliza o acesso às transformações que são executadas pelos componentes UML2BDOR e BDOR2SQL.

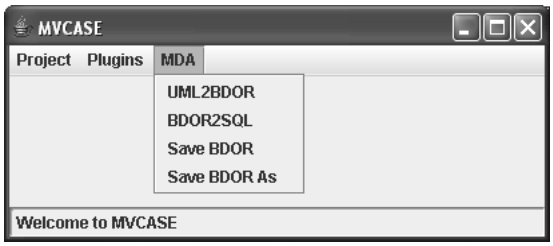


Figura 24 – Interface de acesso às transformações

Assim, o menu MDA é composto pelas seguintes opções:

- **UML2BDOR**: possibilita acesso à transformação de Modelos UML para Modelos BDOR.
- **BDOR2SQL**: possibilita acesso à transformação de Modelos BDOR para Códigos SQL3;
- **Save BDOR e Save BDOR As**: possibilita o armazenamento dos Modelos BDOR em XMI e serialização em arquivo texto dos Códigos SQL desses modelos.

O resultado desta etapa é a extensão da MVCASE para oferecer ao Engenheiro de *Software* apoio na transformação de Modelos UML em Modelos BDOR e geração de Códigos SQL3. Assim, com base no que foi visto até este ponto, seguem as considerações dessa extensão em relação aos requisitos levantados no início deste capítulo.

3.2 Considerações

A extensão efetuada na MVCASE visa facilitar o trabalho do Engenheiro de *Software* no desenvolvimento de sistemas de *software* para o domínio BDOR, visto que podem ser obtidos Modelos BDOR e Códigos SQL3 a partir de um Modelo UML, oferecendo ganhos em termos de produtividade por meio do reuso de modelos e geração de códigos a partir desses.

Além disso, os requisitos apresentados no início deste capítulo foram atendidos:

- **Usar linguagens padronizadas e ser genérico**: o uso de linguagens padronizadas como MOF, UML, CWM, XMI e SQL3 indica que são linguagens que já foram testadas em diversos cenários e são implementadas por outras ferramentas, fator esse que possibilita uma maior integração de informações entre ferramentas distintas que utilizam o mesmo conjunto de linguagens;
- **Transformação de Modelos UML em Modelos BDOR**: essa transformação possibilita a criação de Modelos BDOR a partir de Modelos UML, proporcionando o reuso no nível de modelos;
- **Transformação de Modelos BDOR em Códigos SQL3**: essa transformação possibilita a geração de Códigos SQL3, de acordo com o dialeto SQL do Oracle 10G, a partir de Modelos BDOR;
- **Transformação automática**: as transformações ocorrem de maneira automática. Assim, o Engenheiro de *Software* não necessita ter conhecimento prévio do domínio de BDOR para gerar um sistema de *software* desse domínio; e

Outro aspecto importante resultante deste trabalho de pesquisa é a contribuição científica. Assim, a validade e importância dessas contribuições, tais como a regras de mapeamento responsáveis pela transformação de Modelos UML para Modelo BDOR e Código SQL3 com ênfase nas características OO do padrão SQL3, e a implementação das transformações

por meio de componentes de *software,* podem ser comprovadas através de publicações reconhecidas pela comunidade científica [PEREIRA *et al.* 2007] e [PEREIRA *et al.* 2007a].

O próximo capítulo é dedicado à ilustração do uso da extensão da ferramenta MVCASE em um estudo de caso.

Capítulo 4

Estudo de Caso e Avaliação

Este capítulo apresenta um estudo de caso para ilustrar o processo de desenvolvimento dirigido por modelos implementado na MVCASE de acordo com a abordagem MDA. Esse processo compreende as etapas de construção de um Modelo UML e sua transformação em um Modelo BDOR, e a geração de Códigos SQL3 de acordo com o SGBDOR Oracle 10G.

Apesar de originalmente os BDOR se destinarem a sistemas de *software* que necessitam armazenar informações "complexas", como por exemplo, os sistemas de informações geográficas, este estudo de caso foi baseado no desenvolvimento de um sistema de *software* do domínio de Vendas por Atacado que armazena suas informações em BDOR. Naturalmente, essa simplificação não invalida a abordagem, pois os BDOR também devem ser capazes de suportar as informações dos sistemas convencionais. O sistema desenvolvido neste estudo de caso é chamado de *Sale* e se encontra no seguinte cenário:

- Cada consumidor pode adquirir produtos por meio de um pedido de compra;
- Cada pedido de compra contém um ou mais produtos;
- Cada produto que consta no pedido tem uma quantidade e a seqüência que esse produto foi adquirido;
- Cada pedido tem um único consumidor e um único vendedor;

- Cada vendedor está relacionado a uma ou mais empresas;
- Cada empresa tem um ou mais vendedores cadastrados;

É necessário ainda armazenar as seguintes informações:

- Vendedor: cadastro de pessoa física (CPF), nome, telefone e endereço;
- Endereço: código de endereçamento postal (CEP) e observação;
- Consumidor: cadastro nacional de pessoa jurídica (CNPJ), nome, telefone e endereço;
- Pedido: número do pedido, data do pedido, endereço de entrega, quantidade de cada produto adquirido e a seqüência que o mesmo foi adquirido;
- Produto: número de identificação e uma descrição.

A próxima seção mostra o Modelo UML modelado com base nos requisitos desta seção.

4.1 Modelo UML

No *Sale* foram exploradas as seguintes características semânticas:

- Tipos de dados complexos;
- Generalização simples;
- Relacionamento dos tipos associação e composição com as multiplicidades um pra muitos e muitos pra muitos.

Assim, os requisitos apresentados neste capítulo geraram o Modelo UML da Figura 25. As classes *Vendedor* e *Consumidor* tiveram seus dados comuns generalizados na classe *Pessoa*. São utilizados três tipos de dados pré-definidos neste exemplo, *Integer, String* e *Date,* para representar tipos de dados correspondentes a numéricos, caracteres e datas. O tipo de dado *Telefone* é um tipo de dado que foi construído a partir do tipo pré-definido *Integer*. O tipo de dado *Endereço* é um tipo de dado composto pelos tipos *CEP* e *Observação*. O tipo de dado *CEP* foi

construído a partir do tipo pré-definido *Integer* e o tipo *Observação* foi construído a partir do tipo *String*.

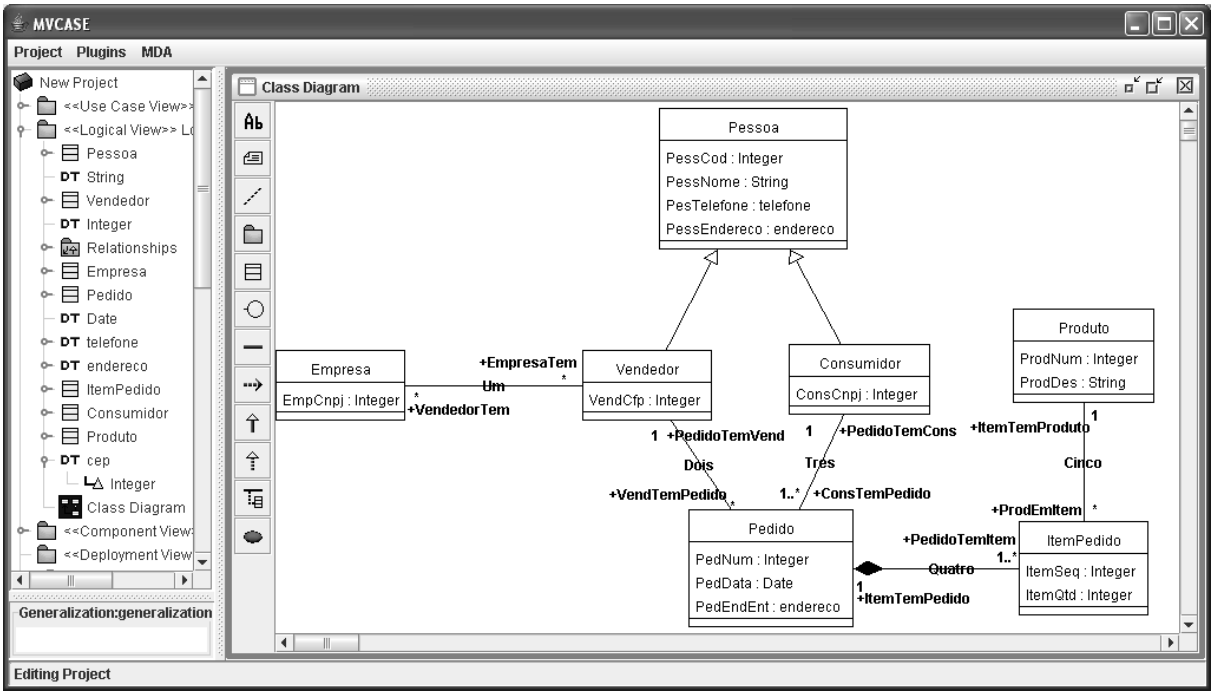


Figura 25 – Modelo UML do *Sale*

O relacionamento das classes *Vendedor* e *Consumidor* com a classe *Pedido* é do tipo associação e tem multiplicidade *um pra muitos*. O relacionamento entre a classe *Pedido* e a classe *ItemPedido* é do tipo composição, isto é, um objeto *ItemPedido* só existe enquanto o objeto *Pedido* existir. Esse relacionamento tem multiplicidade *um pra muitos*. A classe *Empresa* possui um relacionamento do tipo associação com a classe *Vendedor* e tem multiplicidade *muitos pra muitos*.

A próxima seção mostra o modelo BDOR obtido a partir do Modelo UML descrito nesta subseção.

4.2 Modelo BDOR

O Modelo BDOR do *Sale* ilustrado na Figura 26 é obtido a partir do Modelo UML da Figura 25 com base nas regras de mapeamento definidas para a transformação de Modelos UML

em Modelos BDOR e implementadas no componente UML2BDOR descrito na subseção 3.1.2 do capítulo 3.

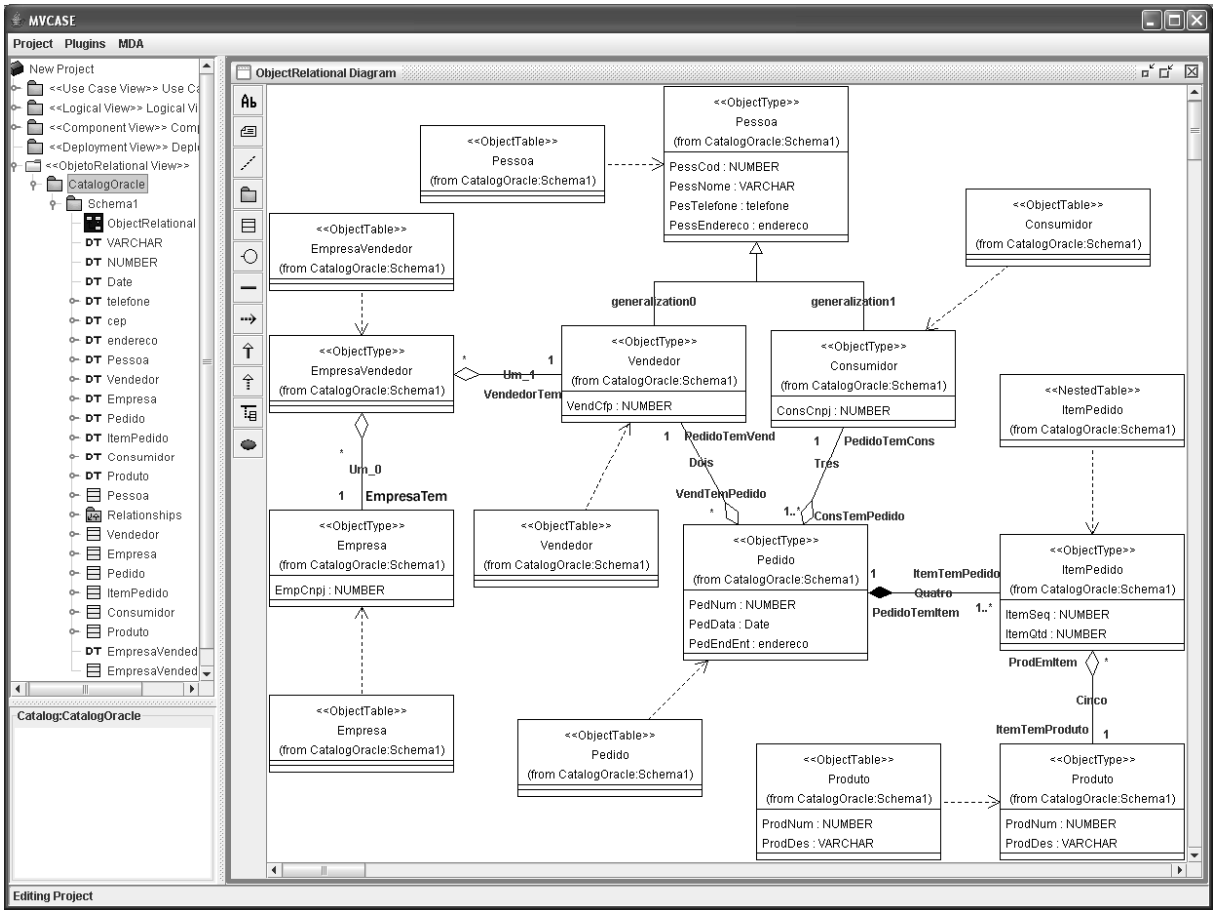


Figura 26 – Modelo BDOR do *Sale*

Como exemplo da regra de mapeamento da transformação de classes para tipos estruturados e tabelas com forte estrutura de tipos, pode-se observar na Figura 27 que a classe *Pessoa* do Modelo UML foi transformada em uma tabela com forte estrutura de tipos dependente de um tipo estruturado, ambas chamadas *Pessoa*, com os estereótipos: *ObjectTable* e *ObjectType*.

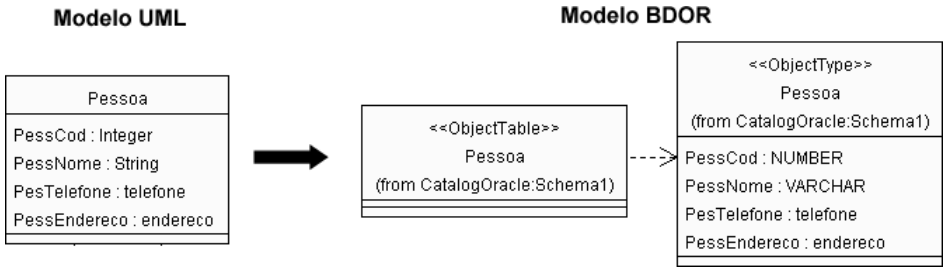


Figura 27 – Transformação de Classes, Atributos e Tipos de Dados

Os atributos da classe *Pessoa* foram mantidos no tipo estruturado *Pessoa*, porém seus tipos de dados foram convertidos de acordo com as regras de mapeamento responsáveis pela transformação de tipos de dados. Assim, os atributos pré-definidos *Integer* e *String* foram convertidos para tipos pré-definidos do SGBDOR Oracle 10G, *Varchar* e *Number.* O tipo *Telefone* foi transformado em um tipo *DistinctType* baseado no tipo *Number* e o tipo *Endereço,* por ser um tipo composto, foi transformado em um tipo estruturado com os atributos *CEP* e *Observacao.* O tipo *CEP* foi transformado em um tipo *DistinctType* baseado no tipo *Number,* e o tipo *Observacao* foi transformado em um outro tipo *DistinctType* baseado no tipo *Varchar.*

A Figura 28 mostra o relacionamento no Modelo UML com multiplicidade *muitos pra muitos* e sua transformação no Modelo BDOR para dois novos relacionamentos com multiplicidade *um pra muitos.*

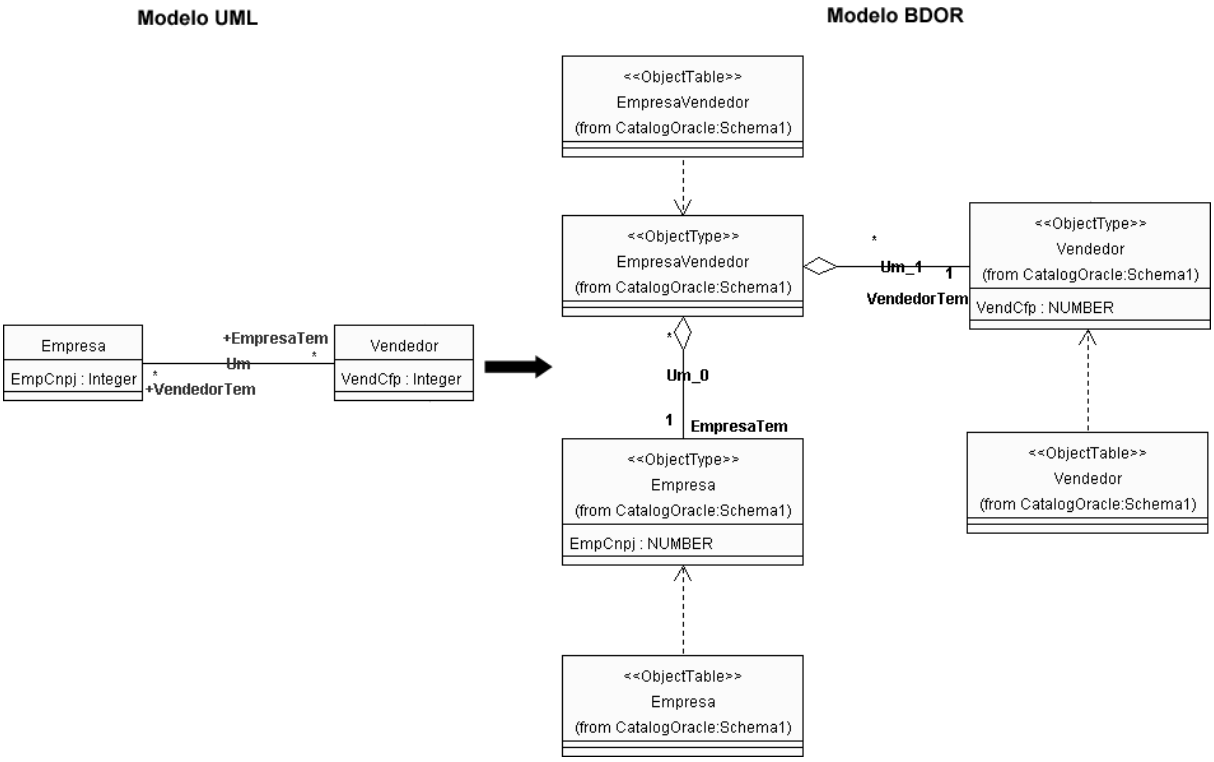


Figura 28 – Transformação de Relacionamentos com Multiplicidade Muitos pra Muitos

Um relacionamento *muitos pra muitos* origina dois novos relacionamentos com multiplicidades *um pra muitos*, estes são estabelecidos por referência (REF) entre tipos estruturados e são representados graficamente pelo símbolo agregação a fim de indicar que um tipo estruturado do lado com multiplicidade *muitos* possui uma coluna com referência ao tipo estruturado do lado com multiplicidade *um*.

Nos casos de relacionamentos *um pra muitos*, são mantidos os relacionamentos que já existem e a agregação fica representada do lado com multiplicidade *muitos*, indicando que o tipo estruturado relacionado a esse lado possui uma coluna que referencia o tipo estruturado do lado com multiplicidade *um*. Para os casos de relacionamentos *um pra um*, é atribuído a um dos lados aleatoriamente a representação de uma agregação para indicar uma referência (REF) entre tipos estruturados.

A próxima seção mostra os Códigos SQL3 obtidos a partir do Modelo BDOR descrito nesta subseção.

4.3 Geração de Códigos SQL3

Os Códigos SQL3 do *Sale* são obtidos a partir do Modelo BDOR apresentado na Figura 26 de acordo com as regras de mapeamento definidas para a transformação de Modelos BDOR em Códigos SQL3 e implementadas no componente BDOR2SQL descrito na subseção 3.1.2 do capítulo 3.

A Tabela 5 mostra os Códigos SQL3 obtidos a partir do Modelo BDOR. Esses códigos são descritos de acordo com o dialeto da linguagem SQL do SGBDOR Oracle 10G conforme especificado no componente BDOR2SQL.

Tabela 5 – Código SQL3 do *Sale*

```
Create DistincType telefone As NUMBER;

Create DistincType cep As NUMBER;

Create Type endereco As Object (enderecamentoPostal cep,
      observacao VARCHAR);

Create Type Pessoa As Object (PessCod NUMBER, PessNome VARCHAR,
      PesTelefone telefone, PessEndereco endereco) NOT FINAL;

Create Type Consumidor Under Pessoa(ConsCnpj NUMBER);

Create Type Vendedor Under Pessoa(VendCfp NUMBER);

Create Type EmpresaVendedor As Object (EmpresaTem Ref Empresa,
      VendedorTem Ref Vendedor);

Create Type Empresa As Object (EmpCnpj NUMBER);

Create Type Produto As Object (ProdNum NUMBER, ProdDes VARCHAR);

Create Type ItemPedido As Object (ItemSeq NUMBER,
      ItemQtd NUMBER, ItemTemProduto Ref Produto);

Create Type Tb_ItemPedido As Table of ItemPedido;

Create Type Pedido As Object (PedNum NUMBER, PedData Date,
      PedEndEnt endereco, PedidoTemItem Tb_ItemPedido,
      PedidoTemVend Ref Vendedor, PedidoTemCons Ref Consumidor);

Create Table Pessoa Of Pessoa;

Create Table Vendedor Of Vendedor;

Create Table Empresa Of Empresa;

Create Table Pedido Of Pedido;
      Nested Table PedidoTemItem Store As NT_PedidoTemItem;

Create Table Consumidor Of Consumidor;

Create Table Produto Of Produto;

Create Table EmpresaVendedor Of EmpresaVendedor;
```

4.4 Discussão

A transformação observada neste estudo de caso possibilitou gerar um Modelo BDOR de acordo com as características OO contidas no padrão SQL3 a partir de um Modelo UML, e ainda possibilitou gerar os Códigos SQL3 a partir de um Modelo BDOR.

Os elementos do Modelo BDOR e os Códigos SQL3 criados a partir do Modelo UML são resumidos na Tabela 6.

Tabela 6 – Comparativo entre Elementos do Modelo UML e do Modelo BDOR

Elementos do Modelo UML	QTD	Elementos do Modelo BDOR	QTD	Códigos SQL3	QTD
Tipos Pré-Definidos	2	Tipos Pré-Definidos	2	Tipos Pré-Definidos	2
Tipos Complexos	2	Tipos Distintos	2	DistinctType	2
Classes	7	Tipos Estruturados	9	Type As Object	9
		Tabelas com Forte Estrutura de Tipos	7	Table Of Type	8
Associação	4	Relacionamento por Referência (Ref)	5	Ref	5
Composição	1	Relacionamento por Tabela Aninhada	1	NestedTable e Type As Table	1
Generalização	2	Generalização	2	Under	2

Conforme pode ser observado na Tabela 6, o Engenheiro de *Software* somente necessitou desenvolver o Modelo UML, e a partir desse, pode obter o Modelo BDOR e seus Códigos SQL3. Portanto, os resultados alcançados neste trabalho de pesquisa puderam ser observados de forma qualitativa, identificada pela redução no tempo de desenvolvimento do sistema de *software Sale*. Essa redução no tempo de desenvolvimento foi obtida através da transformação de Modelos UML em Modelos BDOR e Códigos SQL3.

Outras facilidades como a possibilidade de intercambiar informações do Modelo BDOR por meio de documentos XMI foram atingidas. Este estudo de caso permitiu observar as vantagens do uso da transformação de modelos em uma CASE para apoiar o desenvolvimento de sistemas de *software* para o domínio de BDOR.

Capítulo 5

Considerações Finais

Este capítulo apresenta as contribuições obtidas e trabalhos futuros identificados ao longo do desenvolvimento deste trabalho de pesquisa.

5.1 Contribuições

O protótipo implementado neste trabalho de pesquisa é diferenciado dos trabalhos descritos na seção trabalhos correlatos do capítulo 2 visto que, aquele implementa o padrão CWM e utiliza uma extensão da notação gráfica UML para representar os Modelos BDOR e ainda utiliza componentes de *software* para transformar Modelos UML em Modelos BDOR, os quais suportam as características OO do padrão SQL3, e ainda possibilita a obtenção de Códigos SQL3 de acordo com o SGBDOR Oracle 10G a partir desses modelos. Assim, a extensão implantada na MVCASE por meio de componentes de *software* operacionalizou o processo de desenvolvimento de sistemas de software para o domínio de BDOR usando a abordagem MDA. Em resumo, essa extensão permite a transformação de Modelos UML em Modelos BDOR com ênfase em suas características OO e, também permite a obtenção de Códigos SQL3 a partir desses modelos.

Com base na MDA, esta pesquisa contribuiu nos seguintes aspectos:

- **Produtividade**: as transformações proporcionaram ganho de produtividade, visto que um Modelo BDOR e seus Códigos SQL3 podem ser obtidos a partir de um Modelo UML;
- **Manutenibilidade**: a manutenção pode ser feita em alto nível de abstração, diretamente nos Modelos UML. Assim, os Modelos BDOR e seus Códigos SQL3 podem ser gerados a partir desses Modelos UML;
- **Reutilização**: modelos são construídos em alto nível de abstração e podem ser reutilizados a fim de se obter modelos BDOR. A reutilização também se deu neste trabalho na forma de reuso dos artefatos produzidos, uma vez que as transformações foram implementadas na forma de componentes de *software*, podendo assim serem reutilizados por outras ferramentas de desenvolvimento; e
- **Flexibilidade**: a flexibilidade pode ser alcançada com a adição de novos componentes de transformação para proporcionar o reuso de modelos abstratos para diferentes plataformas de banco de dados.

Como contribuição desta pesquisa vale ressaltar ainda a utilização de linguagens padronizadas, visando a busca pelo uso comum de padrões para possibilitar o intercâmbio de informações de maneira facilitada entre ferramentas de desenvolvimento de sistemas de *software* e a comunicação entre as equipes de desenvolvimento pelo uso comum da notação UML.

Também é contribuição desta pesquisa a forma como as transformações foram integradas à arquitetura da MVCASE, visando a composição da extensão da ferramenta pelo uso de componentes de *software* que apóiam a transformação de Modelos UML em Modelo BDOR e Códigos SQL3.

5.2 Trabalhos Futuros

No decorrer do desenvolvimento deste trabalho de pesquisa identificou-se uma série de possibilidades que podem ser exploradas em projetos de pesquisa futuros, tais como:

- **Com relação à flexibilidade**: construir novas transformações para proporcionar a geração de outros modelos de banco de dados, como por exemplo, modelos XML, modelos multidimensionais, etc. A flexibilidade também pode ser aumentada por meio da construção de mecanismos que possibilitem a manutenção dos modelos BDOR de forma gráfica, possibilitando assim, que esses modelos sejam refinados de acordo com suas necessidades semânticas;
- **Com relação à variação semântica especificada em modelos abstratos**: desenvolver maneiras de tratar a variação semântica especificada nos relacionamentos de generalização, como por exemplo, disjunção, sobreposição, entre outros, para que possam ser refletidas também nos modelos BDOR;
- **Com relação à engenharia reversa de modelos**: com a possibilidade de permitir o refino dos modelos dependentes de plataforma de forma gráfica, faz-se necessário criar mecanismos para traduzir suas características a fim de que sejam retransmitidas aos modelos abstratos;
- **Com relação ao rastreamento de modelos**: desenvolver mecanismos que permitam a visualização das transformações por meio do mapeamento gráfico entre os elementos dos modelos abstratos e os elementos dos modelos específicos de plataforma facilitaria o entendimento das transformações entre os modelos; e

- **Com relação à busca de informações**: visto que a MDA é baseada no desenvolvimento de modelos em alto nível de abstração, seria interessante desenvolver mecanismos que permitissem a recuperação de informações contidas em diversos modelos abstratos de diferentes domínios de problemas para que sejam reaproveitadas na construção de modelos abstratos de novos domínios.

Finalmente, trabalhos futuros podem pesquisar outros meios de armazenamento de metamodelos que não seja o MDR. Visto que a ferramenta Eclipse é uma ferramenta bem aceita pela comunidade de *software* livre, as transformações de Modelos UML em Modelos BDOR e Códigos SQL3 podem ser implementadas na forma de *plugins* da ferramenta e disponibilizadas à comunidade.

Referências

[AIMEIDA et al. 2002] ALMEIDA, E. S., BIANCHINI, C. P., PRADO, A. F., TREVELIN, L. C. MVCase: An Integrating Technologies Tool for Distributed Component-Based Software Development. In: The 6th Asia - Pacific Network Operations and Management Symposium - Poster Session, Jeju Island, Korea. **Proceeding of the IEEE**. 2002.

[ALMEIDA et al. 2002a] ALMEIDA, E. S., LUCRÉDIO, D., BIANCHINI, C. P., PRADO, A. F., TREVELIN, L. C. Ferramenta MVCase - Uma Ferramenta Integradora de Tecnologias para o Desenvolvimento de Componentes Distribuídos. In: XVI Simpósio Brasileiro de Engenharia de Software (SBES) - Sessão de Ferramentas. Gramado, Brasil. **Anais do IX Simpósio Brasileiro de Engenharia de Software (SBES) - Sessão de Ferramentas**. 2002.

[ANDROMDA 2006] **Ferramenta AndroMDA**. Disponível em <http://www.andromda.org>. Acesso em: dezembro 2006.

[ANSI 2005] **American National Standards Institute**. Disponível em <http://www.ansi.org>. Acesso em: agosto 2005.

[ASSÁO 2003] ASSÁO, F. M. **Extensão da MVCASE para suportar a geração do modelo de dados a partir do modelo de objetos**. 2003. Relatório (Iniciação Científica em Computação) - Universidade Federal de São Carlos (UFSCar), São Carlos.

[BARRÉRE 1999] BARRÉRE, T. S. **CASE com Múltiplas Visões de Requisitos de Software e Implementação Automática em Java - MVCASE**. 1999. 100 p. Dissertação (Mestrado em Computação) - Universidade Federal de São Carlos (UFSCar), São Carlos.

[BOOCH 1993] BOOCH, G. **Object-Oriented Analysis and Design with Applications**. Benjamin-Cummings Publishing Company. 1993.

[BOOCH et al. 2005] BOOCH, G., RUMBAUGH, J., JACOBSON, I. **The Unified Modeling Language User Guide - Object Technology Series**. 2nd Edition. Addison Wesley, 2005.

[CHEN 1976] CHEN, P. P., The entity-relational model - toward a unified view of data. **ACM Transations on Database Systems**, v. 1, n. 1, 1976.

[CODD 1970] CODD, E. F. A relational model of data for large shared data banks. **Communications of the ACM**, v. 13, n. 6, p. 377-387. 1970.

[CRONHOLM 1995] CRONHOLM, S. Why CASE Tools in information Systems Development? - an Empirical Study Concerning Motives for Investing in Case Tools. In: **18th Information Systems research In Scandinavia (IRIS 18)**. Gjern, Denmark. 1995.

[CZARNECKI et al. 2005] CZARNECKI, K., ANTKIEWICZ, M. KIM, C.H.P., LAU, S., PIETROSZEK, K. Model-Driven Software Product Lines. **OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications**, New York, pp. 126-127. 2005.

[CZARNECKI e HELSEN 2006] CZARNECKI, K., HELSEN, S. Feature-based survey of model transformation approaches. **IBM Systems Journal**, Riverton, v. 45, n. 3, pp. 621-645. 2006.

[DATE 2003] DATE, C. J. **An Introduction to Database Systems. Addison-Wesley**. 8th Edition. Addison Wesley, 2003.

[DARWEN e DATE 1995] DARWEN, H., DATE, C. J. The third manifesto. **ACM SIGMOD Record**, New York, v. 24, n. 1, p. 39-49. 1995.

[DIRCKZE 2002] DIRCKZE, R. **The Java Metadata Interface (JMI) Specification - JSR-40**. Java Community Process. 2002.

[D'SOUZA e WILLS 1999] D'SOUZA, D. F., WILLS, A. C. **Objects, Components and Frameworks with UML, The Cataliysis Approach**. Addison Wesley, 1999.

[EISENBERG e MELTON 1999] EISENBERG, A., MELTON, J. SQL:1999, formely known as SQL3. **ACM SIGMOD Record**, New York, v. 28, n. 1, p. 131-138. 1999.

[EISENBERG e MELTON 1998] EISENBERG, A., MELTON, J. Standards in Pratice. **ACM SIGMOD Record**, New York, v. 27, n. 3, p. 53-58. 1998.

[EMF 2006] **The Eclipse Modeling Framework Overview - Eclipse Modeling Framework (EMF)**. Disponível em <http://www.eclipse.org/emf/docs>. Acesso em: setembro 2006.

[FARPINYO e SENIVONGSE 2003] FARPINYO, K., SENIVONGSE, T. Designing and creating relational schemas with a CWM-based tool. **ACM ISICT '03: Proceedings of the 1st international symposium on Information and communication technologies**, Dublin, v. 49, p. 456-461. 2003.

[FOLDOC 2006] **Free Online Dictionary of Computing**. Disponível em <http://foldoc..org>. Acesso em: fevereiro 2006.

[FRANKEL 2003] FRANKEL, D.S. **Model Driven Architecture: Applying MDA to Enterprise Computing**. Willey Press, 2003.

[FUGGETTA 1993] FUGGETTA, A. A Classification of CASE technology. **IEEE Computer**, Los Alamitos, v. 26, n. 12, p. 25–38. 1993.

[GALLAGHER 1994] GALLAGHER, L. Influencing database language standards. **ACM SIGMOD Record**, New York, v. 23, n. 1, p. 122-127. 1994.

[GANE 1979] GANE, C. P. **Structured Systems Analysis: Tools and Techniques**. Prentice Hall, 1979.

[GARCIA et al. 2004] GARCIA, V. C., LUCRÉDIO, D., PINTO, L. F. P., ALVARO, A., ALMEIDA, E. S., PRADO, A. F. Uma Ferramenta CASE para o Desenvolvimento de Software Orientado a Aspectos. In: XVIII Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas. Brasília, Brasil. **Anais do XI Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas**. 2004.

[IBM 2006] **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>. Acesso em: Dezembro 2006.

[IBM 2007] **International Business Machines Corporation**. Disponível em <http://www.ibm.com>. Acesso em: setembro 2007.

[IEC 2005] **International Electrotechnical Commission**. Disponível em <http://www.iec.ch>. Acesso em: agosto 2005.

[ISO 2005] ISO (2005). **International Organization for Standardization (ISO)**. Disponível em <http://www.iso.org>. Out.

[ISO/IEC 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.

[JACOBSON 1992] JACOBSON, I. **Object-Oriented Software Engeneering**. Addison Wesley, 1992.

[KLEPPE et al. 2003] KLEPPE, A., WARMER, J., BAST, W. **MDA Explained: The Model Driven Architecture: Practice and Promise - Object Technology Series**. 2nd Edition. Addison Wesley, 2003.

[LUCREDIO 2005] LUCRÉDIO, D. **Extensão da Ferramenta MVCASE com Serviços Remotos de Armazenamento e Busca de Artefatos de Software**. 2005. 103 p. Dissertação (Mestrado em Computação) - Universidade Federal de São Carlos (UFSCar), São Carlos.

[LUCREDIO et al. 2003] LUCRÉDIO, D., ALMEIDA, E. S., PRADO, A. F., YAMAMOTO, C. H., BIAJIZ, M. Abordagens para Recuperação Eficiente de Componentes utilizando Indexação Métrica. In: III WORKSHOP DE DESENVOLVIMENTO BASEADO EM COMPONENTES, São Carlos. **Anais do III Workshop de Desenvolvimento Baseado em Componentes**. 2003.

[LUCREDIO et al. 2003a] LUCRÉDIO, D., ALVARO, A., ALMEIDA, E. S., PRADO, A. F. MVCASE Tool - Working with Design Patterns. **In: The Third Latin American Conference On Pattern Languages Of Programming (SugarLoafPLoP'2003)**, Porto de Galinhas. 2003.

[MATULA 2006] MATULA, M. **NetBeans Metadata Repository (MDR) - NetBeans Community**. Disponível em <http://mdr.netbeans.org/docs.html>. Acesso em: setembro 2006.

[MELLOR et al. 2004] MELLOR, S. J., Scott, K., UHL, A. **MDA Distilled Principles of Model-Driven Architecture**. Addison-Wesley, 2004.

[MELTON e SIMON 2001] MELTON, J., SIMON, A. R. **Understanding Relational Language Components**. Morgan Kaufmann, 2001.

[MENS e VAN GORP 2005] MENS, T., GORP, VAN GORP, P. 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. 2005.

[MORAES 2004] MORAES, J. L. C. **Reutilizacão de Componentes de um Framework do Domínio de Cardiologia**. 2004. 128 p. Dissertação (Mestrado em Computação), Universidade Federal de São Carlos (UFSCar), São Carlos.

[MVCASE 2005] **The Source for Java Technology Collaboration**. Disponível em <https://mvcase.dev.java.net>. Acesso em: outubro 2005.

[NOVAIS 2002] NOVAIS, E. R. A. **Reengenharia de Software Orientada a Componentes Distribuídos**. 2002. 98 p. Dissertação (Mestrado em Computação) - Universidade Federal de São Carlos (UFSCar), São Carlos.

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

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

[OMG 2006Ocl] **Object Constraint Language (OCL) Specification Version 2.0**. Object Management Group. Document formal/2006-05-01 2006.

[OMG 2006Omg] **The Object Management Group (OMG)**. Disponível em <http://www.omg.org>. Acesso em: julho 2006.

[OMG 2003Mda] **The Model-Driven Architecture - Guide Version 1.0.1**. Object Management Group. Document omg/2003-06-01. 2003.

[OMG 2005Uml] **Unified Modeling Language (UML) Specification Version 1.4.2**. Object Management Group. Document formal/05-04-01. 2005.

[OMG 2007Uml] **Unified Modeling Language (UML) Specification Version 2.1.1**. Object Management Group. Document formal/07-02-03. 2007.

[OMG 2007Profile] **UML Profile**. Disponível em <http://www.omg.org/ /technology/documents/profile_catalog.htm>. Acesso em: janeiro 2007.

[OMG 2005Xmi] **XML Metadata Interchange 2.1 (XMI) Specification**. Object Management Group. Document formal/2005-09-01. 2005.

[ORACLE 2005] **Oracle Database - Application Development Guide - Object Relational Features 10G Release 2 (10.2). Disponível em Documentation Library**. Disponível em < http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14260.pdf>. Acesso em: setembro 2006.

[ÖZSU 1996] ÖZSU, M. T. Future of database systems : changing applications and technological developments. **ACM Computing Survey (CSUR)**, New York, v. 28, n. 4, 1996.

[PAIVA et al. 2006] PAIVA, D.M.B., LUCRÉDIO, D., FORTES, R.P.M. MVCASE - including design rationale to help modeling in research projects. In: XX - Simpósio Brasileiro de Engenharia de Software (SBES) - Sessão de Ferramentas, Florianópolis, Brasil. **Anais do XIII Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas**. 2006.

[PARDEDE et al. 2003] PARDEDE, E., RAHAYU, J. W., TANAIR, D. New SQL Standard For Object-Relational Database Applications. **Proceedings of the 3th IEEE Conference on Standardization and Innovation in Information Technology (SIIT 2003)**, Delft, The Netherlands, pp.191-203. 2003.

[PEREIRA et al. 2007] PEREIRA, M. A., PRADO, A. F., BIAJIZ, M., FONTANETTE, V. Usando MDA na Transformação de Modelo no Domínio UML em Modelo no Domínio de Banco de Dados Objeto-Relacional. In: XXXIII Conferência Latino-Americana de Informática (CLEI), San José. **Anais do XXXIII Conferência Latino-Americana de Informática (CLEI)**. 2007.

[PEREIRA et al. 2007a] PEREIRA, M. A., PRADO, A. F., BIAJIZ, M., FONTANETTE, V. Transformando Modelos da MDA com o Apoio de Componentes de Software. In: Simpósio Brasileiro de Componentes de Software, Arquitetura e Reuso - Sessão Técnica, Campinas. **Anais do Simpósio Brasileiro de Componentes de Software, Arquitetura e Reuso**. 2007.

[POOLE et al. 2003] POOLE, J., CHANG, D., TOLBERT, D., MELLOR, D. **Common Warehouse Metamodel - Developer's Guide**. Willey Publishing, 2003.

[PRADO e LUCREDIO 2000] PRADO, A. F., LUCRÉDIO, D. MVCASE: Ferramenta CASE Orientada a Objetos. In: XIV Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas, João

Pessoa, Brasil. **Anais do VII Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas**. 2000.

[PRADO e LUCREDIO 2001] PRADO, A. F., LUCRÉDIO, D. Ferramenta MVCASE - Estágio Atual: Especificação, Projeto e Construção de Componentes. In: XV Simpósio Brasileiro de Engenharia de Software (SBES) - Sessão de Ferramentas. Rio de Janeiro, v.1. p.368-373. **Anais do VIII Simpósio Brasileiro de Engenharia de Software - Sessão de Ferramentas**. 2001.

[PRESSMAN 2004] PRESSMAN, R. S. **Software Engineering**: **A Practitioner's Approach**. 6th Edition. McGraw-Hill, 2004.

[RAMAKRISHNAN e GEHRKE 2002] RAMAKRISHNAN, R., GEHRKE, J. **Database Management Systems**. McGraw-Hill, 2002.

[RUMBAUGH et al. 1990] RUMBAUGH, J., BLAHA, M. R., LORENSEN, W., EDDY, F.M PREMERLANI, W. **Object-Oriented Modeling and Design**. Prentice Hall. 1990.

[STONEBRAKER 1990] STONEBRAKER, M. Third-generation database system manifesto. **ACM SIGMOD Record**, **New York**, v. 19, n. 3, p. 31-44. 1990.

[UML2MOF 2006] **UML2MOF Tool**. **NetBeans Community**. Disponível em <http://mdr.netbeans.org/uml2mof>. Acesso em: agosto 2006.

[VARA et al. 2007] VARA, J. M., VELA, B., CAVERO, J. M., MARCOS, E. Model Transformation for Object-Relational Database Development. **ACM Symposium on Applied Computing (SAC) - Session**: **Model Transformation**, Seoul, p. 112-119. 2007.

[VÖLTER e STAHL 2006] VÖLTER, M., STAHL, T. () **Model-Driven Software Development**: **Technology**, Enginnering, Management. John Willey and Sons Ltda, 2006.

[WERNECK 2006] WERNECK, V. M. B. **Ferramentas Case**. Disponível em <http://www.ime.uerj.br/~vera/analise2/AnalII2007.pdf>. Acesso em: outubro 2006.

[W3C 2006] **World Wide Web Consortium (W3C)**. Disponível em <http://www.w3.org/XML>. Acesso em: setembro 2006.

[XML 2006] **Extensible Markup Language (XML) - W3C Architecture Domain**. Disponível em <http://www.w3.org/XML>. Acesso em: dezembro 2006.

[ZENDULKA 2005] ZENDULKA, J. Object-Relational Modeling in UML. **Encyclopedia of Database Technologies and Applications**, **Idea Group Publishing**, Hershey, pp. 421-426. 2005.

Anexo A

Anexo A – Especificação CWM

A especificação CWM [OMG 2003Cwm] é um documento que descreve o metamodelo estabelecido pelo OMG para suportar o domínio DW e BI. As figuras que seguem neste anexo ilustram os metamodelos que o *Relational* depende. A Figura A. 1 apresenta o metamodelo *Core*.

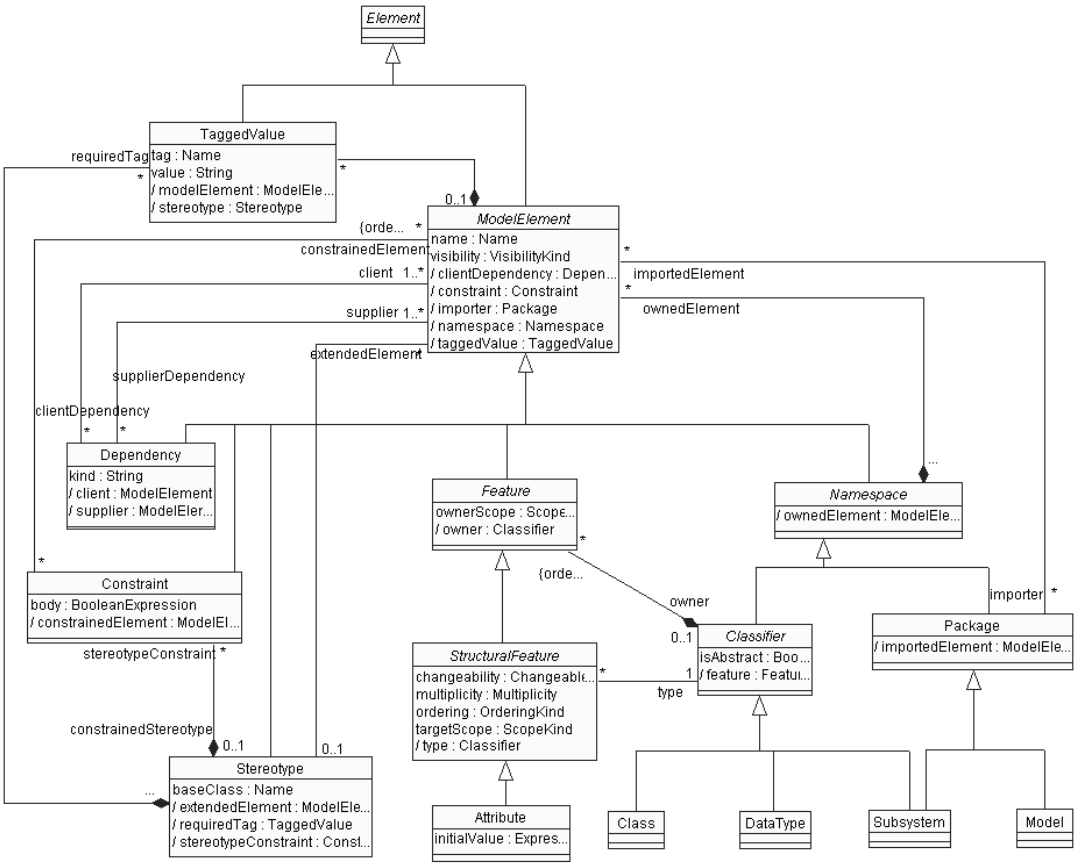


Figura A. 1 – Core [OMG 2003CWM]

A Figura A. 2 ilustra o metamodelo *Behavioral*.

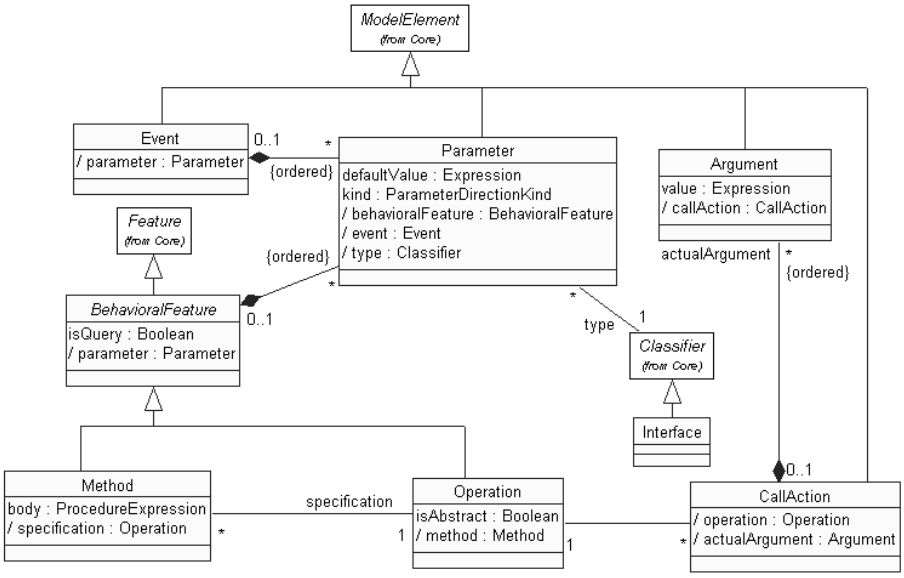


Figura A. 2 – Behavioral [OMG 2003CWM]

A Figura A. 3 mostra o metamodelo *Relationships*.

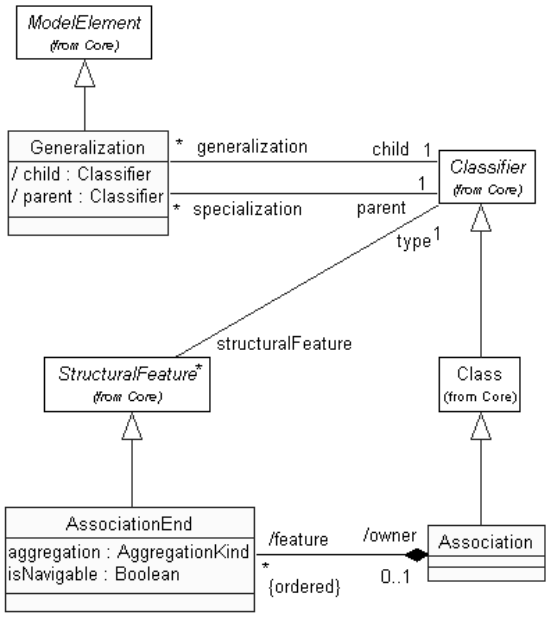


Figura A. 3 – Relationships [OMG 2003CWM]

A Figura A. 4 apresenta o metamodelo *Instance*.

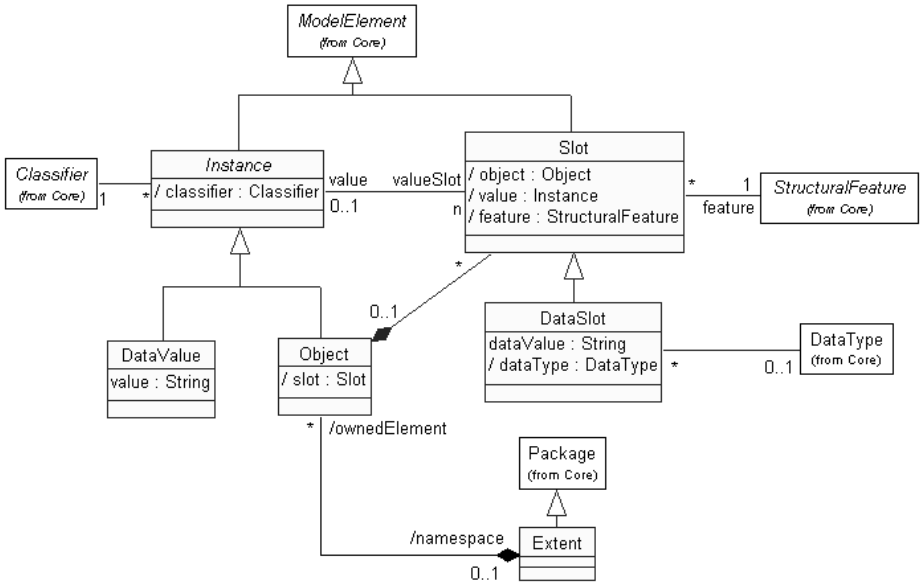


Figura A. 4 – Instance [OMG 2003CWM]

A Figura A. 5 ilustra o metamodelo *Data Types*.

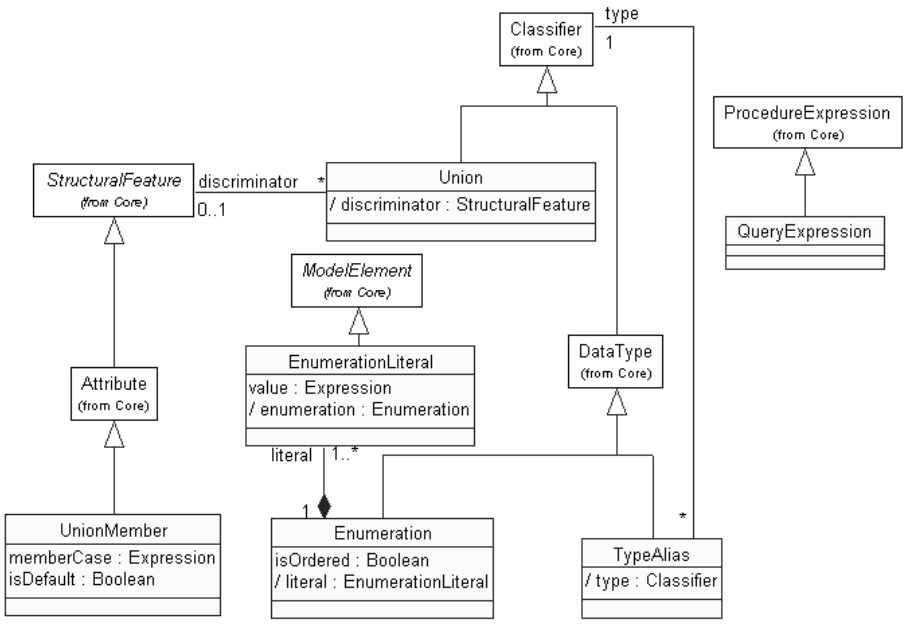


Figura A. 5 – Data Types [OMG 2003CWM]

A Figura A. 6 mostra o metamodelo *Keys and Indexes*.

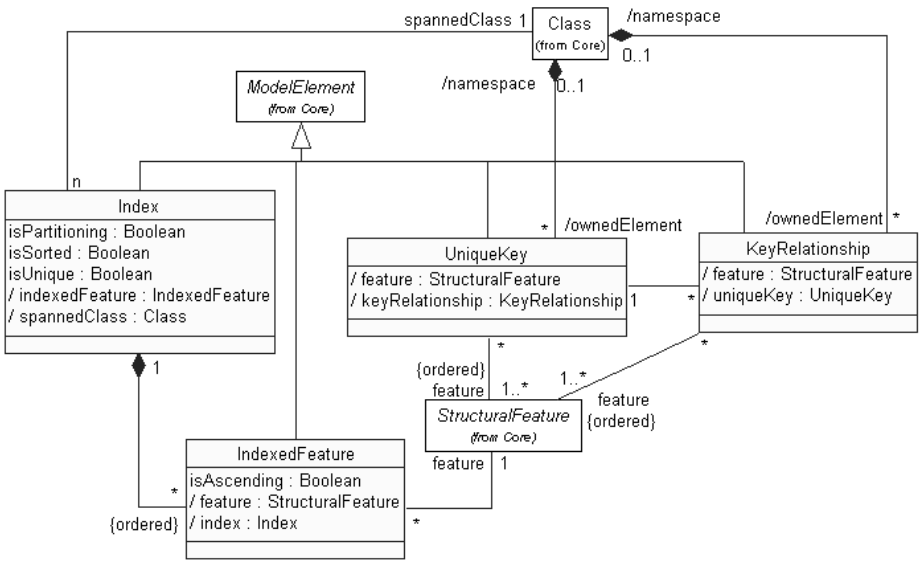


Figura A. 6 – Keys and Indexes [OMG 2003CWM]

A Figura A. 7 apresenta as classes principais do metamodelo *Relational*

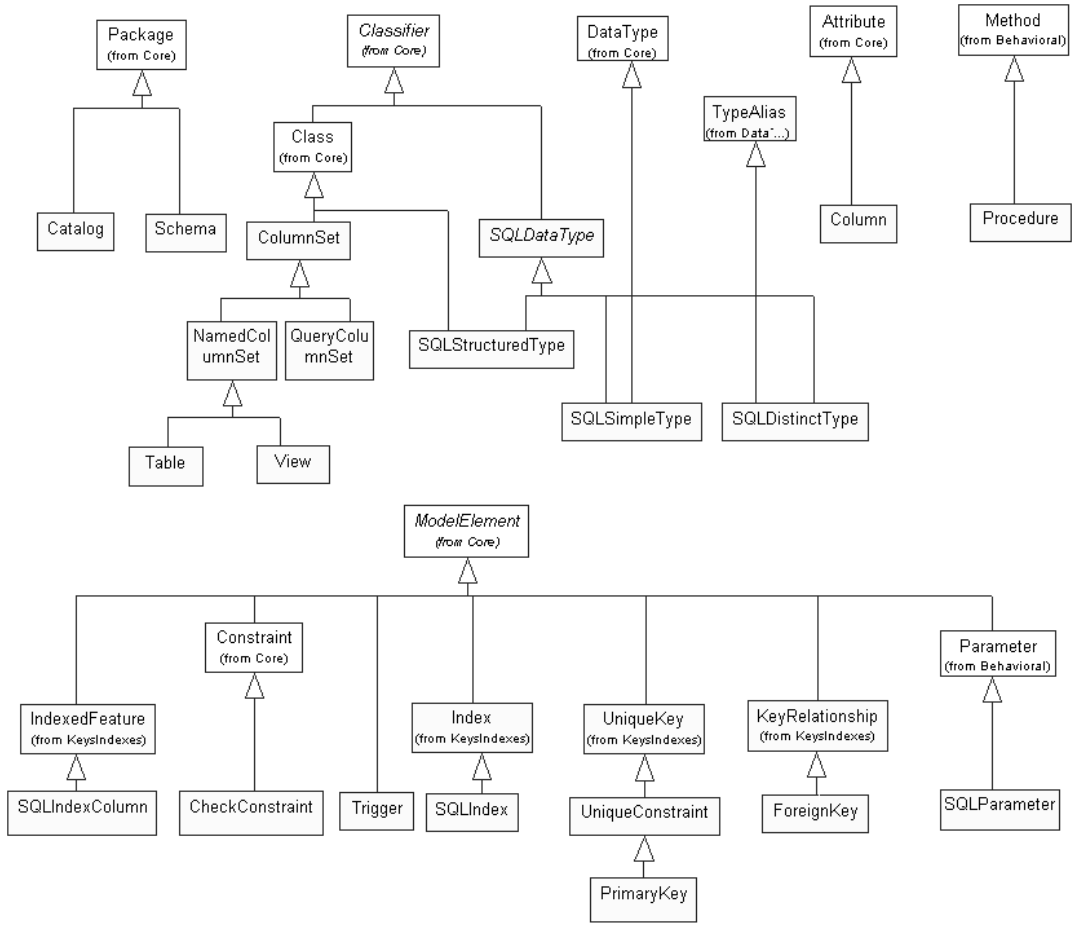


Figura A. 7 – Relational [OMG 2003CWM]

Anexo B

Anexo B – Metamodelo Gráfico

A Figura B. 1 ilustra o metamodelo Gráfico utilizado na ferramenta MVCASE para representar graficamente os Modelos BDOR.

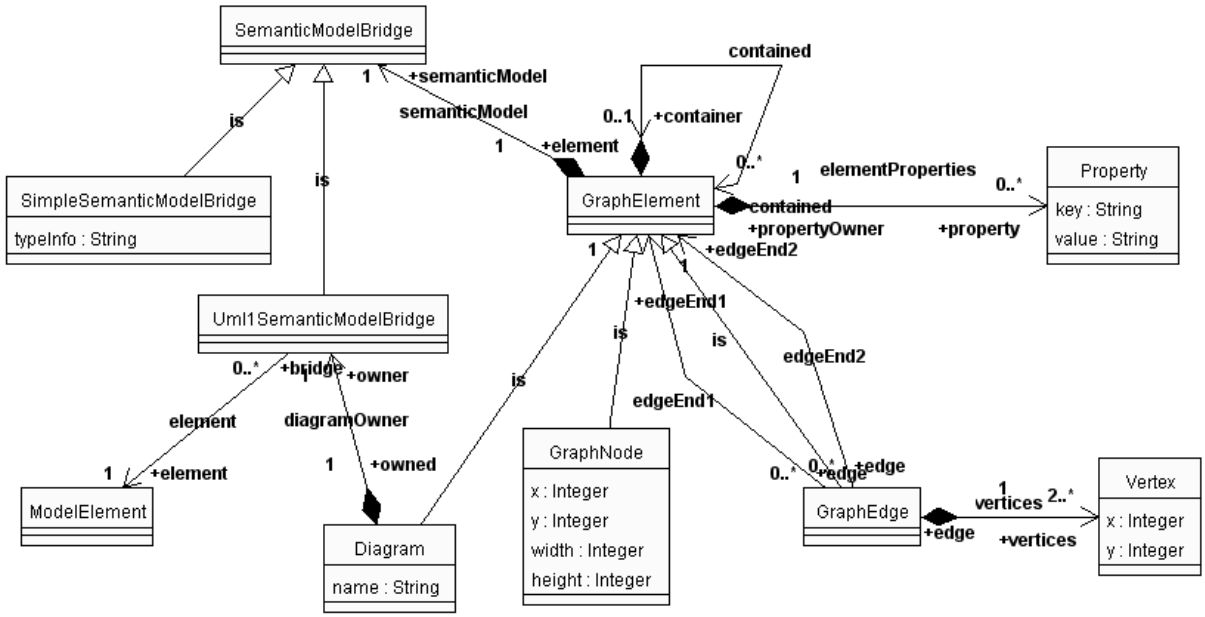


Figura B. 1 – Metamodelo Gráfico [LUCRÉDIO 2005]