THIAGO RAIS DE CASTRO

PROPOSTA DE UM MODELO PARA PROJETOS LÓGICOS GRÁFICOS PARA BDOR COM IMPLEMENTAÇÃO NO ARGOUML

São Paulo

2011

THIAGO RAIS DE CASTRO

PROPOSTA DE UM MODELO PARA PROJETOS LÓGICOS GRÁFICOS PARA BDOR COM IMPLEMENTAÇÃO NO ARGOUML

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

São Paulo

2011

THIAGO RAIS DE CASTRO

PROPOSTA DE UM MODELO PARA PROJETOS LÓGICOS GRÁFICOS PARA BDOR COM IMPLEMENTAÇÃO NO ARGOUML

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

Área de Concentração:
Sistemas Digitais

Orientadora:
Profa. Dra. Solange Nice Alves de Souza

São Paulo
2011

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, de de 2011.

Assinatura do autor _____

Assinatura do orientador _____

FICHA CATALOGRÁFICA

Castro, Thiago Rais de
Proposta de um modelo para projetos lógicos gráficos para BDOR com implementação no ARGOUML / T.R. de Castro. -- ed. rev. --São Paulo, 2011.
93 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Banco de dados. 2. Case. 3. UML. 4. Projeto lógico de computadores. 5. Arquitetura de software. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais. II. t.

DEDICATÓRIA

Dedico este trabalho em memória a minha querida avó Esmeraldina Mendonça (1929-2009). Pessoa iluminada, cuja passagem pela vida foi marcada pela luta, perseverança, retidão e superação. Admirada e amada pela família e conhecidos, ensinou-nos com sua postura sensata diante às adversidades; sua missão foi concluída com louvor e sua jornada serve-nos como um exemplo a ser seguido.

AGRADECIMENTOS

Primeiramente a Deus, que durante a elaboração deste trabalho mostrou-se sempre presente, intervindo de forma sutil nos acontecimentos e na seqüência com que eles foram ocorrendo, fazendo-me forte para superar os obstáculos e iluminando os caminhos escolhidos para serem trilhados.

Agradeço à minha mãe, Susete Z. M. R. C. de Castro, pelo apoio incondicional, por todo o amor e dedicação. Ao meu pai, Vanderley C. de Castro, admirado por sua hombridade, postura, determinação e perseverança.

Gostaria de agradecer ao meu grande amigo Professor Doutor Jaime S. da Veiga, por ter me apresentado o até então desconhecido universo da pesquisa e por ter me ensinado a dar os primeiros passos nessa área.

À minha orientadora, Professora Doutora Solange N. A. de Souza, pelas suas sugestões e idéias, pelo rigor científico, por toda atenção despendida, pelo carinho e palavras amigas e encorajadoras.

Gostaria de agradecer a Aline Hatadani, por ajudar-me a revisar este trabalho e pelo companheirismo ao longo de todos esses anos. E por fim, gostaria de agradecer aos professores e funcionários da Escola Politécnica da Universidade de São Paulo, aos amigos e a tantas outras pessoas que atuaram de forma direta ou indireta na realização deste trabalho.

RESUMO

Investigou-se neste trabalho a proposta de um Modelo Lógico Gráfico para suporte à fase do Projeto Lógico em BDORs. O Modelo Lógico Gráfico proposto é uma extensão da UML para o Diagrama de Classes. A extensão deu-se por meio da elaboração de um Perfil UML, o qual foi disponibilizado em XMI para ser empregado em ferramentas CASE de diferentes fabricantes. Desenvolveu-se dois módulos para a ferramenta CASE ArgoUML. Esses módulos têm por finalidade a automação do desenvolvimento em BDORs, onde, a partir de um esquema gráfico, projetado utilizando-se o Modelo Lógico Gráfico proposto, gera-se código no padrão da SQL:2003 e no dialeto SQL utilizado pelo Oracle 11g. Foi proposta uma arquitetura baseada na ANSI/SPARC e na MDA para o projeto em BDORs que relaciona as fases do Projeto com as tecnologias empregadas para suportá-las. Por meio dessa arquitetura, destacam-se os pontos onde houve contribuição deste trabalho e os pontos que serão alvos de futuras pesquisas. Esta dissertação difunde os recursos existentes em BDORs e facilita a elaboração do Projeto Lógico em BDORs ao disponibilizar o modelo gráfico proposto e ao automatizar seu desenvolvimento na ferramenta CASE ArgoUML.

Palavras-chave: Banco de Dados Objeto-Relacional. Modelo Lógico Gráfico. Ferramenta CASE. Arquitetura em BDORs.

ABSTRACT

A Logical Graphic Model was proposed to support the Logical Design phase in ORDB. The Logical Graphic Model proposed is an extension of the UML Class Diagram. The extension was obtained by the elaboration of a UML Profile, which was released in XMI to be used in CASE tools from different manufacturers. Two modules were developed for ArgoUML CASE tool. These modules were designed to automate the development in ORDB, where, from a logical graphic, projected using the proposed Logical Graphic Model, creates the code standard SQL: 2003 and the SQL dialect used by Oracle 11g. An architecture based on the ANSI / SPARC and MDA for the project in ORDBs was proposed. This architecture associates the Project phases with the technologies used to support them. Through this architecture, the contributions of this paper and the subjects that will be target for future researches are highlighted. This dissertation diffuses the existing resources in ORDBs and facilitates the development of the Logical Design in ORDB, by disposing the proposed graphic model and automating its development in the ArgoUML CASE tool.

Keywords: Object-Relational Database. Logical Graphic Model. CASE tool. Architecture for ORDBs.

LISTA DE FIGURAS

Figura 1: Diagrama de Classes UML apresentado os elementos da SQL:2003 e seus inter-relacionamentos para suporte a BDOR (JUNIOR et al., 2009) ..24

Figura 2: Imagem que correlaciona os padrões utilizados pelos MDA (THE OBJECT MANAGEMENT GROUP, 2009) 26

Figura 3: Metamodelos e modelos envolvidos na geração do Modelo Lógico Gráfico Objeto-Relacional proposto neste trabalho. 30

Figura 4: Diagrama de Classes referente ao Esquema Conceitual (MARCOS et al., 2003) 41

Figura 5: Esquema Lógico de um BDOR utilizando a notação de Marcos et al. (MARCOS et al., 2003) 41

Figura 6: Janela principal do ArgoUML 53

Figura 7: Visão do Perfil ORDB proposto no *Explorer* do ArgoUML.....54

Figura 8: Aba Propriedades do painel Detalhes do ArgoUML. 54

Figura 9: Aba Propriedades do painel Detalhes do ArgoUML exibindo o atributo telefone. 55

Figura 10: Aba Fonte do Painel Detalhes. 55

Figura 11: Aba Estereótipo do Painel Detalhes. 56

Figura 12: Tela de seleção das classes para geração do *script*. 56

Figura 13: Arquitetura utilizada no módulo desenvolvido para geração do código em SQL:2003 58

Figura 14: Diagrama de pacotes do módulo para geração de código em SQL:2003 61

Figura 15: Exemplo de um Projeto Conceitual. 64

Figura 16: Exemplo de um Projeto Lógico Gráfico. 65

LISTA DE TABELAS

Tabela 1: Comparação do mapeamento de classes, atributos e operações.....35

Tabela 2: Comparação do mapeamento dos relacionamentos entre classes 36

Tabela 3: Estereótipos propostos por (MARCOS et al., 2003) para o Esquema Lógico de um BDOR. 38

Tabela 4: Estereótipos para o Modelo Lógico Gráfico.....43

Tabela 5: Mapeamento dos elementos da UML para o Projeto Físico de um BDOR.....44

Tabela 6: Estereótipos do Modelo Lógico Gráfico.....47

Tabela 7: Estereótipos para especificação de Métodos. 48

Tabela 8: Estereótipos para especificação de Restrições. 48

Tabela 9: Mapeamento do Projeto Lógico para o Físico de um BDOR. 49

Tabela 10: Código SQL correspondente ao esquema ilustrado na Figura 16. 67

LISTA DE ABREVIATURAS E SIGLAS

APOO	Análise e Projeto Orientado a Objetos
BD	Banco de Dados
BDR	Banco de Dados Relacional
CA	Atributo Composto
CASE	*Computer-Aided Software Engineering*
CLI	*Call Level Interface*
CWM	*Common Warehouse Metamodel*
DER	Diagramas Entidade-Relacionamento
EPOCA	Escola Potiguar de Computação e suas Aplicações
FK	*Foreign Key*
ID	Identificador
IDEF1X	*Integration Definition for Information Modeling*
ISO	*International Organization for Standardization*
IEC	*International Electrotechnical Commission*
MDA	*Model Driven Architecture*
MER	Modelo Entidade-Relacionamento
Modelo OR	Modelo Objeto-Relacional
MOF	*Meta-Object Facility*
OCL	*Object Constraint Language*
OMG	*Object Management Group*
OO	Orientação a Objetos
PIM	*Platform-Independent Model*
PK	*Primary Key*
PSM	*Persistent Stored Modules*
PSM	*Platform-Specific Model*
REF	Referência
SGBD	Sistema Gerenciador de Banco de Dados
SGBDOO	Sistema Gerenciador de Banco de Dados Orientados a Objetos
SGBDOR	Sistema Gerenciador de Banco de Dados Objeto-Relacional
SQL	*Structured Query Language*
UDT	*User Defined Type*

UML	*Unified Modeling Language*
XMI	*XML Metadata Interchange*
XML	*eXtensible Markup Language*

SUMÁRIO

1 Introdução 15

1.1 Motivação 15

1.2 Objetivo 16

1.3 Resultados Esperados 17

1.4 Metodologia 18

2 Fundamentação Teórica 20

2.1 Principais Aspectos dos Trabalhos Correlatos 20

2.2 Modelo Entidade-Relacionamento 22

2.3 Tecnologias e Modelos Empregados para Geração do Projeto Lógico de um BDOR 22

2.3.1 SQL e BDORs 22

2.3.2 Características da SQL:2003 relacionadas a OO. 23

2.4 Padrões de Modelagem de Sistemas 25

2.4.1 *Model Driven Architecture* (MDA) 25

2.4.2 *Unified Modeling Language* (UML) 27

2.4.3 XML *Metadata Interchange* (XMI) 28

2.5 Conclusões 28

3 Arquitetura proposta para o projeto de BDOR 29

3.1 Correlação da MDA, UML, XMI e SQL:2003 com a Arquitetura Proposta 29

3.2 Conclusões 31

4 Análise dos trabalhos (MARCOS et al., 2003) e (GOLOBISKY et al., 2005) para a composição do Modelo Lógico 33

4.1 Análise comparativa dos artigos (MARCOS et al., 2003) e (GOLOBISKY et al., 2005) 33

4.2 Análise do trabalho (MARCOS et al., 2003) para o aspecto do esquema gráfico 38

4.3 Síntese da Proposta deste Trabalho e dos Trabalhos de Marcos et al. (2003) e de Golobisky et al. (2005) 42

4.4 Conclusões 45

5 Modelo Lógico BDOR Proposto 46

5.1 Estereótipos para Especificação de Classes e Relacionamentos 46
5.2 Estereótipos para Especificação de Métodos 47
5.3 Estereótipos para Definição de Restrições ao Modelo 48
5.4 Mapeamento do Projeto Lógico para o Projeto Físico de um BDOR 49
5.5 Conclusões 50
6 Extensão proposta para o ArgoUML 51
6.1 Visão geral do ArgoUML 51
6.2 Resultado da Implementação no ArgoUML 52
6.3 Arquitetura do Módulo para Geração de Código em SQL:2003 57
6.4 Conclusões 61
7 Exemplo da Utilização da Ferramenta 63
7.1 Projeto Conceitual Desenvolvido para o Exemplo 63
7.2 Projeto Lógico Desenvolvido para o Exemplo 64
7.3 Projeto Físico Gerado para o Exemplo 66
7.4 Conclusões 69
8 Conclusão e Comentários 70
8.1 Sobre o Modelo Lógico Gráfico Proposto 70
8.2 Sobre os Módulos Desenvolvidos para o ArgoUML 71
8.3 Trabalhos futuros 72
Referências Bibliográficas 73
APÊNDICE A – Artigo Publicado na EPOCA 2009 76
APÊNDICE B – Artigo Publicado no SEMINCO 2010 82

1 Introdução

1.1 Motivação

A demanda por desenvolvimento de sistemas cada vez mais complexos para diversas áreas aumenta todos os anos. O advento da abordagem cujo enfoque é a Orientação a Objetos (OO) possibilitou a manipulação de elementos complexos em software.

A difusão de sistemas orientados a objetos, o descasamento de impedâncias entre o paradigma de objetos e o Modelo Relacional, aliado à necessidade de manipulação dos objetos da aplicação em base de dados levou ao surgimento de vários Sistemas Gerenciadores de Banco de Dados Orientados a Objetos (SGBDOOs) (SILBERSCHATZ et al., 2006). Esses produtos, porém acabaram não sendo largamente empregados devido a diversos motivos, dentre os quais: a dificuldade de integração com sistemas existentes e emprego de linguagens não tão padronizadas como a SQL (DATE, 2004).

Diante desses problemas e da necessidade de manipulação de objetos em banco de dados, os fornecedores de Sistemas Gerenciadores de Banco de Dados (SGBDs) relacionais optaram por estender seus produtos. Isso deu origem aos Sistemas Gerenciadores de Banco de Dados Objeto-Relacional (SGBDOR). O surgimento dos primeiros SGBDORs impulsionou a revisão da SQL.

Alguns conceitos de OO começaram a se tornar padrões em Banco de Dados (BD) a partir da especificação do Padrão SQL:1999 e foram consolidados com a especificação SQL:2003.

Apesar das vantagens provenientes dos BDORs, essa tecnologia em ambientes corporativos ainda não é muito utilizada. Isso ocorre por vários motivos, dentre eles a larga experiência e conhecimento de Banco de Dados Relacionais (BDR), além da existência de diversas ferramentas automatizadas que auxiliam no projeto das bases de dados relacionais.

A utilização da UML na modelagem conceitual em BD é uma alternativa (SOMMERVILLE, 2009; NAVATHE, 2009). Contudo, ressalta-se que a UML não possui uma notação específica para a modelagem de banco de dados. Mesmo assim, a UML pode ser usada para representar um modelo semântico de dados.

Devido ao fato do Modelo Objeto-Relacional (Modelo OR) apresentar aspectos da OO, a utilização da UML como meio da modelagem parece ser ainda mais justificada, se for considerado, por exemplo, que alguns elementos do Modelo OR como o Tipo Definido pelo

Usuário (UDT – *User Defined Type*), similarmente a objetos, também apresentam comportamento (métodos). No entanto, como discutido em (MARCOS et al., 2001b, 2003), as notações definidas pela UML não são capazes de representar os recursos da SQL:2003 com o nível de detalhes exigido por um Esquema Lógico Gráfico em Banco de Dados. Sendo assim, é necessária a criação de uma notação específica para a modelagem em BDORs. Dentre as opções, discutidas com mais detalhes no capítulo 2 (página 27), optou-se por estender a UML por meio de um Perfil UML.

Embora a modelagem para o Projeto Lógico de um BDOR seja alvo de intensas pesquisas (MARCOS et al., 2001a; GOLOBISKY et al., 2005; VARA et al., 2007), ainda não foi consolidada uma notação a ser utilizada. Faltam também ferramentas CASE (*Computer-Aided Software Engineering*) que considerem as novas estruturas disponíveis em BDOR para o suporte à modelagem de esquemas lógicos e com a geração automática de código para um determinado SGBDOR, como o Oracle, o PostgreSQL etc.

Por conta disso, propõe-se investigar os elementos que deverão compor o Modelo Lógico de um BDOR para a fase do Projeto Lógico em BDs. Almeja-se também o desenvolvimento de dois módulos para a ferramenta CASE ArgoUML: um módulo para geração automática de código, a partir do esquema lógico gráfico, em SQL:2003 e outro, no dialeto SQL utilizado pelo Oracle 11g.

1.2 Objetivo

Propostas para o Modelo Lógico em Banco de Dados Objeto-Relacional (BDOR) têm sido apresentadas; dentre elas destaca-se a utilização do Diagrama de Classes da UML o qual é estendido com novos estereótipos para suportar a representação dos elementos da OO descritos pelo padrão SQL (MARCOS et al., 2001a, 2001b, 2003; GOLOBISKY et al., 2005; VARA et al., 2007).

O objetivo deste trabalho é a proposta de um Modelo Lógico Gráfico para BDORs como extensão dos trabalhos (GOLOBISKY et al., 2005; MARCOS et al., 2003) discutidos no capítulo 4. Faz parte também do objetivo, o desenvolvimento de dois módulos para a ferramenta CASE ArgoUML. Esses módulos deverão disponibilizar o Modelo Lógico proposto por meio de um Perfil UML para a modelagem de esquemas lógicos dos usuários e serão responsáveis pela geração automática de código, a partir de um esquema lógico gráfico, na linguagem SQL:2003 e no dialeto SQL utilizado pelo Oracle 11g.

Dentre os motivos para utilização da ferramenta ArgoUML (melhor discutidos no capítulo 6), destacam-se o fato ser uma ferramenta *open source* e por fornecer suporte ao desenvolvimento de diagrama de classes UML.

Utiliza-se o padrão SQL:2003 para a especificação dos elementos do Modelo Lógico. Esta especificação considera as restrições que envolvem cada elemento definido pela SQL:2003 para o suporte à OO em Banco de Dados. Os módulos desenvolvidos para a ferramenta ArgoUML deverão possuir mecanismos para a validação da consistência do esquema desenvolvido pelo usuário considerando as restrições definidas pela SQL:2003, sendo que caso alguma irregularidade seja identificada o sistema deverá gerar avisos com o intuito de orientar o usuário a uma correta utilização dos elementos que compõem o Modelo Lógico proposto.

Optou-se por desenvolver os módulos para a geração automática de código no padrão da SQL:2003 e para o dialeto SQL utilizado pelo Oracle 11g, pois a SQL definida pelo padrão pode ser adaptada para qualquer dialeto SQL utilizado por um SGBDOR, como, por exemplo, o PostgreSQL, DB2 ou SQLServer; o Oracle 11g é um SGBD bem difundido e utilizado tanto em projetos acadêmicos quanto em projetos corporativos de pequeno a grande porte (ORACLE, 2010).

Além disso, o Oracle 11g dá suporte aos elementos definidos pelo padrão SQL e possibilita o desenvolvimento de uma interface Objeto-Relacional para uma base de dados puramente relacional. Essa interface é definida por meio de visões e cria uma camada de abstração entre a aplicação OO e a base de dados relacional. Dessa forma, possibilita que os recursos da SQL:2003 sejam utilizados por demanda, à medida que o projetista julgue necessários.

Para alcançar os objetivos deste trabalho, pretende-se a definição de uma arquitetura que faz uso da arquitetura ANSI/SPARC para representação dos modelos e metamodelos envolvidos para a geração dos modelos lógicos e físicos de BDOR. A arquitetura destina-se a contextualização das tecnologias envolvidas no desenvolvimento dos módulos para a ferramenta CASE ArgoUML, além de correlacionar as fases de desenvolvimento em BDs por meio de níveis de abstração e mapeamento que utiliza XMI.

1.3 Resultados Esperados

Espera-se com este trabalho o desenvolvimento de dois módulos para a ferramenta CASE ArgoUML que utilizam o Modelo Lógico gráfico consolidado por meio da investigação dos

trabalhos correlatos (MARCOS et al., 2001a, 2001b, 2003; GOLOBISKY et al., 2005; VARA et al., 2007), com o intuito de difundir e facilitar a utilização de BDORs. A experiência tem mostrado que a difusão de tecnologias está também relacionada ao suporte por ferramentas que facilitem/automatizem sua utilização.

Os módulos desenvolvidos deverão gerar automaticamente código em SQL:2003 e para o dialeto SQL utilizado pelo Oracle 11g a partir do esquema lógico gráfico do banco de dados desenvolvido pelo projetista. Sugestões serão exibidas conforme o projetista elabora um modelo, com o objetivo de orientá-lo à correta utilização dos elementos do perfil UML proposto.

Espera-se com a arquitetura proposta tornar o desenvolvimento de projeto de BDs mais interoperável, pois por meio do XMI será possível desvincular o projeto ou a fase de um projeto de uma ferramenta específica, possibilitando o intercâmbio entre diferentes ferramentas desde que as mesmas estejam de acordo com o padrão XMI.

1.4 Metodologia

Iniciou-se por uma investigação de trabalhos correlatos ao tema: Projeto em BDORs. Em paralelo, estudou-se a especificação da SQL considerando as versões SQL:1999 e SQL:2003. Esse estudo possibilitou o discernimento das principias diferenças existentes entre as versões analisadas do padrão. Os SGBDORs Postgres e Oracle foram investigados e comparados com o padrão SQL:2003. Como resultado, um artigo para divulgar esses estudos iniciais foi publicado (JUNIOR et al., 2009) e encontra-se disponível no apêndice A.

A fase inicial foi fundamental para a elaboração do segundo ao quarto capítulos desta dissertação.

Para o desenvolvimento dos módulos para o Esquema Lógico gráfico em BDORs, uma investigação da viabilidade de extensão das ferramentas CASE disponíveis foi realizada. Essa investigação também ocorreu em paralelo à fase de levantamento bibliográfico.

Observou-se que a ferramenta CASE utilizada neste trabalho necessitaria ser de fácil utilização e deveria possuir interface visual intuitiva. Além disso, outra característica de importância crucial, a ferramenta teria que ser extensível, possibilitando o desenvolvimento de módulos a serem incorporados posteriormente. Por conta disso, optou-se pelo ArgoUML, pois essa ferramenta CASE vai ao encontro das características desejadas conforme discutido no capítulo 6.

Depois da fase de pesquisa bibliográfica, iniciou-se a fase de desenvolvimento dos módulos. Nessa fase, deu-se maior ênfase na leitura de livros e artigos técnicos relacionados à engenharia de software e à ferramenta ArgoUML. A leitura de trabalhos correlatos foi mantida até a conclusão deste trabalho, obviamente com intensidade menor do que no início dos estudos.

Paralelamente às fases de pesquisa bibliográfica e desenvolvimento, a definição de uma arquitetura representativa dos modelos relacionados na geração do Modelo Lógico proposto foi produzida. Essa arquitetura faz uso da arquitetura ANSI/SPARC e possui correlação com a MDA e será discutida em detalhes no capítulo 3. Os módulos propostos seguem essa arquitetura.

Posteriormente à fase de desenvolvimento, iniciou-se a fase de teste dos módulos desenvolvidos. Nessa fase, diversos esquemas lógicos gráficos foram testados e o código gerado automaticamente para esses esquemas foi verificado utilizando-se a especificação da SQL:2003. Além disso, o código gerado no dialeto SQL utilizado pelo Oracle foi submetido ao SGBDOR Oracle 11g para validação da sua consistência. A ferramenta ArgoUML e os módulos desenvolvidos também foram utilizados em uma disciplina ministrada pela Orientadora, com o intuito de ilustrar de forma prática alguns conceitos relacionados à BDOR.

Selecionou-se um exemplo, dentre os modelos testados, para a composição do capítulo 7. Nesse capítulo, aborda-se um estudo de caso em que os elementos que compõem o modelo proposto são utilizados no esquema lógico gráfico desenvolvido para o exemplo. Os módulos desenvolvidos são utilizados para a geração automática de código em SQL:2003 e Oracle 11g. O código relacionado ao esquema lógico do exemplo é discutido no desenvolver do capítulo.

Publicou-se o artigo (CASTRO et al., 2010), que se encontra disponível no apêndice B, para divulgar o modelo lógico gráfico proposto e as técnicas de mapeamento utilizadas pelos módulos desenvolvidos para o ArgoUML. O artigo (CASTRO et al., 2010) utiliza o mesmo exemplo apresentado no capítulo 7 para ilustrar o conteúdo abordado.

Por fim, as conclusões obtidas no decorrer do trabalho foram sintetizadas no último capítulo, onde também são discutidas algumas questões em aberto que serão alvos de futuras pesquisas.

2 Fundamentação Teórica

Neste capítulo, discorre-se sobre os principais aspectos dos trabalhos correlatos e os conceitos e tecnologias utilizados no Modelo ORDB e no desenvolvimento dos módulos para a modelagem do Esquema Lógico de um BDOR no ArgoUML.

Inicia-se com os trabalhos correlatos, em seguida apresenta-se um breve histórico dos BDORs e uma visão geral da SQL:2003.

Abordam-se, na seqüência, as tecnologias MDA, UML e XMI que são gerenciadas pela OMG. Essas tecnologias são necessárias para a contextualização e o desenvolvimento do perfil UML para um BDOR; suas correlações com este trabalho serão descritas no capítulo seguinte, ao apresentar uma arquitetura que as utiliza.

Por fim, encerra-se o capítulo com as conclusões obtidas.

2.1 Principais Aspectos dos Trabalhos Correlatos

Nesta seção, abordam-se trabalhos correlatos, dentre os quais os artigos (MARCOS et al., 2003; GOLOBISKY et al., 2005) foram fundamentais para este trabalho por apresentarem, respectivamente, uma proposta de um Modelo para o Esquema Lógico de um BDOR e por identificar funções de mapeamento do Modelo Conceitual para o Esquema Físico.

O artigo (MARCOS et al., 2003) é uma revisão dos artigos (MARCOS et al., 2001a, 2001b). Dentre as modificações realizadas destacam-se a dos estereótipos «presistent» e «composes» que foram renomeados, respectivamente, para «Object Typed» e «Knows» no Esquema Lógico. Com exceção a algumas modificações de nomenclatura, o conteúdo da revisão permanece o mesmo dos originais acrescidos de discussão mais detalhada do mapeamento de relacionamentos.

A versão do padrão SQL utilizado por Marcos et al. (2003) não possui alguns elementos incorporados posteriormente ao padrão SQL, como, por exemplo, o tipo de dado *Multiset*; por conta disso alguns aspectos do Modelo proposto encontram-se desatualizados com relação à correspondência com o padrão SQL. Assim, faz-se necessário a investigação de novas formas de implementar os relacionamentos utilizando os novos recursos incorporados à SQL:2003.

Em (VARA et al., 2007) é apresentado uma arquitetura MDA para o projeto de um BDOR. Vara et al. utilizou-se da notação proposta por Marcos et al. (2003) para a descrição

de um Perfil UML e para apresentar as técnicas de mapeamento do Modelo Independente de Plataforma (PIM – *Platform Independent Model*) para o Dependente de Plataforma (PSM – *Platform Specific Model*). Contudo, as técnicas apresentadas não contemplam todos os possíveis relacionamentos existentes entre os elementos do Esquema de um BDOR.

Por meio da ferramenta CASE estendida consolidaram-se algumas das idéias discutidas por Marcos e Vara.

Funções de mapeamento do diagrama de classes da UML para objetos do Esquema Lógico de um BDOR são apresentadas em (GOLOBISKY et al., 2005). Sugestões foram feitas para o mapeamento, pois o trabalho de (GOLOBISKY et al., 2005) trata tipos de dados oferecidos pela SQL:2003 de forma muito restritiva. Maiores detalhes são abordados no capítulo 4 desta dissertação.

O artigo (PARDEDE et al., 2005) aborda qual foi o impacto ao processo de modelagem em Banco de Dados decorrente da inclusão de características da OO e de novos tipos de dados ao padrão SQL. O autor mostra como algumas dessas mudanças afetam o Modelo Conceitual, Lógico e Físico. Aborda-se também a implementação de novos objetos em alguns SGBDs.

O tipo de dados Referência foi incorporado ao padrão SQL, ele serve para armazenar uma referência às linhas de uma tabela tipada (MELTON, 2003b; PARDEDE et al., 2003). O autor de (EHLMANN, 2006) sugere que ao definir o domínio de um campo com o tipo Referência, seja declarada uma restrição de chave estrangeira, pois é possível excluir a tupla referenciada por um campo, sendo que o campo ao tentar acessar a tupla excluída irá obter o valor nulo.

As técnicas de mapeamento apresentadas neste trabalho consideram as restrições de chave estrangeira abordada por (EHLMANN, 2006) e discutem-se as ocasiões em que se aconselha a sua utilização.

O artigo (FEUERLICHT et al., 2009) discute algumas das funções de mapeamento existentes para a SQL:2003 e aponta situações em que a abordagem OO é preferível à abordagem Relacional, contudo as discussões não acrescentaram novidades em relação ao que já havia sido publicado pelos trabalhos aqui já citados.

2.2 Modelo Entidade-Relacionamento

O Modelo Entidade-Relacionamento (MER) criado para permitir a representação conceitual de objetos implementados em diferentes estruturas de SGBDs (hierárquicos, em rede e relacionais) tem sido largamente empregado desde sua criação por Peter Chen em 1976 (SILBERSCHATZ et al., 2006). A crescente utilização de SGBDs relacionais levou a geração de várias ferramentas CASE para a elaboração de Diagramas Entidade-Relacionamento (DER) para apoio a implementação de uma base de dados. Com o tempo, o emprego dominante de SGBDs relacionais conduziu não apenas ao surgimento de novas técnicas de modelagem de dados (ex, IDEF1X - *Integration Definition for Information Modeling* (KNOWLEDGE BASED SYSTEMS INC, 2011)) como também ao surgimento de novas ferramentas CASE para a elaboração de diagramas que estendem o DER para a representação de tabelas relacionais, chaves primárias e estrangeiras, relacionamentos, dentre outras características. A utilização de tais ferramentas permite a elaboração de projetos lógicos gráficos de BDRs. A representação gráfica é muito mais fácil de ser entendida do que um código (ou um *script*) numa determinada linguagem.

2.3 Tecnologias e Modelos Empregados para Geração do Projeto Lógico de um BDOR

2.3.1 SQL e BDORs

Aspectos da Orientação a Objetos foram incorporados pela ISO-IEC ao padrão SQL:1992 em 1999 dando origem a versão conhecida como SQL:1999. Os bancos de dados que estão de acordo com a SQL:1999 foram denominados: Banco de Dados Objetos-Relacionais (BDOR).

Desde então, com as versões SQL:2003 e mais recentemente SQL:2008, novos elementos foram acrescentados ao padrão para melhorar o suporte a OO.

Segundo (NAVATHE, 2009) o padrão SQL é atualmente composto dos seguintes componentes:

- SQL/*Framework*: descreve a parte conceitual do *framework* usada em outras partes da SQL para especificar a gramática e o resultado do processamento de declarações. A SQL/*Framework* também define termos e notações usados pelo padrão (MELTON, 2003a).

- SQL/*Foundation*: responsável pelo suporte aos novos tipos de dados, predicados, operações relacionais, cursores, regras e gatilhos, tipos definidos pelo usuário, capacidades de transações e rotinas armazenadas.
- SQL/*Bindings*: tem a SQL embutida e a Chamada Direta (*Direct Invocation*). A SQL embutida foi aperfeiçoada para incluir declarações de exceções adicionais.
- SQL/*Object*: inclui funcionalidades orientadas a objetos.
- SQL/*Temporal*: trata dados históricos, dados de seqüência temporais e outras extensões temporais.
- SQL/*Transaction*: formaliza a interface XA para uso pelos desenvolvedores da SQL.
- SQL/CLI (*Call Level Interface* – Interface em Nível de Chamada): fornece regras que permitem a execução do código da aplicação sem fornecer o código-fonte, evitando a necessidade de pré-processamento.
- SQL/PSM (*Persistent Stored Modules* – Módulos de Armazenamento Persistente): especifica as funcionalidades para particionar uma aplicação entre um cliente e um servidor. Seu objetivo é melhorar o desempenho pela minimização do tráfego de rede.

Maiores detalhes com relação às características e elementos que compõem os BDORs serão dados ao longo deste trabalho.

2.3.2 Características da SQL:2003 relacionadas a OO.

Para melhor entendimento dos elementos disponíveis na especificação SQL:2003, um modelo de classes foi elaborada conforme apresentado a seguir na Figura 1. Tal modelo fez parte do artigo publicado por este e outros autores (JUNIOR et al., 2009).

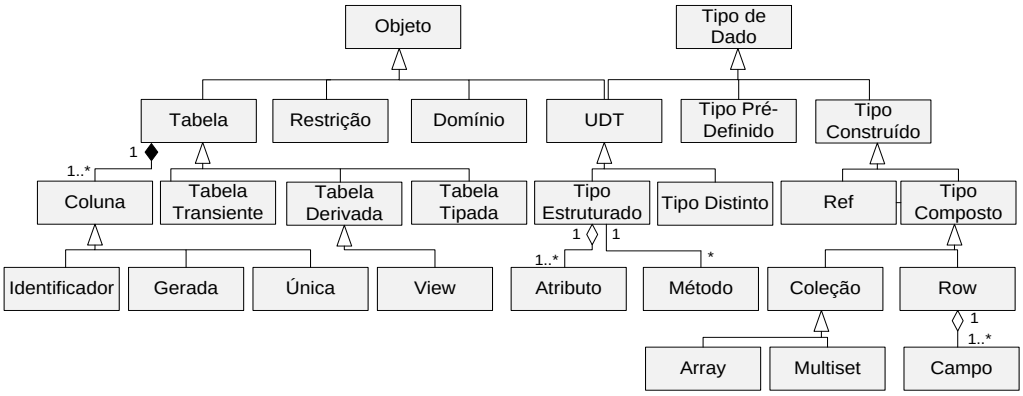

Figura 1: Diagrama de Classes UML apresentado os elementos da SQL:2003 e seus inter-relacionamentos para suporte a BDOR (JUNIOR et al., 2009)

A Figura 1 apresenta os elementos existentes na SQL:2003 e suas correlações por meio de uma ontologia. O diagrama da Figura 1 é dividido em duas subierarquias, uma que traz os tipos de dados e outra, os tipos de objetos que podem ser definidos.

Os Tipos de Dados são classificados em três classes principais: Tipo Pré-Definido, Tipo Construído e Tipo Definido pelo Usuário (UDT - *User Defined Type*).

Conforme Figura 1, os tipos construídos consistem em: tipos atômicos, como o REF (referência), tipos complexos, como ROW, ou ainda coleções, como ARRAY e MULTISET. Um ROW é uma estrutura composta por um ou mais campos.

Um tipo REF é um tipo de dado construído que corresponde a um valor que faz referência a (ou aponta para) um valor do tipo referenciado. Os únicos valores que podem ser referenciados são as linhas de tabelas tipadas. Daqui decorre que qualquer tipo de referência é um tipo estruturado.

Um ARRAY é uma coleção ordenada de valores, cujos elementos são referenciados pela posição ordinal e possui limite de armazenamento estipulado no momento da declaração. Já o tipo MULTISET é uma coleção não ordenada de valores, cujo espaço para armazenamento é alocado dinamicamente.

Segundo a Figura 1, um Tipo Definido pelo Usuário (UDT) pode ser da categoria Tipo Distinto ou Tipo Estruturado. Um tipo distinto corresponde a um UDT que é baseado em algum tipo pré-definido. Os valores de um tipo distinto são representados pelos valores do tipo em que se baseia.

Um Tipo Estruturado é composto por um ou mais atributos, pode gerar hierarquias supertipo-subtipo, e também podem ser definidos métodos para manipular o Tipo Estruturado a que se refere. Um Tipo Estruturado pode ser utilizado para definir uma tabela tipada, ou para especificar o domínio de campos de uma tabela.

De acordo com a Figura 1, um Objeto pode ser classificado em uma das seguintes classes: Tabela, Restrição, Domínio e UDT. Restrição e Domínio já vêm de especificações anteriores da SQL e, portanto, já são largamente conhecidos.

A classe Tabela (Figura 1) pode ser especializada nas classes Transiente, Derivada e Tipada. Uma tabela é composta por uma ou mais colunas. O conceito de tabela e suas especializações, Transiente e Derivada, são o mesmo que já existia nas especificações anteriores da SQL e, portanto, não serão aqui detalhadas.

Tabelas tipadas podem ser especializadas; para que uma tabela tipada possa estender outra tabela tipada, os tipos estruturados associados a essas tabelas devem pertencer à mesma hierarquia de especialização. Por exemplo, seja TT_1 uma tabela tipada associada ao tipo estruturado TE_1 e TT_2 uma tabela tipada associada ao tipo estruturado TE_2. A tabela tipada TT_1 poderá estender a TT_2 se, e somente se, TE_2 for supertipo direto do TE_1, ou seja, TE_2 deve aparecer na cláusula de generalização presente na especificação de TE_1 (JUNIOR et al., 2009).

2.4 Padrões de Modelagem de Sistemas

A *Object Management Group* (OMG), fundada em 1989, é um consórcio sem fins lucrativos formado por centenas de organizações. A OMG é responsável pelo desenvolvimento de padrões para a modelagem de sistemas e processos de negócio visando o desenvolvimento de software orientado a objetos. Ela também provê padrões para a criação de ferramentas de software visando tornar os produtos, de diferentes fabricantes, interoperáveis. Dentre as especificações que a OMG coordena, neste trabalho utilizam-se: MDA (*Model Driven Architecture*) (OMG, 2003), UML (*Unified Modeling Language*) (OMG, 2009) e XMI (*XML Model Interchange*) (OMG, 2007). O conteúdo completo sobre a história da OMG, bem como os outros projetos em que está envolvida, pode ser encontrado em (THE OBJECT MANAGEMENT GROUP, 2009).

2.4.1 *Model Driven Architecture* (MDA)

A Arquitetura Orientada a Modelo (MDA) é uma metodologia de desenvolvimento de aplicações e escrita de especificações que utiliza um Modelo Independente de Plataforma

(PIM[1]) da aplicação ou sistema, obtido da especificação de funcionalidade e comportamento do negócio. A MDA define um processo baseado na criação de modelos em diferentes níveis de abstração, separando os interesses da aplicação da arquitetura a ser implementada (OMG, 2003).

Uma MDA completa consiste em um modelo base, independente de plataforma, e um ou mais Modelos Dependentes da Plataforma (PSM[2]) e um conjunto de definições de interface, cada qual descrevendo como o modelo base é mapeado para uma determinada plataforma.

O desenvolvimento MDA foca primeiro na funcionalidade e comportamento de uma aplicação ou sistema, sem se preocupar com as especificidades da plataforma, ou plataformas, na qual será desenvolvido. Desta forma, MDA separa os detalhes de implementação das funções do negócio. Sendo assim, não é necessário repetir o processo de definição da funcionalidade e comportamento de uma aplicação ou sistema cada vez que uma nova tecnologia surgir. Com a MDA, a funcionalidade e o comportamento são modelados uma única vez. O mapeamento do PIM para um PSM, fornecido pela plataforma MDA, pode ser realizado por ferramentas de modo a facilitar o suporte a novas ou diferentes tecnologias.

A MDA pode utilizar outros padrões definidos pela OMG, conforme apresentado pela Figura 2, entre eles: MOF, UML, XMI etc. As seções seguintes destinam-se a descrição dos padrões de interesse deste trabalho.

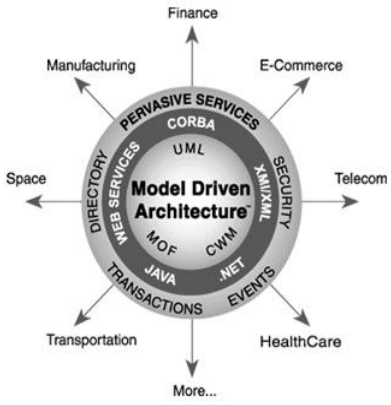

Figura 2: Imagem que correlaciona os padrões utilizados pelos MDA (THE OBJECT MANAGEMENT GROUP, 2009)

[1] PIM é o acrônimo para *Platform Independent Model*
[2] PSM é o acrônimo para *Platform Specific Models*

2.4.2 *Unified Modeling Language* (UML)

A Linguagem de Modelagem Unificada (UML) é uma linguagem visual para especificação, construção e documentação de artefatos dos sistemas. É uma linguagem de propósito geral que pode ser usada com todos os principais métodos OO e componentes e pode ser aplicada em todos os domínios da aplicação (saúde, finanças, Telecom, indústria aérea etc.) e plataformas (J2EE, .NET etc.) (OMG, 2009).

Segundo (OMG, 2009) a UML pode ser estendida de duas maneiras:

1) Um novo dialeto UML pode ser definido utilizando Perfis (*Profiles)* para estender a linguagem para uma plataforma particular, por exemplo, J2EE/EJB, .NET/COM+, e domínio, por exemplo, finanças, telecomunicações etc.
2) Uma nova linguagem relacionada à UML pode ser especificada pela reutilização de parte do pacote *InfrastructureLibrary*.

O primeiro caso define um novo dialeto UML e o segundo define um novo membro da família de linguagens UML.

O pacote *Profiles* contém os mecanismos que permitem que metaclasses de metamodelos existentes sejam estendidas para se adequarem a diferentes propósitos. Por meio do pacote *Profiles* é possível adequar o metamodelo UML para diferentes plataformas (OMG, 2009).

Neste trabalho, utilizou-se o pacote *Profiles* como mecanismo para adequar o metamodelo UML aos elementos que são específicos do domínio Objeto-Relacional (OR) conforme definido pela SQL:2003. As extensões realizadas à UML foram agrupadas em um *perfil*[3], tal como especificado pela OMG (OMG, 2009). Dentre os motivos que influenciaram nessa escolha, pode-se citar:

- O diagrama de classe da UML mostra-se adequado para o desenvolvimento de modelos conceituais de BDs (MARCOS et al., 2003; NAVATHE, 2009; SOMMERVILLE, 2009).
- O perfil corresponde a um meio simples de extensão da UML, possibilitando a definição dos estereótipos necessários para o esquema lógico em BDORs.
- Reutilização de uma infra-estrutura já existente, sendo esta bem difundida, consolidada e gerenciada.

[3] Neste trabalho utiliza-se a palavra *perfil* como tradução da definição UML de *profile*.

2.4.3 XML *Metadata Interchange* (XMI)

XMI é um padrão mantido pela OMG para definição, intercâmbio, manipulação e integração de dados e objetos por meio da Linguagem de Marcação Extensível (XML) (OMG, 2007). Padrões baseados em XMI são aplicados em ferramentas de integração, repositórios, aplicações e *Data Warehouses*.

O XMI provê regras por meio das quais um esquema pode ser gerado para qualquer metamodelo baseado em MOF[4] (*Meta-Object Facility*). A UML utiliza como metalinguagem de definição a MOF, sendo assim, é possível converter um Esquema UML em XMI. Um esquema XMI tem como principal benefício à portabilidade entre diferentes ferramentas CASE, desde que estas estejam de acordo com o padrão (OMG, 2007).

O perfil proposto neste trabalho será utilizado pelos módulos desenvolvidos para a ferramenta CASE ArgoUML, além de ser disponibilizado no formato XMI, com o intuito de torná-lo portável.

2.5 Conclusões

Neste capítulo, discutiu-se de forma sucinta as tecnologias utilizadas neste trabalho: UML, XMI e MDA.

Dentre as opções de extensão da UML, optou-se pelo desenvolvimento de um perfil UML para a geração de esquemas lógicos para BDORs. Essa escolha foi feita com o intuito de reutilizar a infraestrutura UML já existente, a qual é bem consolidada e difundida. O perfil foi disponibilizado através da extensão do ArgoUML. Pensando na portabilidade dos módulos desenvolvidos para o ArgoUML, a implementação dos mesmos foi também disponibilizada em formato XMI.

No capítulo seguinte, correlacionam-se as tecnologias apresentadas à forma com que serão utilizadas neste trabalho.

---

[4] Consulte (OMG, 2006) para obter uma descrição completa a respeito da MOF.

3 Arquitetura proposta para o projeto de BDOR

Neste capítulo, as tecnologias discutidas no capítulo anterior são relacionadas por meio da proposta de uma arquitetura para as fases de Projeto em BDORs. A arquitetura proposta baseia-se na arquitetura ANSI/SPARC e faz uso das fases de Projeto de BDs: Projeto Conceitual, Lógico e Físico (DATE, 2004; SILBERSCHATZ et al., 2006). Propõe-se a utilização de XMI como meio para o mapeamento entre os níveis de abstração, pois, como descrito no capítulo anterior, o XMI é um formato padrão que pode ser utilizado para o intercâmbio de modelos entre ferramentas CASE.

3.1 Correlação da MDA, UML, XMI e SQL:2003 com a Arquitetura Proposta

Nesta seção, discute-se a correlação existente entre a MDA, UML, XMI e SQL:2003 com o uso da arquitetura ANSI/SPARC para representação dos modelos e metamodelos envolvidos para a geração dos Modelos Lógicos e Físicos de BDOR.

A utilização de diagramas de classe UML na modelagem Conceitual em BD é uma alternativa ao MER (NAVATHE, 2009) que permite o uso de um único modelo para o registro dos objetos, tanto persistentes quanto transientes. Isso facilita a fase de desenvolvimento, já que elimina o esforço da construção de um modelo específico para a representação dos objetos persistentes, como também o da verificação da consistência necessária entre os modelos gerados.

Além da opção do uso de diagramas de classe como o modelo conceitual, optou-se também pela extensão do mesmo para a criação de esquemas lógicos gráficos para BDORs. O diagrama de classes UML já dispõe de elementos para representação de objetos com sua parte estrutural e comportamental, além dos vários tipos de relacionamentos e suas restrições intrínsecas da OO. Associado a isso, tem-se ainda a larga aceitação e uso dessa linguagem para representação dos vários aspectos e geração de artefatos tanto para um software quanto para um sistema OO. Concluiu-se que a opção pela extensão do Diagrama de Classe UML seria adequada e o esforço para a representação de elementos exclusivos de BDORs seria minimizado. Tal extensão foi realizada por meio da criação de um Perfil UML. Outros trabalhos seguiram também a opção feita aqui (MARCOS et al., 2001a, 2001b, 2003; GOLOBISKY et al., 2005; VARA et al., 2007).

No esquema da Figura 3, mostram-se os metamodelos e os modelos envolvidos na geração do Modelo Lógico Objeto-Relacional aqui proposto. No esquema, níveis de abstração associados ao projeto de uma base de dados são usados para apresentar os modelos e especificações correspondentes aos níveis conceitual e interno da arquitetura ANSI/SPARC. Destaca-se ainda, na Figura 3, o paralelo da arquitetura ANSI/SPARC com a arquitetura MDA. Se considerarmos que, a partir de um modelo de classe, podemos obter a geração do modelo lógico (nível conceitual / lógico) e a geração do modelo físico (nível interno), este último representado pelo código SQL específico para criação da base de dados num SGBD (Oracle, Postgre, SQLServer etc.), então podemos dizer que um ciclo de projeto de uma base dados reflete uma arquitetura MDA.

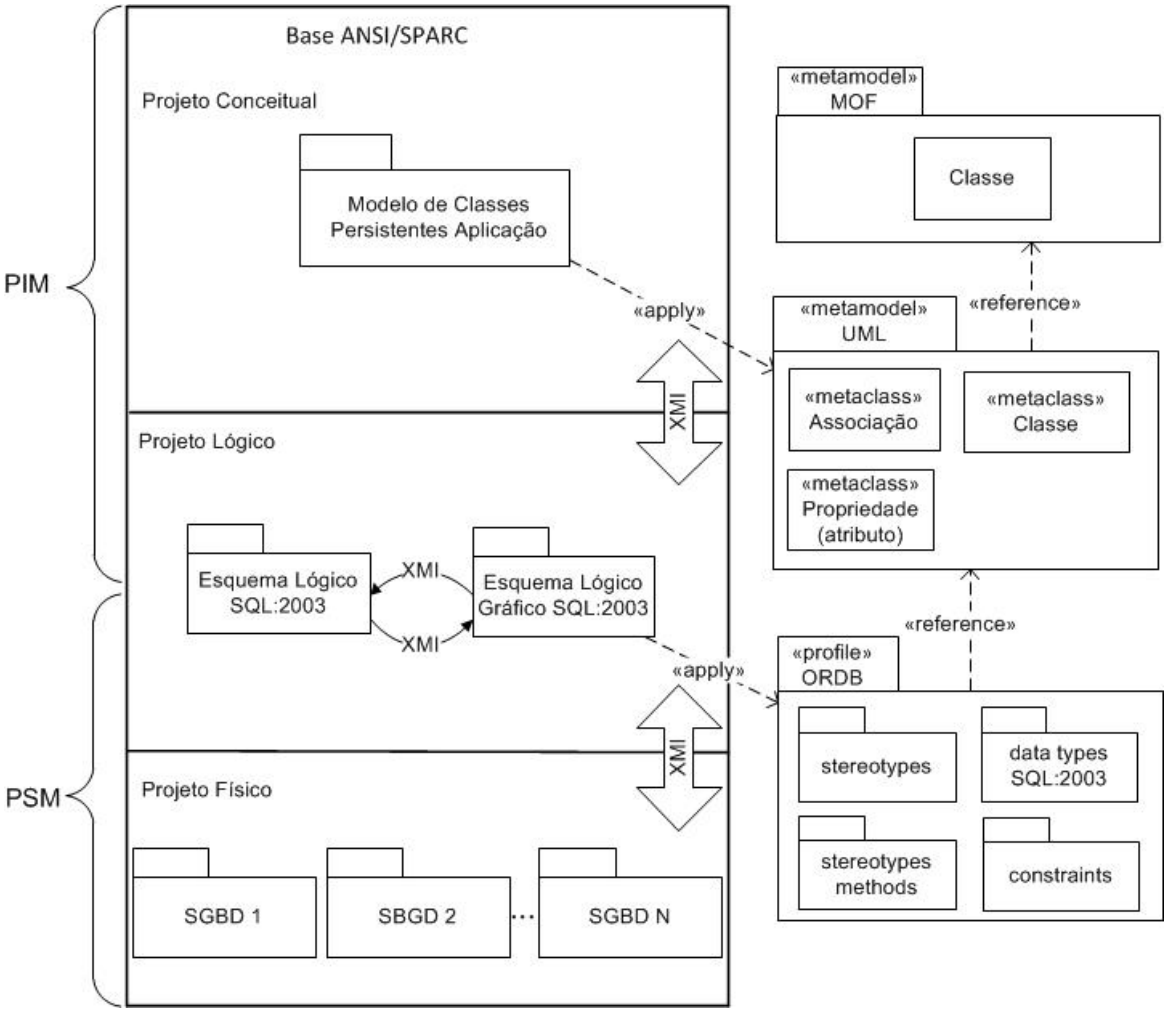

Figura 3: Metamodelos e modelos envolvidos na geração do Modelo Lógico Gráfico Objeto-Relacional proposto neste trabalho.

Ainda na Figura 3 observa-se uma intersecção do Modelo Independente de Plataforma (PIM) com o Modelo Dependente de Plataforma (PSM) no nível do Projeto Lógico. O PIM é

caracterizado por ser independente de uma tecnologia específica; nesse modelo o projetista preocupa-se apenas com os aspectos relacionados ao domínio do negócio a ser modelado. No PSM considera-se algum tipo de tecnologia, sendo que no contexto do projeto de um BDOR o PSM pode ser desenvolvido visando-se algum SGBDOR. O nível do Projeto Lógico ilustrado na Figura 3 não se refere a algum SGBOR (tal como o Projeto Físico). Contudo, nesta proposta utiliza-se a SQL:2003 que se trata de uma especificação padrão para uma tecnologia específica: BDOR. Dado a dificuldade de discernimento entre classificar o nível do Projeto Lógico como sendo um PSM ou um PIM, justifica-se, tal como ilustrado na Figura 3, a classificação desse nível como sendo a interseção desses modelos.

Na arquitetura proposta, o XMI é utilizado como meio para o mapeamento em todos os níveis de abstração, conforme apresentado na Figura 3. O Projeto Conceitual, por ser definido utilizando-se o diagrama de classes da UML, é facilmente traduzido para XMI, pois a UML, por ser uma extensão da MOF (Figura 3), está de acordo com o padrão XMI (OMG, 2009). O mesmo acontece com o esquema lógico do SGBDOR que, após ser definido, poderá ser convertido para o padrão XMI para depois ser mapeado para o esquema físico. Tanto o esquema conceitual quanto o lógico em XMI poderão ser transportados para outras ferramentas CASE, caso o projetista julgue necessário, pois uma das vantagens do XMI é a interoperabilidade.

O perfil ORDB corresponde à extensão da UML e os elementos definidos no perfil em conjunto com os da UML formam o Modelo Lógico Gráfico utilizado para o desenvolvimento de Esquemas Gráficos dos usuários (lado esquerdo da Figura 3). Visando a separação lógica dos elementos, o perfil desenvolvido foi organizado em quatro subpacotes: *stereotypes*, *stereotypes methods*, *constraints* e *data types SQL:2003*. Esses subpacotes são detalhados no capítulo 5.

3.2 Conclusões

Neste capítulo, apresentou-se a proposta de uma arquitetura baseada na arquitetura ANSI/SPARC para representação dos modelos e metamodelos envolvidos para geração dos esquemas lógicos e físicos de BDOR. Essa arquitetura utiliza o XMI para o mapeamento entre os níveis e intercâmbio de dados.

A arquitetura proposta apresenta o inter-relacionamento existente entre as tecnologias discutidas no capítulo anterior (capítulo 2) e por meio desta arquitetura foi possível

contextualizar e evidenciar os pontos em que houve contribuição deste trabalho. Relacionam-se a seguir as contribuições deste trabalho discutidas no decorrer capítulo.

1. Definição de um perfil por meio da extensão da UML (lado direito da Figura 3). Esse perfil corresponde aos elementos pertencentes ao Modelo Lógico Gráfico que define representações para os elementos incorporados à SQL:2003 para suporte a OO. O referido perfil será disponibilizado nos módulos desenvolvidos para o ArgoUML e também no formato XMI, com o intuito de torná-lo utilizável em outras ferramentas CASE.
2. Definição de Diretrizes de Mapeamento. As técnicas de mapeamento abordadas neste trabalho correspondem à tradução do Esquema Lógico Gráfico para o código SQL:2003 (e vice-versa), que se encontram no nível referente ao Projeto Lógico, para o código SQL do Oracle 11g, pertencentes ao nível do Projeto Físico.
3. Proposta de uma arquitetura que se baseia na arquitetura ANSI/SPARC, a qual utiliza o XMI como meio para o mapeamento entre os níveis de abstração em Projetos de BDs.

4 Análise dos trabalhos (MARCOS et al., 2003) e (GOLOBISKY et al., 2005) para a composição do Modelo Lógico

Os artigos (MARCOS et al., 2003; GOLOBISKY et al., 2005) foram fundamentais para a proposta deste trabalho e para a definição do perfil UML.

A proposta de (MARCOS et al., 2003) sugere o uso de estereótipos para a especificação de um modelo lógico gráfico, propõe também formas de mapeamento para SBGDs. Por outro lado, o trabalho de (GOLOBISKY et al., 2005) não considera a geração de um modelo lógico gráfico, concentra-se na construção de esquemas lógicos para SGBDOR. Para tal, define funções de mapeamento de um modelo conceitual, expresso por diagramas de classe UML, para objetos de SGBDORs.

Esses trabalhos apresentam divergências com relação a algumas definições. Por conta disso, neste capítulo realiza-se uma comparação desses artigos, destacando-se os pontos positivos e negativos, e mostram-se como os elementos foram abordados no desenvolvimento do Modelo Lógico aqui proposto.

Primeiro discutem-se as funções de mapeamento apresentadas por (MARCOS et al., 2003), (GOLOBISKY et al., 2005) e as consideradas neste trabalho. Em seguida, analisa-se apenas o trabalho de (MARCOS et al., 2003) à respeito dos estereótipos pertencentes ao Projeto Lógico de um BDOR. Na terceira seção, uma síntese das principais diferenças existentes entre as propostas de Marcos et al. (2003), Golobisky et al. (2005) e as deste trabalho são apresentadas. Encerra-se este capítulo com as conclusões obtidas.

4.1 Análise comparativa dos artigos (MARCOS et al., 2003) e (GOLOBISKY et al., 2005)

No artigo de (GOLOBISKY et al., 2005) foram identificadas funções de mapeamento do Diagrama de Classes da UML para o esquema lógico de BDOR com base na SQL:2003. Também foi definido um modelo de mapeamento constituído de duas fases. Na primeira fase executa-se o que segue:

1. Para toda classe definida num modelo de classes deve-se criar a um UDT (tipo estruturado).
2. Cada classe pode apresentar atributos: simples, compostos ou multivalorados. Assim, o UDT correspondente à classe deve apresentar atributos com, respectivamente, tipo

primitivo (*build-in*), tipo linha (*row type*) e/ou tipo coleção (*Array* ou *Multiset*), já definidos na seção 2.3.2 deste trabalho.

3. Cada classe pode apresentar métodos, então se devem definir métodos correspondentes no UDT relacionado.

O modelo obtido da primeira fase é utilizado como entrada para a segunda fase, onde cada UDT, gerado na fase anterior, dá origem a uma tabela tipada.

Em (GOLOBISKY et al., 2005) é sugerido que toda classe origine uma tabela tipada, considera-se que esta abordagem generalista pode não ser a mais adequada para certos modelos de classes que apresentam particularidades associadas às regras de negócio da aplicação a que se refere: por exemplo, se considerarmos que classes abstratas foram definidas no modelo de classes. Tais classes não devem originar instâncias, portanto, a criação de uma tabela tipada associada a tal classe representará desperdício de espaço, além dos problemas decorrentes da gerência e manutenção da base de dados, que pode apresentar, com o decorrer do tempo, um número significativo de tabelas que não apresentam dados. Por outro lado, também investigando possíveis particularidades de um modelo de classes de uma aplicação, entende-se que a geração do modelo lógico deve ser flexível, permitindo ao desenvolvedor e/ou projetista a decisão se uma classe deve ser representada por um UDT (tipo estruturado), uma tabela tipada, uma tabela convencional (BDR), um tipo linha (*row type*), ou ainda apenas por um UDT (sem a criação de uma tabela tipada). O mapeamento para um tipo linha é sugerido para os casos onde não se pretende definir métodos associados à estrutura.

Semelhante ao trabalho de (GOLOBISKY et al., 2005), em (MARCOS et al., 2003) as classes também são mapeadas em UDTs (tipos estruturados) e tabelas tipadas. Além disso, Marcos et al. (2003) propõe um esquema gráfico, formado por estereótipos, para representar o esquema lógico em BDORs. Em tal proposta as classes persistentes são transformadas em classes marcadas com o estereótipo «Object Type». Uma classe persistente em UML é marcada com o estereótipo «persistente». Para transformar uma classe persistente em uma classe SQL:1999 (versão utilizada por Marcos et. al.), é necessário definir um UDT, e a tabela tipada associada.

A Tabela 1 apresentada a seguir resume o mapeamento para classes, atributos e operações proposto por esses trabalhos.

Tabela 1: Comparação do mapeamento de classes, atributos e operações

Elemento no Diagrama de Classes	**Correspondente no Esquema Lógico de um BDOR**	
	(GOLOBISKY et al., 2005)	**(MARCOS et al., 2003)**
Classe	UDT e Tabela Tipada	UDT e Tabela Tipada
Atributo simples	Tipo Primitivo (*build-in*)	Tipo Primitivo (*build-in*)
Atributo composto	Tipo Linha	Tipo Linha/UDT
Atributo multivalorado	*Array* e *Multiset*	*Array*
Métodos	Métodos de um UDT	Métodos de um UDT

Em relação à Tabela 1, considera-se que:

1. Os tipos de dados utilizados para definir o domínio de atributos simples (ou básico) de um esquema lógico gráfico encontram-se no pacote *data types* SQL:2003 (Figura 3). Esses tipos de dados estão de acordo com os tipos definidos pela SQL:2003. Dessa forma, o mapeamento de atributos simples proposto por ambos os trabalhos será mantido.
2. A proposta de Marcos et al. (2003) para o mapeamento de atributos compostos utiliza as estruturas compostas definidas pelo padrão: tipo linha e UDT. Contudo, ao definir-se um tipo linha (*row*), este fica disponível para utilização, similar aos tipos básicos pré-definidos. Por outro lado, a associação de um UDT a atributos permitiria, por exemplo, a definição de operações, mas atributos de classes UML não têm atrelados a eles operações. Estas são definidas para a classe. Assim, entende-se que isso introduz uma complexidade que não existe no diagrama de classes, não devendo ser adotada. Atributos compostos devem ser mapeados para atributos, cujo tipo de dado deve ser linha (*row*), ou para UDT (tipo estruturado), porém neste caso, sem métodos associados.
3. Atributos multivalorados permitem que uma coleção de valores possam ser registrados para o mesmo e devem ser acessados de forma unitária. Tal coleção pode ser limitada (apresentar um número finito de valores) quanto ilimitada (apresentar um número infinito, ou desconhecido, de valores). Cabe ao desenvolvedor especificar essa restrição. Assim, neste trabalho, propõe-se que atributos multivalorados sejam mapeados para atributos cujos tipos sejam *Arrays* ou *Multiset* (coleções). Essa abordagem é similar a adotada por Golobisky et al. (2005).
4. Diferente da proposta dos trabalhos da Tabela 1, propõe-se que métodos de classes possam ser mapeados para métodos de UDTs, mas o desenvolvedor deve fazer essa indicação explicitamente. Considera-se aqui que uma classe poderá ser mapeada para uma tabela convencional, para uma tabela tipada, para um Tipo Linha e/ou para um UDT, conforme discutido anteriormente. Assim, mesmo que o mapeamento corresponda a uma tabela tipada e/ou a um UDT, caberá ainda ao desenvolvedor definir quais métodos devem ser mapeados para métodos de objetos na base de dados. Entende-se que esta

flexibilidade é fundamental no projeto lógico da base de dados, pois uma série de questões está envolvida nesse processo, dentre eles o desempenho da aplicação. Cabe ao desenvolvedor ou projetista a decisão sobre a forma de mapeamento de métodos.

A Tabela 2 compara os trabalhos (GOLOBISKY et al., 2005; MARCOS et al., 2003) em relação às técnicas de mapeamento dos relacionamentos pertencentes ao Diagrama de Classes para o Esquema Lógico de um BDOR.

Tabela 2: Comparação do mapeamento dos relacionamentos entre classes

Associação		**(GOLOBISKY et al., 2005)**	**(MARCOS et al., 2003)**
Agregação	1..1	Define-se uma referência da classe todo para a classe parte e uma referência da classe parte para a classe todo.	Define-se um *Array* na classe todo que armazena referências da classe parte.
	1..x[5]	Define-se um *Array* de referências na classe agregadora (classe todo).	
	1..*	Define-se um *Multiset* de referências na classe agregadora (classe todo).	Define-se um *Array* de Referências na classe agregadora (classe todo).
Composição	1..1	Define-se um UDT na classe agregadora (classe todo).	Define-se um *Array* de Referências na classe agregadora (classe todo). O controle de restrições é realizado por meio de *checks*, *assertions* e *triggers*.
	1..x	Define-se um *Array* de UDT na classe agregadora (classe todo).	Define-se um *Array* de Referências na classe agregadora (classe todo). O controle de restrições é realizado por meio de *checks*, *assertions* e *triggers*.
	1..*	Define-se um *Multiset* de UDT na classe agregadora (classe todo).	Define-se um *Array* de Referências na classe agregadora (classe todo). O controle de restrições é realizado por meio de *checks*, *assertions* e *triggers*.
Associação Binária		Cada classe mantém referência(s) para a classe associada (referência cruzada).	
	1..1	Define-se uma Referência.	
	1..x	Define-se um *Array* de Referências.	
	1..*	Define-se um *Multiset* de Referências.	Define-se um *Array* de Referências.
Associação Unidirecional		Similar ao bidirecional, no entanto a referência(s) irá existir em apenas uma tabela.	
Associação N-ária (três ou mais classes)		Define-se um UDT com o nome do papel da associação. O UDT referencia as classes envolvidas e será utilizado para definir uma tabela tipada.	Não contemplada
Classe Associativa		Define-se um UDT para a classe de associação e fazê-lo referenciar cada classe associada. O UDT será utilizado para definir uma tabela tipada.	Não contemplada
Generalização-Especialização		Define-se um UDT para cada classe da hierarquia.	Realiza-se generalização de UDTs e de Tabela Tipadas.

[5] x é um número inteiro qualquer e refere-se ao limite superior de ocorrências conhecido pelo projetista. Ele é definido aqui para diferenciar do símbolo * (limite superior desconhecido).

Em relação à Tabela 2, considera-se que:

1. A proposta de Golobisky et al. para mapeamento da agregação é mais adequada por não fazer uso desnecessário de *Array* na associação com multiplicidade 1..1 e por utilizar o *Multiset* quando a multiplicidade for 1..*, pois o *Multiset* utiliza alocação dinâmica de espaço, sendo apropriado seu uso quando o limite superior de ocorrências for desconhecido.
2. O relacionamento de composição cria um vínculo forte entre a classe todo e suas partes, pois caso a classe todo deixe de existir, as suas partes também deverão deixar de existir. Golobisky et al. propõem manter as classes partes como tabelas aninhadas, ou seja, definir as partes por meio de um campo cujo domínio é UDT (tipo estruturado), com o intuito de facilitar a tarefa de manutenção referencial nas operações de exclusão, pois ao se excluir uma tupla no todo, automaticamente exclui-se todas as partes (tabelas aninhadas) associadas. Essa abordagem não é adequada, pois pode gerar níveis profundos de tabelas aninhadas, e dessa forma comprometer o desempenho da aplicação, além disso, essa abordagem dificulta o desenvolvimento de consultas à base de dados, pois para acessar um campo de uma tabela aninhada é necessário referenciar a tabela que a contém.

   Neste trabalho o relacionamento de composição é tratado de forma similar ao de agregação, cabendo ao projetista o desenvolvimento de *triggers* para garantir o vínculo existente entre o todo e as partes. Tal proposta foi cunhada por Marcos et al. (2003).
3. As propostas de mapeamento das associações binária e unidirecional são adequadas por tratar-se de mapeamentos bem difundidos e aceitos em ferramentas CASE para geração de código a partir do modelo visual. Sendo que no caso da associação com multiplicidade 1..*, a proposta de Golobisky et al. (2005) é mais apropriada por utilizar uma estrutura de dados de alocação dinâmica de espaço.
4. No modelo relacional um possível mapeamento para o relacionamento N-ário é a criação de uma tabela com o mesmo nome do relacionamento para armazenar as chaves (chaves estrangeiras) das tabelas envolvidas no relacionamento. A proposta de Golobisky et al. (2005) baseia-se nesta forma de mapeamento. Considera-se essa proposta adequada por ser uma forma já difundida de mapeamento do modelo relacional.
5. A proposta de Golobisky et al. (2005) para o mapeamento de classes associativas mostra-se adequada por centralizar as referências, entre as classes envolvidas na associação, na classe associativa. Esse mapeamento é similar ao mapeamento do relacionamento N para N no modelo relacional, o qual se cria uma tabela associativa para armazenar as chaves estrangeiras e com isso diminui-se o cardinalidade para 1 para N.

6. A proposta de Marcos et al. (2003) para o mapeamento da associação generalização-especialização é mais adequada por utilizar o recurso de extensão da SQL:2003, por esse motivo será adotado neste trabalho.

4.2 Análise do trabalho (MARCOS et al., 2003) para o aspecto do esquema gráfico

Em (MARCOS et al., 2003) são propostos, além do já apresentado, novos estereótipos UML para o Diagrama de Classes; a Tabela 3 resume os principais estereótipos discutidos por ele.

Tabela 3: Estereótipos propostos por (MARCOS et al., 2003) para o Esquema Lógico de um BDOR.

Estereótipo	**Classe do Metamodelo a qual o estereótipo é aplicado.**	**Descrição**
«udt»	Classe	Corresponde ao Tipo Definido pelo Usuário o qual deverá ser utilizado na definição do domínio de atributos.
«Object Type»	Classe	Corresponde a uma tabela Tipada definida a partir de um UDT. O UDT que origina a tabela tipada fica vinculado somente a esta.
«Knows»	Associação	A associação «Knows» é uma relação especial que une uma classe a um UDT. É uma relação unidirecional. A direção da associação é representada por uma seta que aponta para o UDT usado pela classe.
«REF»	Atributo	Um «REF» representa uma referência para algum «Object Type».
«Array»	Atributo	Representa uma estrutura ordenada por índices.
«row»	Atributo	Representa uma estrutura composta por elementos, cada um deles podendo ser de diferentes domínios.
«PK»	Atributo	Restrição de chave primária.
«FK»	Atributo	Restrição de chave estrangeira.
«NOT NULL»	Atributo	Restrição *not null*.
«Unique»	Atributo	Restrição de valor único.
«Check»	Atributo	Restrição referente ao domínio do negócio.
«redef»	Método	O estereótipo «redef» foi definido para ser aplicado em métodos herdados por uma subclasse com o intuito de atribuir um comportamento diferente do apresentado na superclasse.
«def»	Método	A implementação de um método que possui o estereótipo «def» é diferida para ser realizada por suas subclasses.

Em relação à Tabela 3, considera-se que:

1. Na proposta de Marcos et al. (2003), o estereótipo «udt» é aplicado em classes as quais serão utilizadas na definição do domínio de atributos, contudo um UDT também pode ser utilizado na definição de tabelas tipadas. Almejando tornar o esquema lógico gráfico de acordo com a SQL:2003, neste trabalho a classe marcada com o estereótipo «udt» poderá ser utilizada na definição do domínio de atributos e para definir tabelas tipadas.
2. Marcos et al. (2003) propõem o estereótipo «Object Type» para a definição de tabelas tipadas que possuem um UDT implicitamente vinculado. O estereótipo «Object Type» não possibilita o reuso, pois a representação de um UDT e de uma tabela tipada por meio de uma única notação não viabiliza o aproveitamento do UDT na definição de outras tabelas tipadas ou no domínio de atributos. Por conta disso, neste trabalho representa-se de forma separada um UDT (por meio do estereótipo «udt») e uma tabela tipada (por meio do estereótipo «typed table»). Para vincular a tabela tipada ao UDT, propõe-se uma associação marcada com o estereótipo «define».
3. O estereótipo «Knows» é aplicado em associações sempre que existir algum tipo de dependência entre duas classes. Se existirem dependências mútuas entre duas classes, por exemplo, geradas devido a uma associação no modelo de classes, então duas associações «Knows», cada uma direcionada para uma das classes, são representadas no diagrama do esquema lógico gráfico. Isto torna tal diagrama visualmente pesado, ficando difícil de ser entendido (Figura 4 e Figura 5). Assim, entende-se que isso introduz uma complexidade que não existe no diagrama de classes, não devendo ser adotado.
4. Marcos et al. (2003) propõem os estereótipos «REF», «Array» e «row» para a definição do domínio de atributos no esquema lógico. A especificação do domínio de um atributo por meio de um estereótipo introduz mais uma complexidade gráfica desnecessária no modelo lógico gerado. Além disso, um atributo para concretizar o relacionamento no modelo gráfico pode ser implicitamente representado por meio de associações e suas multiplicidades do diagrama gerado. Dado os argumentos expostos, esses estereótipos não deverão ser adotados.

O tipo linha, como discutido em seção 2.3.2 (página 23), corresponde a uma estrutura composta por um ou mais campos, similar ao UDT quando utilizado para definir o domínio de um atributo, contudo o tipo linha não possui comportamento (métodos) em sua definição. Sugere-se utilizar o estereótipo «row type» aplicado em classe (tal qual o «udt»), ao invés de aplicá-lo em atributos como proposto por Macos et al. Isso possibilitará

o reuso, pois um mesmo tipo linha poderá ser reutilizado para definir o domínio de atributos em diferentes classes.

5. Marcos et al. (2003) utilizam os estereótipos «PK», «FK», «NOT NULL», «Unique» e «Check» para definir restrições em atributos de classes no esquema lógico gráfico. Esses estereótipos serão adotados neste trabalho, pois correspondem a possíveis restrições ao esquema definidas pela SQL:2003.
6. Marcos et al. (2003) definem dois estereótipos para serem aplicados em métodos de classes: «def» e «redef». O estereótipo «redef» representa o conceito de polimorfismo, que na SQL:2003 é indicado pela palavra reservada *overriding*. Aqui, utiliza-se a definição da SQL:2003. O estereótipo «def» corresponde ao conceito de método abstrato, cuja implementação é diferida para ser realizada por suas subclasses. Como tal conceito não está presente na especificação da SQL:2003, considerou-se tal estereótipo desnecessário ao modelo.

Para ilustrar a utilização dos estereótipos propostos por Marcos et al. (2003), retirou-se um exemplo de seu trabalho que corresponde ao mapeamento do esquema conceitual (Figura 4) para o esquema lógico gráfico (Figura 5) de um BDOR.

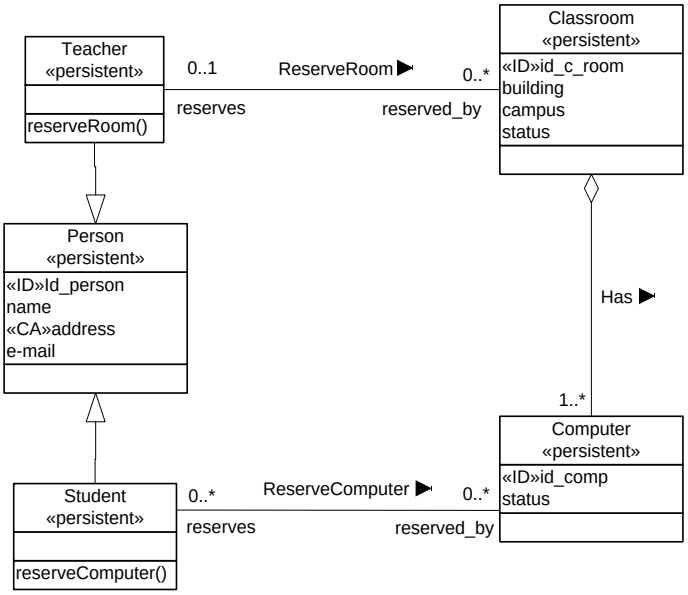

Figura 4: Diagrama de Classes referente ao Esquema Conceitual (MARCOS et al., 2003)

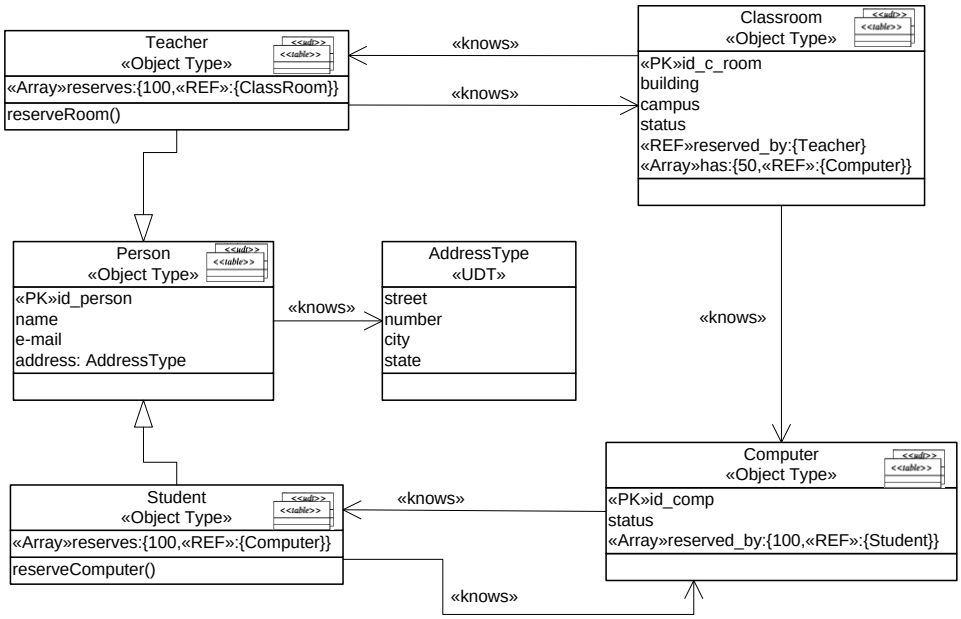

Figura 5: Esquema Lógico de um BDOR utilizando a notação de Marcos et al. (MARCOS et al., 2003)

No esquema conceitual (Figura 4) os autores utilizam o estereótipo «persistent» que é um estereótipo definido pela UML para diferenciar classes que serão persistidas das classes transientes. Eles também utilizam o estereótipo «ID» em identificadores de classes e o estereótipo «CA» em atributos compostos.

No mapeamento, as linhas representando as associações com multiplicidade foram substituídas pelos tipos de dados dos atributos REF. Assim, no modelo lógico BDOR deve-se

prestar atenção nesses tipos de dados para reconhecer as multiplicidades da associação. Por outro lado, foi incluída também a associação «Know»: a associação «Know» é representada tantas vezes existirem REF. Por exemplo: em *Teacher*, o atributo *reserves* é um *Array* de REFs para *Classroom*, então existe um «Know» apontando para *Classroom*. Também existe outro «Know» apontando para *Teacher*, por causa do atributo *reserved-by* (em *Classroom*) que é do tipo REF [*Teacher*] – ou seja, o que no modelo conceitual é representado por uma única linha (associação) na proposta gera duas linhas direcionadas. Um aspecto negativo dessa abordagem é que o modelo lógico final fica com vários «Know», pois cada associação gera pelo menos um «Know». Outro ponto, as agregações também são mapeadas por «Know» e neste caso, um único atributo é incluído na classe pai, seu tipo vai ser determinado pela multiplicidade associada às subclasses parte.

O Projeto Lógico proposto por Marcos et al. (2003) (Figura 5) pode gerar confusão, pois ocorre indicação explícita dos atributos do tipo REF e dos relacionamentos «Know» , ou seja, utilizam-se dois jeitos diferentes para representar a mesma coisa.

No Projeto Conceitual (Figura 4) o relacionamento é representado em um único local. Na Figura 5, o relacionamento é representado em ambos os lados, como feito em linguagem de programação.

4.3 Síntese da Proposta deste Trabalho e dos Trabalhos de Marcos et al. (2003) e de Golobisky et al. (2005)

Nesta seção, apresenta-se uma síntese das principais diferenças existentes entre as propostas de Marcos et al. (2003), Golobisky et al. (2005) e as deste trabalho. As tabelas 4 e 5, a seguir, representam de forma concisa os conceitos descritos pelo modelo proposto, destacando-se as contribuições obtidas.

Abordam-se primeiramente as diferenças entre os Modelos Lógicos de um BDOR apresentados nesses trabalhos. Uma explicação mais detalhada, com relação ao Modelo aqui proposto, será dada no capítulo que segue. Encerra-se esta seção apresentando as diferenças entre as propostas de mapeamento para o Projeto Físico de um BDOR.

A Tabela 4 apresenta na primeira coluna os estereótipos propostos por Marcos et al. (2003) e na segunda, os utilizados neste trabalho. Na última coluna, descrevem-se as diferenças entre nas notações.

Tabela 4: Estereótipos para o Modelo Lógico Gráfico.

Marcos et al.	**Este Trabalho**	**Diferença**
«udt»	«udt»	Neste trabalho, o UDT definido no Esquema Lógico pode ser utilizado na especificação do domínio de campos e na definição de tabelas tipadas (como especificado na SQL:2003), diferente da proposta do Marcos et al. (2003) que utiliza-o apenas no domínio de campos.
«Object Type»	_	Por representar uma tabela tipada e seu UDT associado impossibilitando a reutilização do UDT, o estereótipo «Object Type» não foi incorporado ao Modelo aqui proposto.
«Knows»	_	O estereótipo «Knows» não será incorporado ao Modelo aqui proposto, pois torna o diagrama visualmente pesado, ficando difícil de ser entendido. No modelo proposto, uma única linha de associação, coerentemente com o modelo de classes UML, é utilizada para representar o relacionamento.
«REF»	_	Referências são representadas por meio das linhas de associação entre classes, dessa forma, torna o estereótipo «REF» desnecessário ao Modelo aqui proposto.
«Array»	_	Neste trabalho, não se utiliza de estereótipos para definição de domínios de campos, pois acrescenta uma complexidade visual desnecessária ao Esquema Lógico Visual. As linhas de associações e a definição de sua multiplicidade são usadas pelo modelo proposto para especificar domínios do tipo *array* ou *row*.
«row»	«row type»	O estereótipo «row type» é aplicado em classe (tal qual o «udt») ao invés de ser aplicado em atributos como proposto por Macos et al, dessa forma, poderá ser reutilizado para definir o domínio de atributos em diferentes classes.
«Table»	«table»	Utiliza-se dessa notação para representa uma tabela.
_	«typed table»	Utiliza-se dessa notação para representar uma Tabela Tipada.
_	«define»	O estereótipo «define» é utilizado para associar uma Tabela Tipada ao UDT que a define.
_	«static»	No Modelo aqui proposto, utiliza-se a notação UML para os tipos de métodos definidos pela SQL:2003.
_	«constructor»	
_	«instance»	
«redef»	«overriding»	Utiliza-se *overriding* ao invés de *redef*, coerente com a definição da SQL:2003.
«def»	_	Como o conceito de método abstratos não está presente na especificação da SQL:2003, considerou-se tal estereótipo desnecessário ao modelo.
«PK»	«pk»	Por corresponderem às restrições definidas pela SQL:2003, esses estereótipos serão incorporados ao Modelo aqui proposto.
«FK»	«fk»	
«NOT NULL»	«not null»	
«Unique»	«unique»	
«Check»	«check»	

A Tabela 5 apresenta na primeira coluna os elementos da UML que serão mapeados para o Projeto Físico de um BDOR. Na segunda coluna, apresentam-se, respectivamente, as propostas de mapeamento de Marcos et al. (2003) e de Golobisky et al. (2005). Na última coluna apresenta-se a abordagem adotada neste trabalho.

Tabela 5: Mapeamento dos elementos da UML para o Projeto Físico de um BDOR.

Elementos UML	**Marcos et al./ Golobisky et al.**	**Este Trabalho**
Classe	UDT e Tabela Tipada	UDT, Tipo Linha, Tabela e Tabela Tipada.
Classe abstrata	–	UDT (sem a criação de uma tabela tipada)
Atributo simples	Tipo Primitivo (*build-in*)	Tipo Primitivo (*build-in*).
Atributo composto	Tipo Linha e UDT /Tipo Linha	Tipo Linha e UDT sem métodos associados.
Atributo multivalorado	*Array* / *Array* e *Multiset*	*Array* e *Multiset*
Métodos	Métodos de um UDT	Métodos de um UDT indicados pelo desenvolvedor.
Agregação	Define-se um *Array* de Referências na classe agregadora (classe todo) / Define-se uma referência da classe todo para a classe parte e uma referência da classe parte para a classe todo utilizando um *Array* ou *Multiset* quando a multiplicidade for superior a 1.	Define-se uma referência da classe todo para a classe parte e uma referência da classe parte para a classe todo utilizando um *Array* ou *Multiset* quando a multiplicidade for superior a 1.
Composição	Define-se *Array* de Referências na classe agregadora. O controle de restrições é realizado por meio de *checks*, *assertions* e *triggers*/ Define-se um UDT, um *Array* de UDT ou um *Multiset* de UDT na classe agregadora dependendo da multiplicidade do relacionamento.	Define-se *Array* de Referências na classe agregadora. O controle de restrições é realizado por meio de *checks*, *assertions* e *triggers*.
Associação Binária	Cada classe mantém referência(s) para a classe associada (referência cruzada) utilizando o tipo *Array* dependendo da multiplicidade da associação/ Idem a Marcos et al., contudo utiliza o tipo *Array* e *Multiset* quando necessário.	Cada classe mantém referência(s) para a classe associada (referência cruzada) utilizando o tipo REF, *Array* ou o *Multiset* dependendo da multiplicidade da associação.
Associação Unidirecional	Similar ao bidirecional, no entanto a referência(s) irá existir em apenas uma tabela.	Similar ao bidirecional, no entanto a referência(s) irá existir em apenas uma tabela.
Associação N-ária (três ou mais classes)	Não contemplada/ Define-se um UDT com o nome do papel da associação. O UDT referencia as classes envolvidas e será utilizado para definir uma tabela tipada.	Define-se um UDT com o nome do papel da associação. O UDT referencia as classes envolvidas e será utilizado para definir uma tabela tipada.
Classe Associativa	Não contemplada/ Define-se um UDT para a classe de associação e fazê-lo referenciar cada classe associada. O UDT será utilizado para definir uma tabela tipada.	Define-se um UDT para a classe de associação e fazê-lo referenciar cada classe associada. O UDT será utilizado para definir uma tabela tipada.
Generalização-Especialização	Realiza-se generalização de UDTs e de Tabela Tipadas/ Define-se um UDT para cada classe da hierarquia.	Realiza-se generalização de UDTs e de Tabela Tipadas.

4.4 Conclusões

Neste capítulo, discutiram-se os trabalhos de (GOLOBISKY et al., 2005) e (MARCOS et al., 2003), os quais foram fundamentais para a composição de um Modelo Lógico para BDORs. Ao longo deste capítulo, uma análise foi realizada, levantando-se pontos positivos e negativos das duas propostas. Sugestões foram também apresentadas e adotadas neste trabalho, em contrapartida aos pontos considerados não adequados nas propostas dos trabalhos (GOLOBISKY et al., 2005; MARCOS et al., 2003).

Dessa forma, o modelo proposto incorpora os pontos considerados apropriados de ambos os trabalhos correlatos e possui melhorias para tornar sua utilização mais flexível, permitindo que o projetista defina a representação que melhor atenda a sua realidade, sem que o modelo force a criar elementos desnecessários ao projeto, que podem inclusive comprometer a legibilidade dos diagramas.

O Modelo Lógico proposto neste trabalho foi consolidado por meio de um Perfil UML, esse perfil foi incorporado aos módulos desenvolvidos para geração automática de código SQL no ArgoUML. O Modelo proposto será apresentado com maior detalhe no próximo capítulo.

5 Modelo Lógico BDOR Proposto

Neste capítulo, propõe-se um modelo lógico para a modelagem gráfica de um BDOR; tal modelo foi concebido como uma extensão dos trabalhos discutidos no capítulo anterior, visando melhorar os pontos fracos e propondo alternativas para as questões não abordadas pelos trabalhos correlatos. O modelo aqui apresentado deve atender ao projeto lógico gráfico de um BDOR, pois esse modelo será disponibilizado em uma ferramenta CASE para geração automática de código em SQL:2003 e Oracle 11g.

Este capítulo foi dividido em quatro seções, sendo que as três primeiras apresentam, respectivamente, os pacotes *stereotypes*, *stereotypes methods* e *constraints* pertencentes ao Perfil ORDB desenvolvido neste trabalho (conforme ilustrado na Figura 3 do capítulo 3). Essa divisão aumenta a coesão das seções, dado que os pacotes agrupam elementos relacionados do modelo. A seguir, descreve-se o conteúdo de cada seção:

Na primeira seção, discutem-se os estereótipos para especificação de classes e relacionamentos, esses estereótipos encontram-se no pacote *stereotypes* do Perfil ORDB. Na seqüência, os estereótipos para especificação de métodos pertencentes ao pacote *stereotypes methods* são apresentados. Na terceira seção, abordam-se os estereótipos utilizados para definição de restrições ao modelo, esses estereótipos encontram-se no pacote *constraints*. O pacote *datatypes SQL:2003* do Perfil ORDB (Figura 3) não será detalhado, pois corresponde ao que está definido na especificação SQL:2003. Na quarta seção, é apresentado o mapeamento dos estereótipos pertencentes ao Projeto Lógico para o Projeto Físico de um BDOR. Por fim, encerra-se o capítulo com as conclusões obtidas.

5.1 Estereótipos para Especificação de Classes e Relacionamentos

No pacote *stereotypes* (Figura 3, seção 3.1), encontram-se os estereótipos aplicados em classes e relacionamentos do esquema lógico gráfico conforme detalhado na Tabela 6. Nessa tabela, a primeira coluna corresponde ao nome do estereótipo, a segunda descreve o elemento da UML em que o estereótipo se aplica e a última coluna descreve o significado do estereótipo.

Tabela 6: Estereótipos do Modelo Lógico Gráfico

Estereótipo	**Elemento Base**	**Descrição**
«table»	Classe	Define uma tabela (NAVATHE, 2009).
«udt»	Classe	Define um Tipo Definido Pelo Usuário: pode ser composto por um ou mais atributos, ter métodos e gerar hierarquias supertipo-subtipo
«typed table»	Classe	Define uma Tabela Tipada: uma tabela definida a partir de um UDT.
«row type»	Classe	Define um Tipo Linha: estrutura composta por um ou mais campos.
«define»	Associação	Define uma Tabela Tipada a partir de um UDT.

Os estereótipos «table», «udt», «typed table» e «row type» (tabela 6) foram concebidos com base nos elementos definidos pela SQL:2003. O estereótipo «define» é utilizado para relacionar um UDT a uma Tabela Tipada, esse relacionamento pode ser lido da seguinte forma: "o UDT X define a Tabela Tipada Y" ou "a Tabela Tipada Y é definida pelo UDT X". Esse estereótipo tem sentido semântico, ou seja, agrega um significado ao relacionamento entre um UDT e uma Tabela Tipada.

5.2 Estereótipos para Especificação de Métodos

A SQL:2003 disponibiliza recursos para a definição de métodos em um UDT muito próximos do que se tem no ambiente da aplicação; são eles: métodos de instância, estático e construtores. Estes são representados no modelo lógico proposto por estereótipos, conforme Tabela 7. Além disso, é possível sobrescrever algum método da classe pai na classe filha possibilitando o polimorfismo, usando para isso a palavra reservada *overriding*. O Perfil proposto deve possuir elementos para identificar (ou diferenciar) os métodos de uma classe no Modelo Lógico. Tal notação, além de fornecer uma fácil visualização e compreensão do esquema, permite também a distinção dos tipos de métodos, possibilitando dessa forma a geração automática de código para um determinado esquema.

Métodos de objetos são funções ou procedimentos que são declarados na definição de um UDT e definem o comportamento desejado para objetos desse tipo. A seguir, apresenta-se a descrição sucinta dos tipos de métodos suportados pela SQL:2003.

Métodos de Instância: fornecem acesso aos dados de um objeto ou definem as operações que um aplicativo executa sobre os dados. Para executar um método de instância, o aplicativo chama o método desejado do objeto.

Métodos Estáticos: servem para comparar instâncias de objetos e realizar operações que não usam os dados de algum objeto em particular, mas, ao contrário, são globais para um tipo de objeto.

Métodos Construtores: Um método construtor padrão, que não possui parâmetros em sua definição, é definido implicitamente para cada tipo de objeto, a menos que seja substituído por um construtor definido pelo usuário. Um método construtor é chamado sobre um tipo para criar uma instância do objeto.

Tabela 7: Estereótipos para especificação de Métodos.

Estereótipo	**Elemento Base**	**Descrição**
«static»	Method	Define um método estático em um UDT.
«constructor»	Method	Define um construtor em um UDT.
«instance»	Method	Define um método de instância. Quando nenhum estereótipo for especificado no esquema lógico, os módulos desenvolvidos para o ArgoUML assumem que o método é de instância.
«overriding»	Method	Utilizado para redefinir (sobrescrever) um método herdado.

A Tabela 7 apresenta os elementos pertencentes ao pacote *stereotypes methods* (Figura 3, seção 3.1). A primeira coluna corresponde ao nome do estereótipo, a segunda descreve o elemento da UML em que o estereótipo se aplica e a terceira coluna descreve o significado do estereótipo

5.3 Estereótipos para Definição de Restrições ao Modelo

O pacote *constraints* (Figura 3, seção 3.1) refere-se às restrições definidas pela SQL:2003 para colunas de tabelas. Como a informação de restrições deve fazer parte de um projeto lógico de uma base de dados, definiram-se os estereótipos apresentados na Tabela 8.

Tabela 8: Estereótipos para especificação de Restrições.

Estereótipo	**Elemento Base**	**Descrição**
«pk»	attribute	A restrição de chave primária designa uma ou mais colunas como chave primária de uma tabela.
«fk»	attribute	Uma restrição de chave estrangeira (associada à regra de integridade referencial) estabelece uma relação entre a chave estrangeira e uma chave primária ou única.
«not null»	attribute	Um campo definido com a restrição *not null* fica impossibilitado de aceitar valores nulos.
«unique»	attribute	A restrição *unique* indica que o valor da coluna (ou colunas) deve ser exclusivo (não se repetir para outro objeto da tabela).
«check»	attribute	Uma restrição de verificação permite que se especifique uma condição que cada objeto da tabela deve satisfazer. Para satisfazer a restrição, cada objeto da tabela deve fazer a condição ser verdadeira (*TRUE*) ou desconhecida (devido a um valor nulo).

A primeira coluna da Tabela 8 corresponde ao nome do estereótipo, a segunda descreve o elemento da UML em que o estereótipo se aplica e a terceira coluna descreve o significado do estereótipo

5.4 Mapeamento do Projeto Lógico para o Projeto Físico de um BDOR

Nas seções anteriores deste capítulo, foram apresentados os estereótipos que compõem o Modelo Lógico de um BDOR. Nesta seção, apresenta-se o mapeamento desses estereótipos para o código SQL pertencente ao Projeto Físico (Figura 3, seção 3.1).

Observa-se que a partir do modelo lógico proposto é possível gerar o Projeto Físico de um BDOR; contudo, o código SQL gerado irá variar conforme o SGBDOR utilizado. O SGBDOR (PostgreSQL, SQLServer etc.) pode ter, ou não, suporte a determinado elemento do Perfil ORDB; além disso, o código gerado para determinado elemento do perfil pode diferenciar-se do código definido na especificação da SQL.

A Tabela 9 apresenta o mapeamento dos estereótipos do Perfil ORDB para o respectivo código em SQL:2003 e Oracle 11g pertencente ao Projeto Físico de um BDOR.

Tabela 9: Mapeamento do Projeto Lógico para o Físico de um BDOR.

Estereótipo	**SQL:2003**	**Oracle 11g**
«table»	Tabela	Tabela
«udt»	UDT	UDT
«typed table»	Tabela Tipada	Tabela Tipada
«row type»	Tipo Linha	UDT
«define»	Cláusula da Tabela Tipada[1]	Cláusula da Tabela Tipada
«static»	STATIC	
«constructor»	CONSTRUCTOR	
«instance»	INSTANCE	MEMBER
«overriding»	OVERRIDING	
«pk»	PRIMARY KEY	
«fk»	REFERENCES	
«not null»	NOT NULL	
«unique»	UNIQUE	
«check»	CHECK	

[1] A Cláusula da Tabela Tipada corresponde ao seguinte código em SQL:2003 e Oracle 11g: <nome da tabela> OF <nome do UDT>, pois toda tabela tipada está associada a um UDT, mas não o contrário.

Os estereótipos «table», «udt» e «typed table» foram concebidos com base nos elementos definidos pela SQL:2003, por conseguinte, o mapeamento para a SQL:2003 e Oracle 11g pode ser feito de forma direta. Contudo, devido ao fato do Oracle 11g não dar suporte ao Tipo Linha, o estereótipo «row type» não possui mapeamento direto, tal como os estereótipos citados, para esse SGBDOR. O módulo desenvolvido relacionado ao Oracle 11g

sugerirá, por meio de mensagens, o emprego do UDT no lugar do Tipo Linha, contudo cabe ao projetista a decisão final na utilização, ou não, do UDT. Cabe ressaltar que, neste caso, o UDT não deve ter comportamento, pois o Tipo Linha não possui métodos na sua definição.

Devido ao Oracle 11g estar de acordo com a SQL:2003 na definição de Tabelas Tipadas, o mapeamento do estereótipo «define», pertencente ao projeto lógico, para o código SQL, pertencente ao projeto físico, pode ser feito tal como definido na especificação.

Observa-se que o Oracle 11g dá suporte à sobrescrita de métodos de instância («overriding») e a todos os tipos de métodos especificados pela SQL:2003. A única divergência existente é a forma de declaração dos métodos de instância em que a SQL:2003 utiliza a palavra *instance* e o Oracle 11g, *member*.

Com relação aos estereótipos para definir restrições ao modelo, o Oracle 11g segue a especificação da SQL:2003, por conseguinte, não há variação na forma de mapeamento dos estereótipos.

5.5 Conclusões

Neste capítulo, apresentou-se a proposta de um modelo para o projeto lógico gráfico de BDORs. O modelo aqui proposto corresponde a uma extensão da UML feita por meio da criação de um Perfil UML denominado Perfil ORDB. O modelo proposto possui notações específicas para definição dos diferentes tipos de métodos e elementos disponibilizados pela SQL:2003.

Os estereótipos que constituem o Modelo ORDB foram agrupados em pacotes, cada qual com funções e aplicações bem definidas. Os pacotes apresentados fazem parte do Perfil ORDB, o qual foi materializado utilizando o padrão XMI e será disponibilizado nesse formato e nos módulos para a geração automática de código em SQL:2003 e no dialeto SQL utilizado pelo Oracle 11g.

6 Extensão proposta para o ArgoUML

Os módulos desenvolvidos utilizam o perfil UML proposto e têm por objetivo aferir a validade das técnicas de mapeamento discutidas nos dois capítulos anteriores a este. O mapeamento refere-se à tradução automática do Esquema Lógico, que se encontra no segundo nível da arquitetura apresentada na Figura 3 (seção 3.1)**,** para código SQL pertencente ao Projeto Físico de um BDOR. As linguagens contempladas neste trabalho para geração de código foram: SQL:2003 e a linguagem SQL do Oracle 11g.

Os dois módulos desenvolvidos correspondem a extensões da ferramenta CASE ArgoUML. Por meio deles, o Perfil ORDB foi incorporado aos elementos pertencentes ao Diagrama de Classes do ArgoUML com o intuito de disponibilizá-lo para ser utilizado na modelagem de esquemas lógicos gráficos.

A seguir, apresenta-se a forma como este capítulo foi dividido.

Na primeira seção, discute-se sucintamente a história do ArgoUML, em seguida, aborda-se as funcionalidades dos módulos desenvolvidos para o ArgoUML. Na seção 3, apresenta-se a arquitetura de software utilizada para o desenvolvimento dos módulos e justificam-se os padrões de projeto empregados. Por fim, encerra-se o capítulo com as conclusões obtidas.

6.1 Visão geral do ArgoUML

O ArgoUML é uma ferramenta CASE utilizada para a modelagem gráfica de sistemas de software; ele foi projetado e desenvolvido por Jason Elliot Robbins durante seu Doutorado na Universidade da Califórnia na cidade de Irvine, Califórnia dos EUA. A primeira *release*[6] do ArgoUML foi disponibilizada em 1998, sendo que atualmente o projeto encontra-se na *release* 0.30.2.

O ArgoUML foi concebido como uma ferramenta para uso na Análise e Projeto de sistemas de software Orientado a Objeto. Atualmente, encontra-se traduzido em dez idiomas e é distribuído e usado em todo mundo. As características citadas a seguir fizeram com que o ArgoUML ganhasse visibilidade no meio acadêmico e comercial (O'RIORDAN, 2009):

[6] *Release* pode ser traduzido como "liberação de software", refere-se ao lançamento de uma nova versão oficial de um produto de software. Neste trabalho, optou-se em manter o termo em inglês por ser mais difundido e facilmente identificado.

1) O ArgoUML é baseado em pesquisas em psicologia cognitiva para fornecer características que aumentam a produtividade por meio do apoio às necessidades cognitivas dos desenvolvedores de software Orientado a Objeto e arquitetos.
2) O ArgoUML é compatível com padrões abertos amplamente difundidos como a UML, XMI, SVG, OCL, entre outros. A este respeito, o ArgoUML ainda está à frente de muitas ferramentas comerciais.
3) O ArgoUML é uma aplicação 100% Java. Essa característica permite que o ArgoUML execute em todas as plataformas para as quais a plataforma Java 2 está disponível.
4) O ArgoUML é um projeto *open source*[7]. A disponibilidade do código fonte possibilita que uma nova geração de projetistas de software e pesquisadores dêem continuidade ao desenvolvimento/evolução do projeto.

O nome ArgoUML, proposto pelo Jason Elliot Robbins, foi retirado da história "*Jason and the Argonauts*" pertencente à mitologia grega, pois o nome Jason da história grega coincide com o nome do idealizador da ferramenta e o tema da história tem haver com experimentação. Argo é o nome do barco utilizado pelos aventureiros para exploração, daí a origem do nome ArgoUML (UML foi incorporado ao nome, pois a ferramenta dá suporte aos diagramas e notações da UML).

A ferramenta CASE ArgoUML disponibiliza diversos diagramas da UML, como exemplo, os diagramas de Casos de Uso, Classes, Seqüência, entre outros, além de dispor de funcionalidades, como a geração/leitura de XMI e exportação de imagens, que tornam o ArgoUML uma ferramenta versátil para utilização em projetos de software. Na seção seguinte, discute-se como os módulos desenvolvidos integram-se com o ArgoUML.

6.2 Resultado da Implementação no ArgoUML

Nesta seção, apresenta-se uma visão geral das funcionalidades dos módulos desenvolvidos para ArgoUML utilizadas para dar suporte ao projeto de esquemas lógicos gráficos de BDORs. As funcionalidades aqui apresentadas integram-se ao ArgoUML por meio dos módulos, sendo transparente para o usuário final a troca de informações existente entre eles.

[7] *Open source*, ou em português código aberto, segundo (Free Software Foundation, 2010): "é qualquer programa de computador que pode ser usado, copiado, estudado e redistribuído sem restrições".

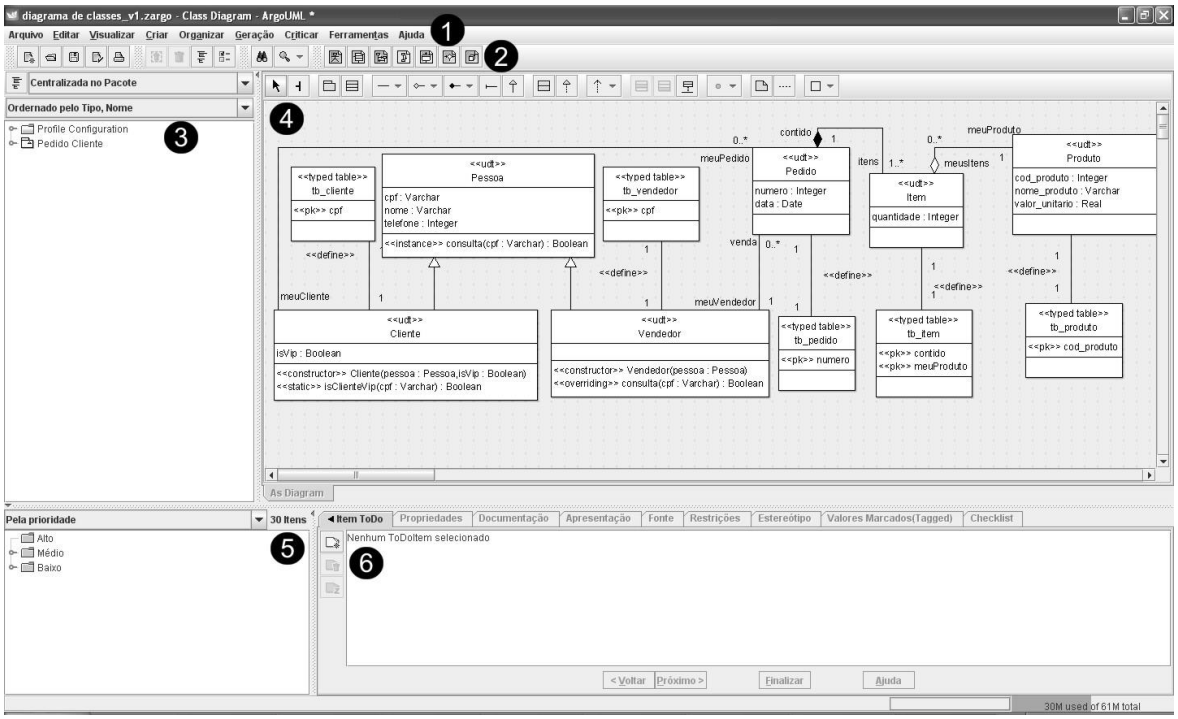

Figura 6: Janela principal do ArgoUML

A janela principal do ArgoUML, ilustrada pela Figura 6, possui uma barra de menu (número 1), uma barra de ferramentas (número 2), e quatro painéis principais: *Explorer* (número 3), Edição (número 4), Tarefa (número 5) e Detalhes (número 6).

Projeta-se o esquema lógico gráfico no painel Edição (Figura 6, número 4) utilizando-se do Perfil ORDB proposto e do Diagrama de Classes da UML. O Perfil ORDB pode ser acessado por meio das abas pertencentes ao painel Detalhes (Figura 6, número 6) ou pelo próprio painel Edição clicando com o botão direito do mouse. Já os elementos pertencentes ao diagrama de classes UML estão acessíveis pela barra de ferramentas (Figura 6, número 2).

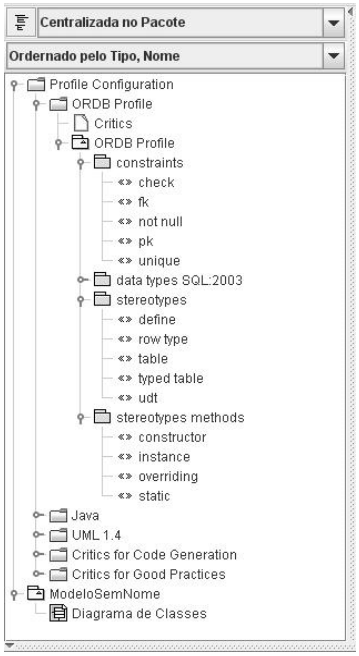

Figura 7: Visão do Perfil ORDB proposto no *Explorer* do ArgoUML

O painel *Explorer* do ArgoUML, ilustrado pela Figura 7, exibe os elementos do projeto (classes, relacionamentos, pacotes, entre outros), os diagramas pertencentes ao projeto e os perfis disponíveis. Ainda na Figura 7, destaca-se o Perfil ORDB desenvolvido, cujos elementos do modelo encontram-se organizados fisicamente em pacotes tal como discutido no capítulo 5.

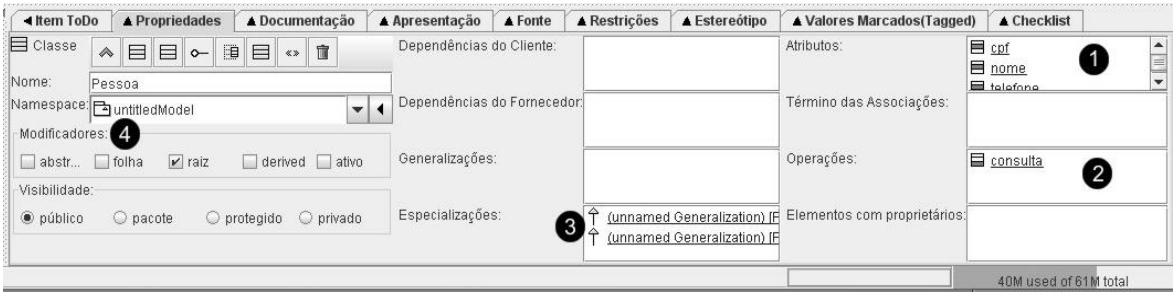

Figura 8: Aba Propriedades do painel Detalhes do ArgoUML.

A aba Propriedades do painel Detalhes do ArgoUML, ilustrada pela Figura 8, mostra as propriedades do elemento selecionado. O conteúdo da aba varia de acordo com o tipo de elemento do projeto. A aba Propriedades é definida para todos os elementos do esquema lógico e ela possibilita a criação/alteração/exclusão de atributos, operações, relacionamentos, entre outros.

No exemplo da Figura 8, uma classe denominada Pessoa foi selecionada. Observa-se que a aba exibe diversas propriedades dessa classe como: os atributos (número 1), operações (número 2), especializações (número 3), generalizações, entre outros. Por meio da aba Propriedades, o projetista pode selecionar algum modificador para a classe, dentre as opções disponíveis (número 4). O modificador selecionado poderá, ou não, influenciar no código gerado pelos módulos, por exemplo, caso a opção 'folha' fosse selecionada, a classe seria impossibilitada de ser estendida por outra classe e o código gerado pelos módulos teria a declaração FINAL. Nesse exemplo, selecionou-se a opção raiz, pois se almeja que essa classe seja superclasse de outras classes.

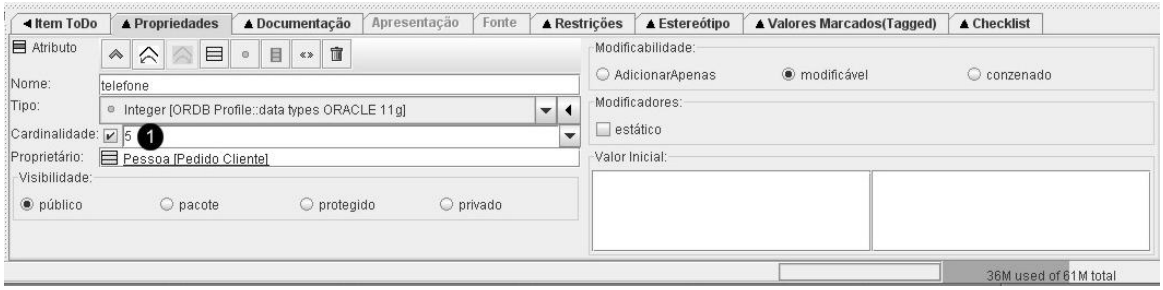

Figura 9: Aba Propriedades do painel Detalhes do ArgoUML exibindo o atributo telefone.

A Figura 9 ilustra a aba Propriedades para o atributo telefone da classe Pessoa. Observe que os campos e opções exibidos são diferentes dos apresentados pela Figura 8. O campo Cardinalidade (número 1) refere-se à cardinalidade do atributo. O código gerado para um determinado atributo irá variar de acordo com sua cardinalidade e da linguagem selecionada (SQL:2003 ou o dialeto SQL utilizado pelo Oracle 11g), no exemplo ilustrado na Figura 9, o módulo SQL:2003 irá gerar para o atributo telefone um *Array* de tamanho 5 (Figura 10). As técnicas de mapeamento utilizadas pelos módulos estão de acordo com o que foi discutido no capítulo 4.

Figura 10: Aba Fonte do Painel Detalhes.

A aba Fonte do painel Detalhes, ilustrado pela Figura 10, permite a visualização do código-fonte gerado para um determinado elemento selecionado no esquema lógico. A linguagem alvo pode ser selecionada a partir do menu indicado com o número 1 na Figura 10. O menu indicado com o número 2, permite a seleção do *script* para exportação do esquema.

Figura 11: Aba Estereótipo do Painel Detalhes.

Na aba Estereótipos do painel Propriedades da Figura 11, encontram-se os estereótipos definidos no perfil. Os estereótipos serão exibidos nessa aba de acordo com o elemento selecionado, por exemplo, se o projetista selecionar uma classe do esquema lógico gráfico e clicar na aba Estereótipos, apenas os estereótipos aplicáveis a classes («udt», «typed table», «table» etc.) serão exibidos. Nessa aba o projetista pode atribuir um estereótipo para um determinado elemento selecionado do esquema.

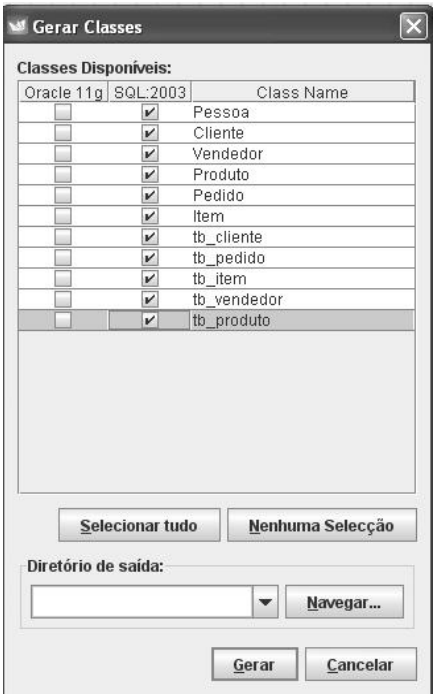

Figura 12: Tela de seleção das classes para geração do *script*.

Por fim, na barra de menu indicada com o número 1 na Figura 6, aparecerá a opção “Gerar Todas as Classes” clicando-se no botão “Geração”, essa opção exibe a tela ilustrada na Figura 12. Por meio dessa tela, o projetista pode selecionar as classes do esquema para gerar o *script* contendo o código SQL na linguagem escolhida.

6.3 Arquitetura do Módulo para Geração de Código em SQL:2003

Nesta seção, discute-se a arquitetura de software utilizada no desenvolvimento do módulo referente à geração automática do código em SQL:2003. O módulo desenvolvido para gerar código no dialeto SQL utilizado pelo Oracle 11g segue a mesma arquitetura aqui discutida.

Optou-se em utilizar padrões de projeto baseando-se em (LARMAN, 2008; FOWLER, 2008), visando tornar o código flexível a mudanças/evoluções da especificação da SQL e possíveis extensões. Os trabalhos de (LARMAN, 2008; FOWLER, 2008) discutem a respeito de padrões arquiteturais para o desenvolvimento de software, essas obras apresentam as situações em que cada padrão é indicado na prática, sendo boas referências para Análise e Projeto Orientado a Objetos (APOO).

Um módulo no ArgoUML é uma coleção de classes e arquivos de recursos (como o perfil SQL:2003 que é um arquivo em XMI) que podem ser habilitados e desabilitados na ferramenta. Os módulos podem ser desenvolvidos de forma independente e são carregados quando o ArgoUML é iniciado. Eles fornecem um meio de estender os recursos da ferramenta sem que para isso seja necessário modificar o código da mesma. Essa flexibilidade de extensão foi planejada pelos projetistas do ArgoUML para incentivar o desenvolvimento de outros ambientes de código aberto e/ou comercial.

Quando os módulos são carregados eles têm a capacidade de acessar os elementos internos do ArgoUML. Uma vez que o módulo instalado é reconhecido pelo sistema de carregamento de módulos do ArgoUML, ele receberá chamadas que serão respondidas por meio de determinadas ações (como gerar código, exibir perfil ORDB etc.).

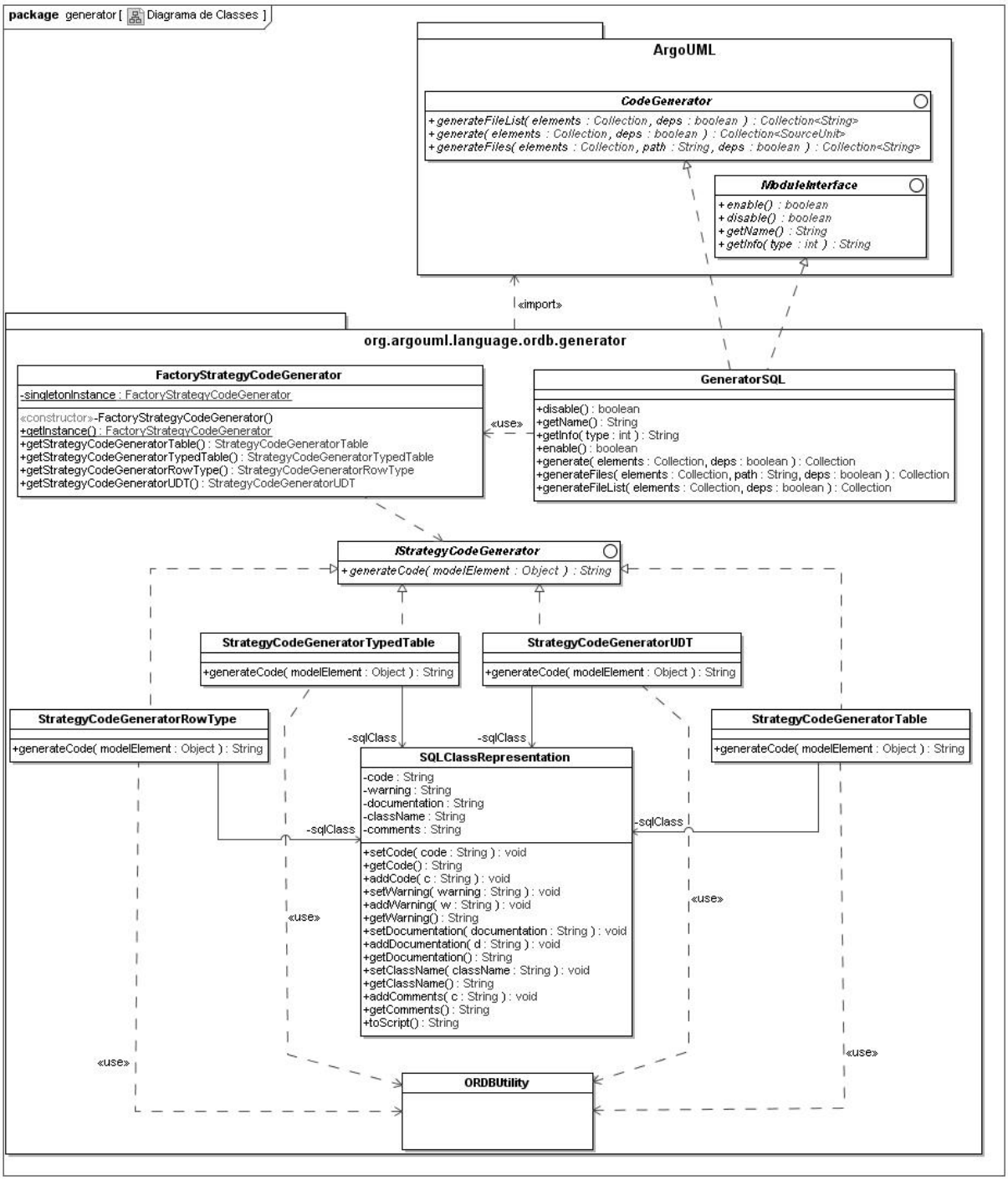

Figura 13: Arquitetura utilizada no módulo desenvolvido para geração do código em SQL:2003

A Figura 13 ilustra a arquitetura utilizada no desenvolvimento do módulo para geração de código em SQL:2003. Para facilitar a visualização do diagrama e com isso melhorar a compreensão da arquitetura, focou-se apenas nas propriedades mais relevantes, omitindo-se as propriedades privadas, protegidas e os métodos estáticos da classe utilitária *ORDBUtility*. As propriedades abstratas de uma classe aparecem em itálico e as estáticas apresentam-se

sublinhadas, o sinal de mais indica que a propriedade possui acesso público e o nome de uma classe em itálico representa uma interface.

A classe de controle é a *org.argouml.moduleloader.ModuleLoader2*. Essa classe é um *singleton*[8] criado quando utilizado pela primeira vez. O *singleton* é utilizado na rotina de inicialização principal; quando criado, ele realiza uma busca através de todos os módulos instalados e cria uma lista dos objetos principais (implementações da interface *ModuleInterface*, ver Figura 13). Os módulos encontrados são, por padrão, selecionados e no final da rotina de inicialização eles são habilitados.

Cada classe carregada pelo *ModuleLoader2* deve implementar a interface *ModuleInterface*. Essa interface tem os seguintes métodos: *enabling* (habilitar), *disabling* (desabilitar), e *getName* (identificar) e *getInfo* (informar) (Figura 13). As classes contidas no módulo podem atuar em qualquer recurso do ArgoUML, sendo que o ArgoUML não necessita ter qualquer conexão com o módulo de carga, ou seja, existe um acoplamento unidirecional entre o módulo desenvolvido e o ArgoUML (observe na Figura 13 que a dependência possui o estereótipo «import»). Os módulos são tratados como se fossem parte do *core* do ArgoUML, contudo, fisicamente estão separados.

Uma importante característica dos módulos é que uma vez que eles são carregados, não é possível distingui-los do resto do ambiente do ArgoUML, ou seja, para o usuário final os módulos e o ArgoUML correspondem a um único sistema.

Outra interface pertencente ao ArgoUML utilizada para o desenvolvimento do módulo SQL:2003 é a interface *CodeGenerator*. Essa interface define os métodos básicos de um módulo para geração de código em determinada linguagem (por exemplo, SQL:2003 e Oracle 11g). É por meio dos métodos *generateFileList*, *generate* e *generateFiles* (Figura 13) que o ArgoUML comunica-se com os módulos, passando as informações necessárias para a geração do código.

A classe *GeneratorSQL* pertence ao módulo desenvolvido (*argouml-module-ordb*) e implementa duas interfaces, *CodeGenerator* e *ModuleInterface*, discutidas anteriormente. Essa classe é o ponto de contato entre os pacotes pertencentes ao módulo ORDB com ArgoUML (Figura 13).

Como o comportamento do módulo para a geração de código varia de acordo com o elemento da SQL, optou-se em utilizar o padrão de projeto Estratégia. Para isso, criou-se

[8] *Singleton* é um padrão de projeto de software que garante que exatamente uma única instância da classe será criada, trata-se de um objeto unitário. A classe que implementa esse padrão disponibiliza um ponto de acesso global e único para obtenção da instância da classe.

várias classes *StrategyCodeGenerator*, cada uma com um método polimórfico *generateCode*. Cada método *generateCode* recebe um objeto de contexto denominado *modelElement* como parâmetro, de modo que o objeto estratégia de geração de código possa gerar o código apropriado de acordo com esse objeto. A implementação de cada método *generateCode* será diferente dependendo do escopo da classe a qual pertence.

As classes estratégia de geração de código são: *StrategyCodeGeneratorRowType*, *StrategyCodeGeneratorTypedTable*, *StrategyCodeGeneratorUDT* e *StrategyCodeGeneratorTable*; elas implementam a interface *IStrategyCodeGenerator*. Cada uma dessas classes é responsável pela geração do código, por meio do método *generateCode*, referente a um determinado elemento da SQL:2003, por exemplo, a classe *StrategyCodeGeneratorTypedTable* é responsável pela geração de código para tabelas tipadas, a classe *StrategyCodeGeneratorUDT* para UDTs etc.

Um detalhe importante da arquitetura utilizada para o desenvolvimento dos módulos é a sua flexibilidade em evoluir/estender de acordo com as mudanças provenientes das versões da SQL posteriores a utilizada neste trabalho (SQL:2003). Por exemplo, se em uma próxima versão da SQL possuir um elemento não existente na versão utilizada neste trabalho, bastará criar uma classe estratégia de geração de código com o nome do novo elemento e estendê-la da interface *IStrategyCodeGenerator* e fazer como que o método polimórfico *generateCode* gere o código apropriado para o novo elemento. Dessa forma, é possível evoluir o código dos módulos para que correspondam a versões posteriores da SQL.

A classe *SQLClassRepresentation* representa os elementos definidos pela SQL. Observe que essa classe possui as propriedades relacionadas ao código gerado pela ferramenta, são elas: código fonte (*code*), mensagens de alerta/sugestões (*warning*), documentação do código (*documentation*), nome definido pelo projetista (*className*) e os comentários feitos pelo projetista (*comments*). Cada classe estratégia de geração de código possui um atributo privado *SQLClassRepresentation* (ver Figura 13), sendo que o código armazenado por esse atributo irá variar de acordo com o escopo de cada estratégia.

A classe *ORDBUtility* é comum às classes estratégia de geração de código. O objetivo dessa classe é fornecer um conjunto de métodos utilitários de uso comum. Por possuir todos os métodos estáticos, ou seja, métodos de classe, as estratégias não necessitam criar uma instância da classe utilitária para acessar seus métodos.

A classe *singleton FactoryStrategyCodeGenerator* é responsável pela criação das estratégias de geração de código, optou-se por utilizar o padrão Fábrica pelas seguintes razões (LARMAN, 2008):

- Separam a responsabilidade da criação complexa em objetos auxiliares coesos.
- Ocultam a lógica de criação.

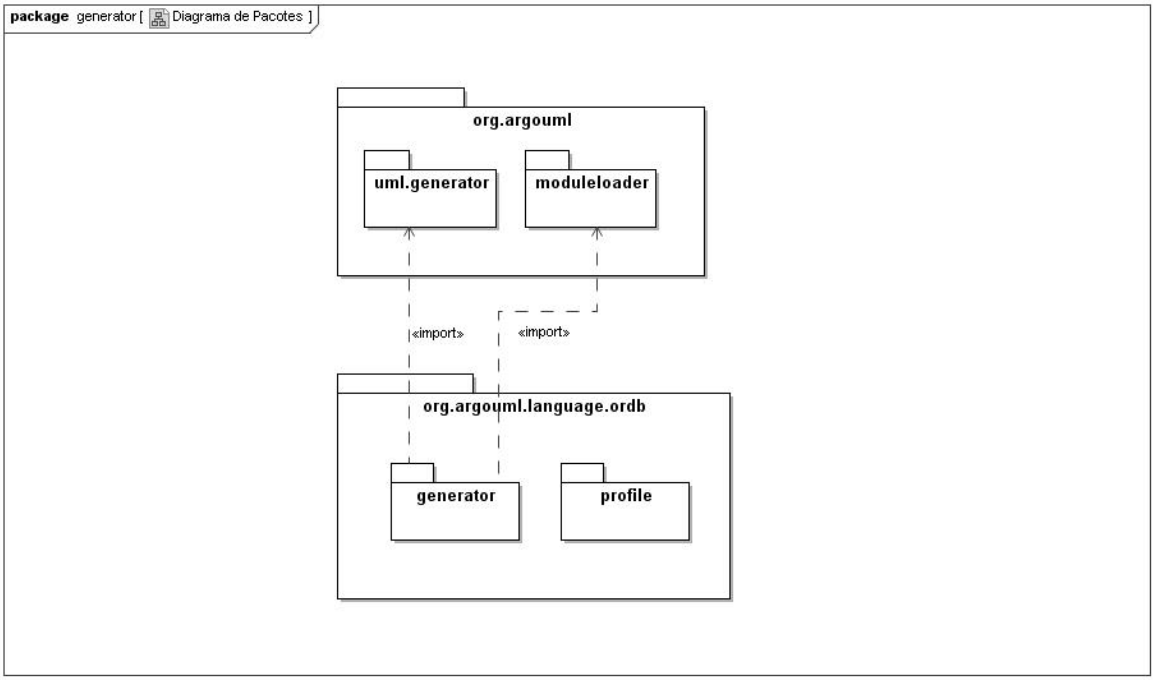

Figura 14: Diagrama de pacotes do módulo para geração de código em SQL:2003

Na Figura 14, apresenta-se o diagrama de pacotes simplificado do módulo gerador de código em SQL:2003. Esse diagrama descreve a estrutura do módulo desenvolvido em um nível mais elevado, destacando as dependências entre os pacotes. O pacote *argo.argouml* pertence ao núcleo do ArgoUML e o pacote *argo.argouml.language.ordb* pertence ao módulo desenvolvido. O pacote *profile* do módulo desenvolvido contém o perfil ORDB, discutido no capítulo anterior**,** armazenado em um arquivo no formato XMI.

6.4 Conclusões

Iniciou-se o capítulo apresentando de forma sucinta a origem do ArgoUML. Essa ferramenta CASE teve origem no meio acadêmico durante o Doutorado de Jason Elliot Robbins. Conclui-se que o ArgoUML é uma ferramenta CASE versátil, pois dá suporte a diversos padrões como XMI, UML, entre outros, além de possuir funcionalidades para exportação de imagens e de ter sido desenvolvido atendendo a aspectos ligados as necessidades cognitivas dos desenvolvedores de software.

Na segunda seção, apresentaram-se os resultados da implementação dos módulos desenvolvidos para o ArgoUML. Discutiram-se as principais funcionalidades desses módulos e a forma como essas funcionalidades encontram-se disponíveis na ferramenta CASE para a modelagem gráfica de um BDOR. Destaca-se a transparência para o usuário final da interdependência e a troca de mensagens que ocorrem entre os módulos desenvolvidos e a ferramenta ArgoUML durante toda a modelagem e para a geração de código SQL.

Por fim, apresentou-se a arquitetura de software utilizada para o desenvolvimento dos módulos responsáveis pela geração automática de código em SQL:2003 e no dialeto SQL utilizado pelo Oracle 11g. A arquitetura apresentada foi projetada para suprir as necessidades atuais dos módulos e ser flexível para atender às mudanças das versões da SQL posteriores a utilizada neste trabalho (SQL:2003).

7 Exemplo da Utilização da Ferramenta

Neste capítulo, apresenta-se um exemplo de utilização dos módulos desenvolvidos para o Projeto Lógico de BDORs na ferramenta CASE ArgoUML. Para o exemplo, elaborou-se um estudo de caso fictício que consiste no desenvolvimento de um sistema simplificado para uma loja online especializada em produtos de informática. Para esse projeto, utilizaram-se a arquitetura apresentada no capítulo 3, o Modelo Lógico Gráfico proposto e as técnicas de mapeamento discutidas no capítulo 5.

O objetivo desse estudo de caso é prover um exemplo prático de um projeto em BDORs, que se inicia pela fase de Concepção do sistema e a modelagem do Projeto Conceitual para os requisitos levantados (seção 1). Em seguida, elabora-se o Projeto Lógico (seção 2) com base no Projeto Conceitual desenvolvido inicialmente. O Projeto Lógico é mapeado para o Projeto Físico na seção 3. Por fim, as conclusões obtidas são apresentadas na seção 4 deste capítulo.

7.1 Projeto Conceitual Desenvolvido para o Exemplo

Nesta seção, primeiro apresentam-se os requisitos identificados na fase de concepção do sistema, em seguida, é exposto o Projeto Conceitual.

Ressalta-se que, por tratar-se de um exemplo ilustrativo, optou-se por contemplar apenas uma parte simplificada dos requisitos que um sistema para uma loja de informática *online* poderia exigir. A seguir, apresentam-se os requisitos para o exemplo:

1) A loja trabalha com a venda de produtos nos seguimentos de software e hardware;
2) Clientes cadastrados, identificados por nome e CPF, podem realizar compras dos produtos disponíveis no site da loja;
3) O sistema deve armazenar o endereço (número e a rua) dos clientes cadastrados;
4) Os produtos no seguimento de software apresentam o número de versão. Já os produtos no seguimento de hardware apresentam a montadora;
5) O sistema deve possibilitar a geração de relatórios tanto para clientes, relacionando as compras realizadas em um determinado período, como para gerência.

O Projeto Conceitual corresponde à primeira fase da arquitetura apresentada na seção 3.1 do capítulo 3. De acordo com a arquitetura da Figura 3 (seção 3.1), inicia-se o projeto de

um BDOR com o desenvolvimento de um esquema conceitual utilizando-se os elementos definidos pela UML para o Diagrama de Classes.

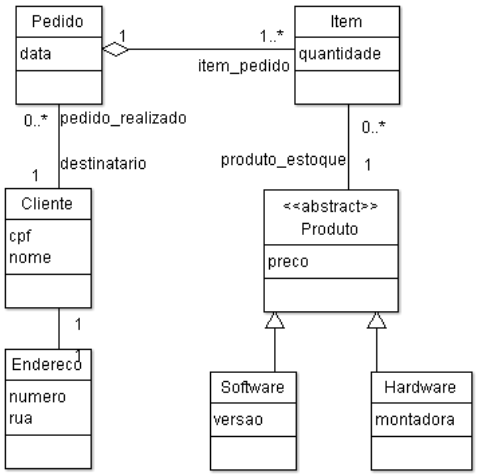

Figura 15: Exemplo de um Projeto Conceitual.

A Figura 15 ilustra o esquema conceitual desenvolvido no ArgoUML com base nos requisitos apresentados anteriormente.

Para que uma ferramenta CASE possa automatizar o mapeamento do Projeto Conceitual para o Projeto Lógico, é necessário o desenvolvimento de uma notação específica, pois, como discutido no capítulo 4, um único elemento UML pode gerar diferentes elementos no Projeto Lógico. Por exemplo, uma Classe pode corresponder a um UDT, a um Tipo Linha, a uma Tabela ou a uma Tabela Tipada. A notação específica para o Projeto Conceitual e seu respectivo mapeamento para o Projeto Lógico não fazem parte do escopo deste trabalho, sendo alvo de futuras investigações, como será discutido no capítulo de encerramento (capítulo 8). Dessa forma, a tradução do esquema apresentado na Figura 15 para o esquema lógico da Figura 16 é realizada manualmente.

7.2 Projeto Lógico Desenvolvido para o Exemplo

Dentre os objetivos deste trabalho estão o desenvolvimento de uma notação específica para o Projeto Lógico de BDORs, a qual foi apresentada e discutida no capítulo 5, e o desenvolvimento de dois módulos para o ArgoUML. Os módulos dão suporte à modelagem gráfica de esquemas lógicos, utilizando-se a notação proposta, e à geração automática de código em SQL:2003 e no dialeto SQL utilizado pelo Oracle 11g.

Nesta seção, apresenta-se a segunda fase da arquitetura apresentada na Figura 3 (seção 3.1). Essa fase corresponde ao Projeto Lógico de um BDOR e faz uso do Perfil UML desenvolvido.

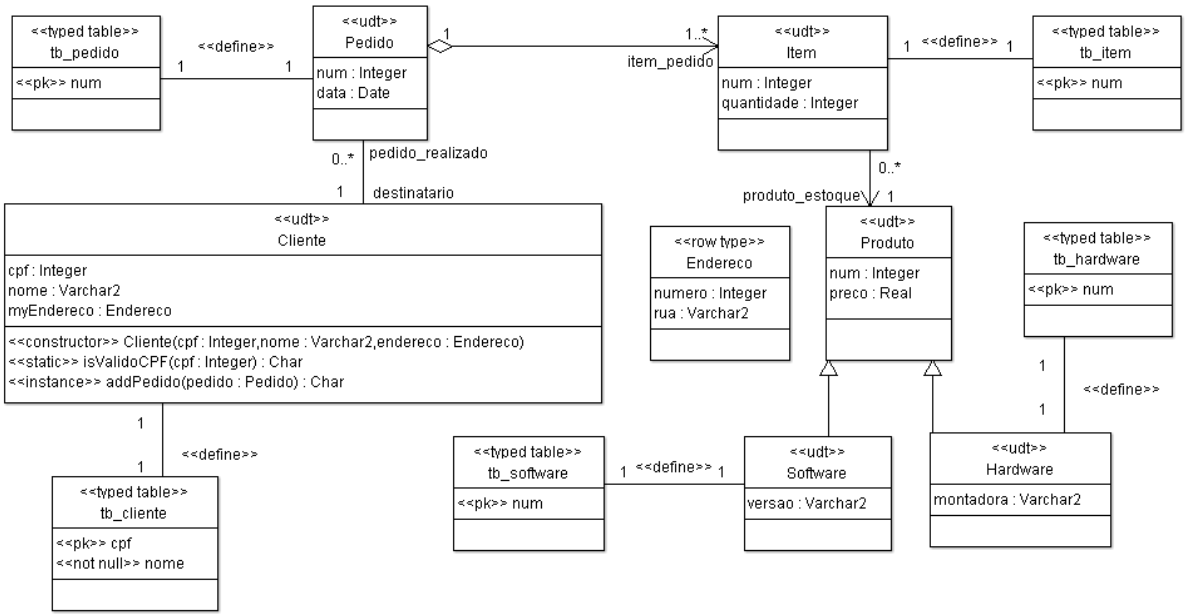

Figura 16: Exemplo de um Projeto Lógico Gráfico.

A Figura 16 apresenta o esquema lógico gráfico desenvolvido para exemplo, o qual foi modelado no ArgoUML utilizando-se os componentes desenvolvidos. Com relação ao esquema ilustrado pela Figura 16, têm-se as seguintes considerações:

1. Devido ao fato de em um UDT não ser possível declarar restrições em seus atributos, estas devem ser declaradas nos atributos apropriados da tabela tipada.
2. A classe Endereco do esquema conceitual foi mapeada para um tipo linha no esquema lógico, pois essa classe não apresenta métodos na sua definição e não haverá necessidade de acessá-la individualmente, ou seja, no estudo de caso o acesso a Endereco está estritamente relacionado ao acesso a Cliente, dessa forma, julgou-se apropriado a utilização de uma estrutura aninhada. Para isso definiu-se o atributo myEndereco em Cliente (Figura 16) cujo domínio é do tipo Endereco.
3. A classe abstrata Produto foi mapeada para um UDT sem uma tabela tipada associada, pois, como discutido no capítulo 4, classes abstratas não podem ser instanciadas, logo a criação de uma tabela para persistir objetos dessa classe corresponderia a desperdício de espaço.

4. As demais classes (Software, Hardware, Item, Pedido e Cliente) foram mapeadas para UDTs com tabelas tipadas associadas, pois essas classes são instanciáveis e seus objetos poderão ser persistidos, além disso, com essa implementação é possível utilizar os recursos disponibilizado pela SQL:2003, são eles: criação de métodos, herança, polimorfismo e criação de referências.

7.3 Projeto Físico Gerado para o Exemplo

Nesta seção, discute-se o Projeto Físico de um BDOR que corresponde à última fase da arquitetura apresentada na Figura 3 (seção 3.1). Essa fase caracteriza-se por ser dependente de plataforma, ou seja, depende de uma tecnologia específica, que neste trabalho corresponde ao Oracle 11g. Apresenta-se também o código em SQL:2003 gerado para o esquema lógico com a finalidade de compará-lo com código no dialeto SQL utilizado pelo Oracle 11g.

A Tabela 10 apresenta na primeira coluna o código em SQL:2003 e na segunda coluna o código utilizando o dialeto SQL do Oracle 11g. Esse código foi gerado automaticamente por meio dos módulos desenvolvidos para o ArgoUML a partir do esquema apresentado na Figura 16.

Tabela 10: Código SQL correspondente ao esquema ilustrado na Figura 16.

SQL:2003	Oracle 11g
CREATE TYPE Produto AS (num INTEGER, preco REAL) NOT FINAL	CREATE TYPE Produto AS OBJECT (num INTEGER, preco REAL)NOT FINAL
CREATE TYPE Software UNDER Produto AS (versao VARCHAR2) FINAL CREATE TABLE tb_software OF Software(num WITH OPTIONS PRIMARY KEY)	CREATE TYPE Software UNDER Produto (versao VARCHAR2(15))FINAL / CREATE TABLE tb_software OF Software(num PRIMARY KEY)
CREATE TYPE Hardware UNDER Produto AS (montadora VARCHAR2)FINAL CREATE TABLE tb_hardware OF Hardware(num WITH OPTIONS PRIMARY KEY)	CREATE TYPE Hardware UNDER Produto (montadora VARCHAR2(30))FINAL / CREATE TABLE tb_hardware OF Hardware(num PRIMARY KEY)
CREATE TYPE Item AS (num INTEGER, quantidade INTEGER, produto_estoque REF(Produto))FINAL CREATE TABLE tb_item OF Item(produto_estoque WITH OPTIONS NOT NULL, num WITH OPTIONS PRIMARY KEY)	CREATE TYPE Item AS OBJECT (num INTEGER, quantidade INTEGER, produto_estoque REF Produto)FINAL / CREATE TABLE tb_item OF Item(num PRIMARY KEY)
CREATE TYPE Pedido AS (num INTEGER, data DATE, destinatario REF(Cliente), item_pedido REF(Item) MULTISET)FINAL	CREATE TYPE item_nested_table_type AS TABLE OF REF Item / CREATE TYPE Pedido AS OBJECT (num INTEGER, data DATE, destinatario REF Cliente, item_pedido item_nested_table_type)FINAL
CREATE TABLE tb_pedido OF Pedido(destinatario WITH OPTIONS NOT NULL, item_pedido WITH OPTIONS NOT NULL, num WITH OPTIONS PRIMARY KEY)	CREATE TABLE tb_pedido OF Pedido(num PRIMARY KEY) NESTED TABLE item_pedido STORE AS item_pedido_nested_table
Endereco ROW(numero INTEGER, rua VARCHAR2)	CREATE TYPE Endereco AS OBJECT (numero INTEGER, rua VARCHAR2(30))FINAL
CREATE TYPE Cliente AS (cpf INTEGER, nome VARCHAR2(30), myEndereco ENDERECO, pedido_realizado REF(Pedido) MULTISET)FINAL CONSTRUCTOR METHOD Cliente(cpf INTEGER, nome VARCHAR2, endereco ENDERECO) RETURNS SELF AS RESULT, STATIC METHOD isValidoCPF(cpf INTEGER) RETURNS CHAR, INSTANCE METHOD addPedido(pedido PEDIDO) RETURNS CHAR	CREATE TYPE pedido_nested_table_type AS TABLE OF REF Pedido / CREATE TYPE Cliente AS OBJECT (cpf INTEGER, nome VARCHAR2(30), myEndereco ENDERECO, pedido_realizado pedido_nested_table_type, CONSTRUCTOR FUNCTION Cliente(cpf INTEGER, nome VARCHAR2, endereco ENDERECO) RETURN SELF AS RESULT, STATIC FUNCTION isValidoCPF(cpf INTEGER) RETURN CHAR, MEMBER FUNCTION addPedido(pedido PEDIDO) RETURN CHAR)FINAL
CREATE TABLE tb_cliente OF Cliente(nome WITH OPTIONS NOT NULL, cpf WITH OPTIONS PRIMARY KEY)	CREATE TABLE tb_cliente OF Cliente(nome NOT NULL, cpf PRIMARY KEY) NESTED TABLE pedido_realizado STORE AS pedido_realizado_nested_table

Com relação ao código apresentado pela tabela 10, são feitas as seguintes considerações:

1. Comparando-se o código SQL utilizado pelo Oracle 11g com o padrão SQL:2003, observa-se que o dialeto do Oracle é similar ao definido pelo padrão, embora apresente uma pequena variação notacional.
2. O relacionamento de composição entre Pedido (classe UDT todo) e Item (classe UDT parte) é implementado por meio de um *Multiset* de referências na classe todo. O *Multiset* é utilizado, pois a multiplicidade do relacionamento é * (limite superior da multiplicidade não é definido).
3. O Oracle 11g não garante a integridade referencial em *Multisets* e *Arrays* de referências, caso ocorra uma tentativa de acesso a uma referência que não exista, o valor nulo será retornado. Sendo assim, cabe ao projetista a definição de métodos ou *triggers* (gatilhos) para garantir a integridade referencial nessas estruturas.
4. Como o Oracle 11g não suporta o tipo linha, optou-se por utilizar um UDT sem a definição de métodos (comportamento), além disso, devido ao fato do tipo linha não poder ser estendido, definiu-se o UDT como final, ou seja, não é possível estendê-lo.
5. O Oracle 11g define o *Multiset* de forma diferente da definida pela SQL:2003 e conseqüentemente, diferente do modelo lógico gráfico proposto. Observa-se, por exemplo, a classe Pedido; no código em SQL:2003 (Tabela 10 e na Figura 16), Pedido possui um *Multiset* de referências denominado item_pedido, já o código correspondente no Oracle 11g a definição de um *Multiset* é feita por meio da criação de um tipo tabela de referências denominado item_nested_table_type. Outra diferença entre o padrão SQL:2003 e o Oracle 11g está na definição de uma tabela tipada a partir de um UDT que possui um *Multiset*. Ao definir-se uma tabela tipada, cada atributo do tipo tabela de referências (*Multiset*) deverá ser explicitamente declarado, observe que na definição da tabela tipada tb_pedido, declarou-se uma tabela aninhada (*Nested Table*) denominada item_pedido que será armazenada fisicamente com o nome item_pedido_nested_table.

7.4 Conclusões

Neste capítulo, apresentou-se um exemplo prático de um projeto em BDORs. Utilizaram-se a arquitetura apresentada no capítulo 3 (seção 3.1) como referência e o ArgoUML como ferramenta CASE para suporte à modelagem e para geração automática de código SQL para o Oracle 11g.

Na fase do Projeto Lógico, empregou-se o perfil UML desenvolvido. Esse perfil corresponde à implementação do Modelo Lógico proposto no capítulo 5. O esquema lógico gráfico foi modelado utilizando-se os componentes desenvolvidos neste trabalho para o ArgoUML. Conclui-se que é mais fácil e rápido compreender o projeto lógico gráfico (Figura 16) que utiliza a notação proposta ao compará-lo com seu correspondente código em SQL:2003 (Tabela 10).

A Fase de Projeto Físico da arquitetura apresentada (seção 3.1) é caracterizada por ser dependente de plataforma, ou seja, dependente de uma tecnologia específica. Para o exemplo deste capítulo, utilizou-se o SGBDOR Oracle 11g nessa fase do projeto.

Destaca-se que o código SQL do Oracle 11g é muito próximo do especificado pelo padrão SQL:2003, pois muitos dos recursos definidos pelo padrão são suportados por esse SGBDOR. Contudo, nem todos os elementos são suportados, como é o caso do tipo linha. Como alternativa, o módulo responsável pela geração de código para o Oracle 11g apresenta ao projetista, por meio de mensagens, sugestões e alternativas para auxiliá-lo na modelagem. No caso do tipo linha, por exemplo, um UDT não extensível (marcado como final) que não possua métodos será sugerido como opção de uso.

Por fim, mostrou-se o mapeamento automático realizado pelos componentes desenvolvidos para o ArgoUML. Esse mapeamento corresponde à tradução do esquema lógico gráfico para o esquema físico. Conclui-se que os esforços foram reduzidos com essa automatização, contudo, ainda é necessária a definição de uma notação específica para o Projeto Conceitual com a finalidade de que a tradução do esquema conceitual para o esquema lógico também seja feita automaticamente.

8 Conclusão e Comentários

A proposta deste trabalho teve como escopo o Projeto em Banco de Dados Objeto-Relacionais para o qual se criou uma arquitetura baseada na ANSI/SPARC e na MDA, cujo objetivo foi de contextualizar as tecnologias envolvidas e utilizadas nas diferentes fases de um projeto; essa arquitetura também serviu para destacar os pontos em que houve contribuição deste trabalho, norteando o exemplo de utilização da ferramenta CASE (capítulo 7) e as discussões durante todo o texto. Dentre as fases de um projeto (Figura 3 da seção 3.1) focou-se principalmente na fase de Projeto Lógico e em seu mapeamento para o Projeto Físico de um BDOR.

As seções seguintes apresentam as conclusões obtidas para cada um dos objetivos deste trabalho.

8.1 Sobre o Modelo Lógico Gráfico Proposto

A falta de um Modelo aceito e difundido para o Projeto em BDORs impulsionou a proposta de um Modelo Lógico Gráfico para BDORs, o qual foi desenvolvido analisando-se os trabalhos correlatos com intuito de elaborar um Modelo conciso que atendesse as necessidades dos projetistas/desenvolvedores e que estivesse de acordo com o padrão SQL:2003.

Os trabalhos de (MARCOS et al., 2003; GOLOBISKY et al., 2005), os quais serviram como base da proposta, não abordaram algumas características do padrão SQL:2003, como por exemplo, a possibilidade de definirem-se métodos estáticos, métodos de instância e métodos construtores. Essas propostas também não apresentam a mesma flexibilidade apresentada pela SQL:2003, como por exemplo, a possibilidade de definirem-se tabelas e tabelas tipadas em um mesmo projeto, pois, nessas propostas todas as classes do modelo devem ser mapeadas para tabelas tipadas.

Os aspectos que foram identificados como não apropriados para o projeto de um BDOR nos trabalhos de (MARCOS et al., 2003; GOLOBISKY et al., 2005) foram discutidos ao longo do capítulo 4 e uma síntese das principais diferenças existentes entre esses trabalhos e o modelo aqui proposto foi apresentada na seção 4.3.

O modelo proposto consolida-se por meio do desenvolvimento de um Perfil UML, o qual incorpora os elementos existentes na SQL:2003 fazendo uso de estereótipos. O esquema

lógico do projeto de um BDOR desenvolvido utilizando o perfil proposto permite uma fácil visualização e compreensão. O perfil desenvolvido será disponibilizado no formato XMI, com o intuito de torná-lo utilizável em ferramentas CASE de diferentes fabricantes, além disso, o perfil faz parte integrante dos módulos desenvolvidos para o ArgoUML, isso implica que os projetistas que utilizarem os módulos terão acesso ao modelo sem que haja necessidade de importá-lo para a ferramenta CASE.

Apresentou-se um exemplo no capítulo 7 que fez uso do Modelo Lógico Gráfico proposto. Ao defrontar o esquema lógico gráfico desenvolvido para o exemplo (Figura 16) com seu respectivo código em SQL:2003 (Tabela 10), fica visível que o esquema gráfico é mais compreensível que o código, embora ambos signifiquem a mesma coisa, dado que cada elemento do esquema lógico gráfico possui um correspondente em SQL:2003 (o mapeamento do modelo lógico gráfico para o código em SQL:2003 foi discutido no capítulo 5).

8.2 Sobre os Módulos Desenvolvidos para o ArgoUML

Desenvolveu-se dois módulos para a ferramenta CASE ArgoUML: um módulo que a partir do esquema lógico gráfico gera código no dialeto SQL utilizado pelo Oracle 11g (mapeamento do Projeto Lógico para o Projeto Físico) e outro que, também, a partir do esquema lógico gráfico gera código em SQL:2003 (mapeamento do esquema lógico gráfico para o esquema lógico em SQL:2003).

O desenvolvimento dos módulos possibilitou:

1. Dar início ao desenvolvimento de ferramentas CASE para suporte à modelagem objeto-relacional, dada a ausência desse tipo de ferramenta.
2. Estimular o desenvolvimento de extensões de ferramentas CASE para o projeto Lógico em BDORs por meio da disponibilização dos módulos, código fonte e da arquitetura do software.
3. Fornecer um ambiente para o desenvolvimento de Projeto Lógico em BDOR que utiliza o Modelo proposto, automatizando a geração de código e auxiliando o projetista por meio de mensagens exibidas pela ferramenta durante o desenvolvimento de esquemas.
4. Disponibilizar o Modelo Lógico Gráfico proposto para ser utilizado e testado. Por meio dos módulos é possível aferir não só a validade do Modelo para modelagem de projetos lógicos, como também as técnicas de mapeamentos discutidas no capítulo 5.

8.3 Trabalhos futuros

Para o caso do modelo lógico ser gerado a partir do diagrama de classes (mapeamento do Projeto Conceitual para o Projeto Lógico da arquitetura apresentada pela Figura 3 da seção 3.1), necessita-se estabelecer uma notação específica para marcar os seus elementos. Isto porque um mesmo elemento do Diagrama de Classes pode ter diferentes correspondentes no Modelo Lógico, por exemplo, uma Classe pode ser mapeada para um UDT, um Tipo Linha, uma Tabela ou ainda uma Tabela Tipada. Essa notação permitirá que a ferramenta CASE possa fazer o mapeamento adequado e de forma automática a partir do projeto conceitual, ou seja, um único modelo canônico poderá ser utilizado tanto pela aplicação quanto para a geração automática do projeto físico da base de dados.

Destaca-se também a necessidade de investigar o mapeamento do modelo lógico proposto para outros SGBDs, tal como o MS SQL Server, PostgreSQL, entre outros. Para tanto, faz-se necessário o estudo do suporte desses SGBDs ao padrão SQL, com o intuito de definirem-se técnicas de mapeamento.

O estudo de técnicas de mapeamento do modelo lógico gráfico proposto para outros SGBDOR possibilitará a extensão dos módulos desenvolvidos para que os mesmo gerem código SQL automaticamente a partir de esquemas.

Por fim, necessita-se investigar os aspectos relacionados ao desempenho dos recursos incorporados à SQL:2003 e também do impacto decorrente desses recursos no desenvolvimento de aplicações e de projeto de BDs.

Referências Bibliográficas[9]

CASTRO, T. R.; SOUZA, S. N. A.; SOUZA, L. S. **Proposta de um Modelo para Projetos Lógicos Gráficos para Bancos de Dados Objeto-Relacionais (BDORs)**. XIX SEMINCO – Seminário de Computação, Blumenau, SC, 2010.

DATE, C. J. **Introdução a Sistemas de Banco de Dados**. Rio de Janeiro: Ed. Campus, 2004.

EISENBERG, A.; KULKARNI, K.; MELTON, J.; MICHELS, J.; ZEMKE, F. **SQL:2003 Has Been Published**. ACM SIGMOD, New York, NY, v. 33, n. 1, p. 119-126, 2004.

EHLMANN, B. K. **Incorporating Object Relationship Notation (ORN) into SQL – Revisited**. ACM-SE 44 Proceedings of the 44th annual Southeast regional conference, New York, NY, p. 389-394, 2006.

FEUERLICHT , G.; POKORNÝ, J.; RICHTA, K. **Object-Relational Database Design: Can Your Application Benefit from SQL:2003?** Springer Science: Information Systems Development: Challenges in Practice, Theory, and Education, Galway, Ireland, v. 2, p. 975-988, 2009.

FOWLER, M. **Padrões de Arquitetura de Aplicações Corporativas**. São Paulo, SP: Artmed, 2008.

FRANKEL, D. S. **Model Driven Architecture: Applying MDA to Enterprise Computing**. Indiana: Wiley Publishing, 2003.

GOLOBISKY, M. F.; VECCHIETTI, A. **Mapping UML Class Diagrams into Object-Relational Schemas**. Proceedings of Argentine Symposium on Software Engineering, Rosario, Argentina, p. 65-79, 2005.

JUNIOR, C. A. R.; CASTRO, T. R.; SOUZA, S. N. A. **Banco de Dados Objeto-Relacional: comparação do suporte oferecido por SGBDs para a persistência de objetos**. EPOCA'09 - Escola Potiguar de Computação e suas Aplicações, Natal, RN, p. 40-45, 2009.

KNOWLEDGE BASED SYSTEMS INC. **IDEF1X Data Modeling Method**. Disponível em < http://www.idef.com/idef1X.htm > . Acessado em: 12 de janeiro de 2011.

LARMAN, C. **Utilizando UML e Padrões: Uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo**. São Paulo, SP: Editor Artmed, 2008.

[9] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023

LEE, G.; ZEMKE, F. **SQL 2003 Standard Support in Oracle Database 10g**. Oracle Corporation, Redwood Shores, CA, Nov. 2003. Disponível em: < http://www.oracle.com/technetwork/database/focus-areas/application-development/sql-2003-twp-129141.pdf>. Acesso em: 19 abril 2011.

MARCOS, E.; VELA, B.; CAVERO, J. M.; CÁCERES, P. **Aggregation and Composition in Object-Relational Database Design**. Advances in Databases and Information Systems - ADBIS' 2001, Vilnius, Lithuani, v. 1, p. 195-209, 2001.

MARCOS, E.; VELA, B.; CAVERO J. M. **Extending UML for Object-Relational Database Design**. Heidelberg/Springer Berlin: 4th International Conference, Toronto, Canada, p. 225-239, 2001.

_____. **A Methodological Approach for Object-Relational Database Design using UML**. Heidelberg/Springer Berlin: Software and Systems Modeling, v. 2, n. 1, p. 59-72, 2003.

MELTON, J. **Database languages SQL: Part 1 Framework (SQL/Framework)**. ISO-ANSI WD 9075, ISO, Working Group WG3, 2003.

_____. **Database languages SQL: Part 2 Foundation**. ISO-ANSI WD 9075, ISO, Working Group WG3, 2003.

MOK, W. Y. **Designing Nesting Structures of User-Defined Types in Object-Relational Databases**. Information and Software Technology, Newton, MA, v. 49, p. 1017-1029, 2007.

NAVATHE, B. Shamkant; ELMASRI, Ramez. **Sistemas de Banco de Dados**. São Paulo: Pearson Education, 2009.

OBJECT MANAGEMENT GROUP. **MDA Guide Version 1.0.1 - Document Number: omg/2003-06-01**. 2003. Disponível em: <http://www.omg.org/cgi-bin/doc?omg/03-06-01>. Acesso em: 10 de novembro de 2009.

_____. **Unified Modeling Language - version 2.2**. 2009. Disponível em: <http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/>. Acesso em: 05 de fevereiro de 2010.

_____. **Meta Object Facility (MOF) Core Specification - Version 2.0**. 2006. Disponível em: < http://www.omg.org/spec/MOF/2.0/PDF/>. Acesso em: 10 de novembro de 2009.

_____. **MOF 2.0/XMI Mapping - Version 2.1.1**. 2007. Disponível em: <http://www.omg.org/cgi-bin/doc?formal/07-12-02>. Acesso em: 10 de novembro de 2009.

ORACLE. Disponível em: <http://www.oracle.com>. Acesso em: 20 de setembro de 2010.

O'RIORDAN, A. **ArgoUML Tutorial – Part 1 of 2.** Department of Computer Science, University College Cork, Ireland, 2009. Disponível: <http://www.cs.ucc.ie/~adrian/cs560/ArgoUML.pdf>. Acesso em: 20 de janeiro de 2010.

PARDEDE, E.; RAHAYU, J. W.; TANIAR, D. **New SQL Standard For Object-Relational Database Applications**. Institute of Electrical and Electronics Engineers, Inc: Standardization and Innovation in Information Technology, 2003. The 3rd Conference on, p. 191- 203, 2003.

_____. **Impact of New SQL Standard to Database Modeling**. Encyclopedia of Information Science and Technology, 2005.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S.; **Sistemas de Banco de Dados**; Campus, 1a edição; 2006.

SOMMERVILLE, I. **Engenharia de Software**. São Paulo: Pearson Education, 2009.

THE OBJECT MANAGEMENT GROUP (OMG). Disponível em < http://www.omg.org/>. Acesso em: 17 de novembro de 2009.

VARA, J. M.; VELA, B.; CAVERO, J. M.; MARCOS, E. **Model Transformation for Object-Relational Database Development**. ACM: SAC '07 Proceedings of the 2007 ACM symposium on Applied computing, New York, NY, p. 1012-1019, 2007.

APÊNDICE A – Artigo Publicado na EPOCA 2009

Banco de Dados Objeto-Relacional: comparação do suporte oferecido por SGBDs para a persistência de objetos

Carlos A. R. Junior, Thiago Rais de Castro, Solange N. Alves de Souza

Escola Politécnica da Universidade de São Paulo (EP-USP)
R. Prof. Luciano Gualberto, trav. 3, nº. 158 – 05508-900 – São Paulo – SP – Brazil

jr.rombaldo@usp.br, thiago.castro@poli.usp.br,
solange.souza@poli.usp.br

Abstract. *This article presents a model which describes the features to support the object concepts in the SQL:2003 standard. This model was based in the ontology was proposed to allow to understand the standard SQL:2003 more easily. This model also is used to guide exposition of the features in the SQL:2003 and in the Oracle 11g and PostgreSQL 8.4 DBMS. This approach give more complete view and better comprehension of the existing features in Object-Relational DBMS (ORDBMS), allowing its use in a lot of the real application. In the end, a study of case is used to compare the SQL:2003 standard and the previous ORDBMS.*

Resumo. *Neste artigo uma ontologia, proposta para facilitar o entendimento da especificação SQL:2003, é usada como base para a apresentação de um modelo que descreve as características voltadas para o suporte de objetos na SQL:2003. Este modelo também é usado para guiar a apresentação dessas características tanto na SQL:2003 quanto nas versões dos SGBDs Oracle 11g e PostgreSQL 8.4. Esta abordagem permite uma estruturação mais lógica, uma visão mais completa e o entendimento mais fácil das características existentes nos SGBDOR, permitindo seu real uso em muitas aplicações reais. Ao final, um estudo de caso permite a comparação da especificação SQL:2003 e dos SGBDORs utilizados.*

Introdução

Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR) mostraram-se pouco eficientes para o suporte às aplicações que envolvem a manipulação de dados complexos como aplicações científicas e médicas, de informações geográficas, dentre outras. Para suprir esta necessidade esforços foram empregados ao longo dos anos e culminaram na especificação da SQL:99 (ANSI/ISO/IEC 9075-x:1999) - revisão do padrão SQL:92 (base de SGBDRs). A especificação da SQL: 1999 foi impulsionada por vários SGBDs existentes que apresentavam algum grau de extensão para manipulação de objetos. A SQL:2003 (ANSI/ISO/IEC 9075:2003), revisão da SQL:99, define um conjunto de elementos ou recursos para a especificação de objetos dentro de banco de dados. Os SGBDs atuais que implementam a SQL:2003 ou parte dela são nomeados Sistemas Gerenciadores de Banco de Dados Objeto-Relacional (SGBDOR). Embora os recursos para manipulação de objetos já estejam disponibilizados por esses SGBDs, existe uma tendência em se utilizar apenas a parte relacional do padrão. Além disso, não existe uniformidade dos recursos implementados nestes SGBDs e nem em relação à especificação SQL:2003. Informações sobre esses recursos e seus inter-relacionamentos precisam ser difundidas. Por outro lado, os artigos que apresentam as novas características da norma, o fazem de maneira dispersa, sem a preocupação em mostrar a interdependência existente entre essas características. Além disso, poucos trabalhos mostram

como esses recursos estão disponibilizados em SGBDs existentes. Diferente dos demais, este artigo utiliza um modelo de classes UML, o qual foi baseado na ontologia proposta por Calero et al. (2005). O modelo aqui apresentado fornece uma visão geral dos elementos existentes para a definição de esquemas de objetos e seus inter-relacionamentos. Esta abordagem permite uma estruturação mais lógica, uma visão mais completa e o entendimento mais fácil das características existentes nos SGBDOR, permitindo seu real uso em muitas aplicações reais. O estudo de caso auxilia a compreensão para o uso desses elementos.

Este trabalho apresenta também os recursos disponíveis para o suporte de objetos nos SGBDs Oracle 11g release 1 e PostgreSQL 8.4. Outros trabalhos correlatos apresentaram o suporte oferecido por versões não atuais de SGBDOR e/ou basearam-se na especificação SQL:99.

Trabalhos Correlatos

No artigo de Eisenberg et al. (2004) são apresentadas algumas das novas funcionalidades da SQL:2003 relacionadas com a parte relacional do padrão. O Artigo não descreve como esses recursos estão disponibilizados em SGBDs existentes. A SQL:2003 é implementada de forma diferente entre os fabricantes de SGBDs, e há recursos disponibilizados em determinados SGBDs que não estão presentes em outros. Por conta disso, se faz necessário a investigação, não apenas dos novos elementos da SQL:2003, como também da forma de como estão implementados em SGBDs existentes.

Em Pardede et al. (2003) é discutido de forma isolada os aspectos da SQL:2003 referentes a parte orientada a objetos. Essa abordagem torna difícil a abstração do conceito associado a cada elemento e sua interdependência. Por outro lado, o universo relacional é bem difundido e sua implementação nos SGBDs em uso é quase que homogênea. Para que os novos recursos associados ao paradigma de orientação a objetos sejam largamente utilizados é preciso entender num primeiro momento que elementos existem, seus inter-relacionamentos e como estão disponíveis nos SGBDs. Posteriormente novos dados associados ao desempenho, por exemplo, serão também necessários.

Em Marcos, et al. (2003), Cavero, et al. (2001) e Vara, et al. (2007) é proposto uma extensão do diagrama de classes da UML para o Projeto Conceitual em BDORs. Nestes trabalhos discutem-se maneiras de se fazer o mapeamento de objetos e seus relacionamentos do diagrama de classes para a SQL:1999. Apesar do esforço por parte dos pesquisadores em discutir a parte OR do padrão, a fim de difundir suas vantagens e propor novos modelos para representá-la conceitualmente, ainda são necessários trabalhos para não apenas apresentar os recursos existentes na norma, mas fazê-lo de forma estruturada, mostrando o significado desses recursos, como estão inter-relacionados e como podem ser utilizados. Além disso, devido às diferenças de implementação é necessário mostrar como esses recursos, caso existam, estão disponíveis em SGBDs existentes.

SQL:2003

Para apresentar os elementos da SQL:2003 que fornecem o suporte a especificação de um BDOR, a ontologia desenvolvida por Calero et al. (2005) foi utilizada. Tal ontologia utiliza um diagrama de classes e regras expressas em OCL (*Object Constraint Language*) para permitir o entendimento dos elementos da linguagem e seus inter-relacionamentos, bem como o entendimento das diversas divisões da norma ISO/IEC 9075: 2003. No presente artigo apenas parte dos elementos dessa ontologia é utilizada. A figura A-1 mostra um diagrama de classes UML que tem como base o definido por Calero et al (2005). O diagrama da figura A-1 é mais conciso, fornecendo uma visão geral dos elementos da linguagem SQL:2003 para o

suporte de objetos. Conforme os conceitos são apresentados, realiza-se um comparativo entre as versões mais recentes dos gerenciadores Postgres 8.4 e Oracle 11g com objetivo de difundir a forma com que esses recursos são definidos.

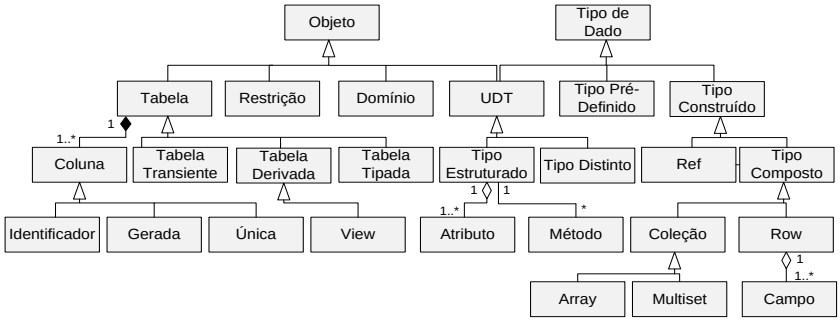

Figure A-1. Diagrama de Classes UML apresentado os elementos da SQL:2003 e seus inter-relacionamentos para suporte a BDOR

Ainda figura A-1, foram omitidas algumas associações, classes, além das regras expressas em OCL presentes na ontologia, pois o objetivo é usar um modelo que explique os elementos ou recursos existentes na norma e guie a apresentação estruturada e completa dos elementos da SQL:2003 para suporte a definição de esquemas objeto relacionais.

Similarmente à ontologia, o diagrama de classes da figura A-1 está dividido em duas sub-hierarquias, uma que traz os tipos de dados e outra, os tipos de objetos que podem ser definidos. Algumas classes (por exemplo, tipos Pré-Definidos, Restrição, Domínio, Tabelas Transientes e Derivadas) em cada uma das sub-hierarquias não são detalhadas neste artigo, pois apresentam nenhum, ou pouco acréscimo em relação às versões antigas da norma e não influenciam a definição de um esquema de objetos.

Tipo de Dado

Os Tipos de Dados são classificados em três classes principais: Tipo Pré-Definido, Tipo Construído e Tipo Definido pelo Usuário (*User Defined Type* - UDT).

Conforme figura A-1, os tipos construídos consistem em: tipos atômicos, como o REF, tipos complexos, como ROW, ou ainda coleções, como ARRAY e MULTISET. Um Row é uma estrutura composta por um ou mais campos. A tabela A-1 traz a definição de cada um desses tipos na SQL:2003 e como estão implementados nos SGBDs Oracle e PostgreSQL.

Um Tipo Definido pelo Usuário pode ser da categoria Tipo Distinto ou Tipo Estruturado. Um Tipo Estruturado é composto por um ou mais atributos, pode gerar hierarquias supertipo-subtipo, além disso podem ser definidos Métodos para manipular o Tipo Estruturado a que se refere. Um Tipo Estruturado pode ser utilizada para definir uma tabela tipada, ou para especificar o domínio de campos de uma tabela base (tabelas definidas por uma <definição de tabela>, similar a versões anteriores da norma) ou de uma tabela tipada (item 3.2.2).

Um Tipo Estruturado no Oracle 11g é similar à norma.

No PostgreSQL 8.4, como na norma, um Tipo Estruturado é composto por um ou mais atributos. Este tipo pode ser utilizado para definir domínios de colunas de tabelas.

Table A-1. Tipo Construído

Tipos	SQL:2003	ORACLE	PostegreSQL
REF	um valor REF é o valor de uma referência para uma tupla em uma tabela Tipada (a tupla é derivada de um tipo estruturado).	possui um tipo REF que é usado para referenciar um objeto através do seu OID (*Object Identifier*)	não suporta.
ROW	consiste de uma seqüência de campos, onde cada campo é definido pelo par (<nome_campo>, <tipo_de_dado>)	até versão atual do Oracle (11g release 1) não existe o tipo ROW.	não existe o tipo ROW
ARRAY	consiste de uma coleção de elementos do mesmo tipo de dado. Cada elemento está associado a uma posição ordinal. A cardinalidade determina o número máximo de elementos na coleção.	existe o tipo VARRAY. Quando um objeto do tipo VARRAY é definido, deve-se explicitar sua cardinalidade, no entanto seus campos são alocados dinamicamente.	existe o tipo Array, É posível definir array bi-dimensional.
MULTISET	consiste de uma coleção de elementos de mesmo tipo de dado. Os elementos não estão associados a uma posição ordinal.	não possui o tipo MULTISET.	não possui o tipo Multiset.

Objeto

De acordo com a figura A-1, um Objeto pode ser classificado em uma das seguintes classes: Tabela, Restrição, Domínio e UDT.

A classe Tabela (figura A-1) pode ser especializada nas classes Transiente, Derivada e Tabela Base (tabelas persistentes ou temporárias definidas a partir de uma <definição de tabela> - similar as versões antigas da norma). Uma tabela é composta por uma ou mais colunas.

Uma Tabela Tipada é definida por um Tipo Estruturado, ou seja, as linhas de uma Tabela Tipada são derivadas de um Tipo estruturado. Tabelas tipadas podem ser especializadas; para que uma tabela tipada possa estender outra tabela tipada, os tipos estruturados associados a essas tabelas devem pertencer à mesma hierarquia de especialização. Por exemplo, seja TT_1 uma tabela tipada associada ao tipo estruturado TE_1 e TT_2 uma tabela tipada associada ao tipo estruturado TE_2. A tabela tipada TT_1 poderá estender a TT_2 se, e somente se, TE_2 for supertipo direto do TE_1, ou seja, TE_2 deve aparecer na cláusula de generalização presente na especificação de TE_1.

No Oracle 11g uma tabela Tipada é definida a partir de um tipo Estruturado, similar a norma.

O Postgre 8.4 não oferece suporte a tabelas tipadas.

Herança

Na SQL:2003 relacionamentos de herança podem ser definidos entre tipos Tipos Estruturados, Tipos ROW e entre Tabelas. Na herança de tabelas, não apenas a estrutura como na herança de tipos, mas também os dados e restrições sobre os dados são herdados.

No Oracle não existe herança de tabelas. Neste, um tipo estruturado pode estender outro tipo estruturado. Tabelas tipadas podem ser definidas a partir de tipos na hieraquia, ou seja , criam-se os tipos de dados já definindo a hierarquia, na seqüência criam-se tabelas a partir dos tipos definidos. Pode-se incluir à definição da tabela restrições (*constraints*) e especificar valores *default*, contudo cabe ressaltar que essas restrições não são herdadas por tabelas definidas por uma extensão de um tipo estruturado, já que não existe herança de tabela.

O PostgreSQL 8.4 permite herança apenas entre tabelas, sendo possível também herança múltipla entre tabelas. Na herança de tabelas, dados também são herdados, porém restrições não o são. Registros inseridos numa subtabela podem ser acessados via a supertabela que a originou. Um problema observado, como as restrições não são herdadas, pode acontecer de ser inserido um ou mais registros na subtabela com valores iguais para as colunas que definem a chave primária da supertabela. Neste caso, ao acessar os dados da supertabela será possível se deparar com mais de um registro contendo valores iguais para as colunas que constituem a chave-primária.

Estudo de Caso

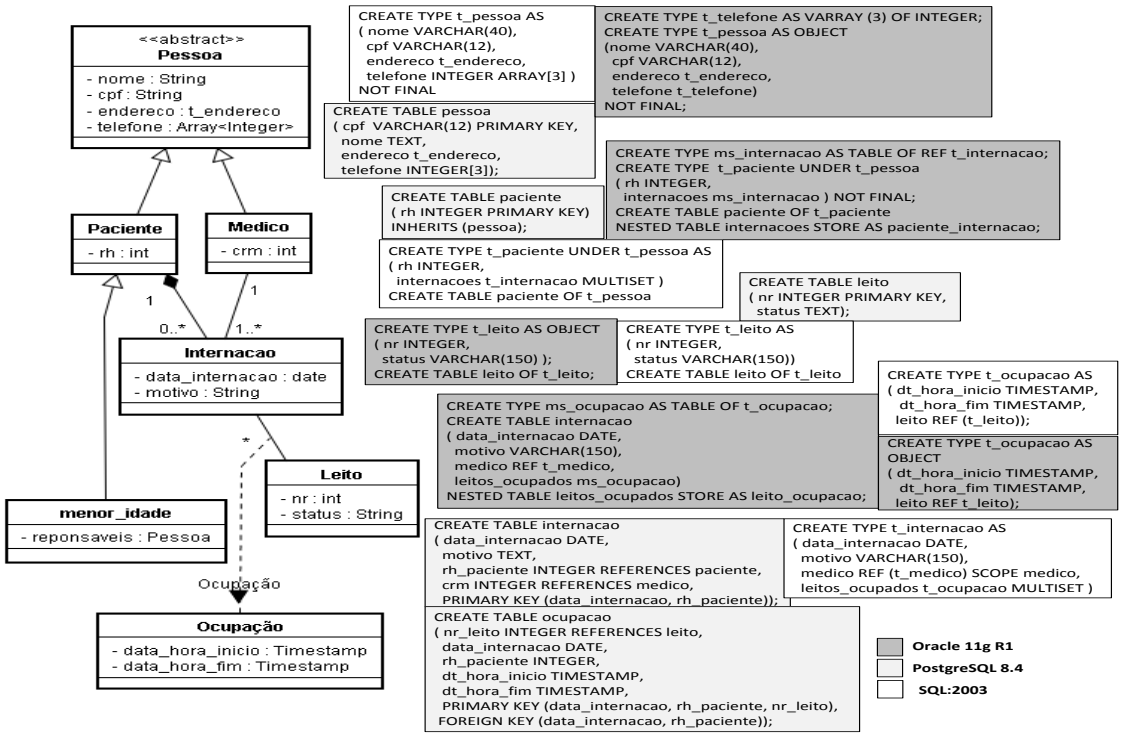

Figura A-2 – Modelo de classes UML para parte de uma aplicação hospitalar

A figura A-2 apresenta o modelo de classes UML e os respectivos códigos em SQL gerados, para melhor visualização, para parte do modelo. Este mostra a representação simplificada de internações de pacientes. Nesta, pacientes internados podem ocupar leitos diferentes durante a estadia, sendo necessário o registrado do momento de início e fim da ocupação do leito para

que o mesmo possa ser ocupado por outras internações em momentos diferentes. Todo paciente para ser internado precisa ter um registro hospitalar (rh) e se o paciente for menor de idade registra-se também um responsável.

O Oracle, como na norma, suporta a definição de tabelas a partir de tipos, permitindo a reutilização de código. Além disso, a definição de um tipo não determina a definição de tabelas daquele tipo (ex. classe Pessoa). Outro destaque, o suporte ao tipo REF elimina a necessidade de junções. Apesar de não possuir o tipo Multiset, é possível criar tabelas aninhadas através da cláusula NESTED TABLE.

O PostgreSQL não possui o tipo REF. Assim, é necessário especificar chaves estrangeiras para representar associações, agregações e composições. Como também não é possível a definição de tabelas de tipos, foi necessário criar a tabela Pessoa (figura A-2).

Neste artigo, devido a limitação de espaço, apresenta-se apenas o código para o mapeamento dos objetos do modelo de classes na base de dados considerando a SQL:2003 e os SGBDs Oracle e PostgreSQL.

Comentários e Conclusões

Este trabalho preocupou-se em apresentar as características da SQL:2003 juntamente com um comparativo entre os SGBDs Oracle e o PostgreSQL com relação ao suporte fornecido à SQL:2003. Observou-se que, comparado ao PostgreSQL, o Oracle permite um mapeamento para a base de dados mais próximo ao do modelo de classes, diminuindo o "descasamento de impedância" que existe para o modelo relacional. Cabe ressaltar que o MS SQL Server, versão 2008, também está sendo avaliado. Contudo não foi possível finalizar essa pesquisa, de modo a poder incluir seus resultados em tempo de finalização deste artigo.

Referências

Calero, C., Ruiz, F., Baroni, A., Brito, F. e Piattini, M. (2005) "An Ontological Approach to Describe the SQL:2003 Object-Relational Features." *Journal of Computer Standards & Interfaces - Elsevier*.

Eisenberg, A., J. Melton, K. Kulkarni, J. Michels e Zemke, F. (2004) "SQL:2003 Has Been Published." *SIGMOD Record*.

Pardede, E., Taniar, D. e Rahayu, J. W. (2004) "New SQL Standard For Object-Relational Database Applications." *Institute of Electrical and Electronics Engineers, Inc*.

Eisenberg, A.; Melton , J.(1999) "SQL:1999, formerly known as SQL3." *SIGMOD RECORD*.

Marcos, E.; Vela, B.; Cavero, J. M. (2003) "A Methodological Approach for Object-Relational Database Design using UML". *Software and Systems Modeling*.

Cavero, J. M.; Marcos, E.; Vela, B. (2001)"Extending UML for Object-Relational Database Design". *Lecture Notes in Computer Science*, pp. 225-239.

Vara, J. M.; Vela, B.; Cavero, J. M.; Marcos, E. (2007) "Model Transformation for Object-Relational Database Development". *Proceedings of the 2007 ACM symposium on Applied computing*, pp. 1012 – 1019.

APÊNDICE B – Artigo Publicado no SEMINCO 2010

Proposta de um Modelo para Projetos Lógicos Gráficos para Bancos de Dados Objeto-Relacionais (BDORs)

Thiago Rais de Castro[1], Solange N. Alves de Souza[1], Luiz Sergio de Souza[2]

[1] Escola Politécnica da Universidade de São Paulo – Departamento de Engenharia de Computação e Sistemas Digitais, São Paulo, Brasil (USP)

[2] Faculdades de Tecnologia, Carapicuíba, São Paulo, Brasil (FATEC)

thiago.castro@poli.usp.br,solange.souza@poli.usp.br, desouza.luizsergio@gmail.com

Resumo. *BDORs oferecem novos elementos que podem ser utilizados em projetos de uma base de dados, sendo necessário pesquisar o mapeamento de um modelo conceitual para o modelo lógico objeto-relacional. Este artigo apresenta um novo modelo lógico para BDOR. A SQL:2003 é a base para os elementos do modelo lógico. O modelo aqui proposto é baseado em dois outros trabalhos. Discutem-se as diferenças entre o modelo proposto e os anteriores. O modelo apresentado permite o mapeamento de um diagrama de classes UML (modelo conceitual) para elementos de um BDOR, bem como a elaboração de um modelo lógico gráfico do BDOR. Os elementos do modelo gráfico correspondem a elementos da SQL:2003.*

1. Introdução

Entende-se que a disponibilização de uma ferramenta CASE para modelagem conceitual de objetos persistentes e geração do modelo lógico do banco de dados, que tenham como base os elementos disponíveis em Banco de Dados Objeto-Relacionais (BDORs) é um recurso relevante para aumentar o uso dos recursos de objetos disponíveis em Sistemas Gerenciadores de Banco de Dados Objeto-Relacionais (SGBDORs). Neste contexto, este trabalho apresenta um novo modelo gráfico para a elaboração de projetos lógicos de BDORs. O modelo proposto é uma extensão aos trabalhos de [Golobsky and Vecchietti 2005] e de [Marcos and Vela and Cavero 2003] e [Vara and Vela and Cavero 2007], os quais propõem o mapeamento de elementos de um modelo de classes UML para elementos de um BDOR. As diferenças entre a proposta aqui apresentada e a dos trabalhos antecedentes são discutidas na Seção 2 deste artigo.

Este artigo está organizado da seguinte forma: a Seção 2 trata dos principais aspectos de trabalhos correlatos. Na Seção 3, apresentam-se modelos, tecnologias usadas e opções feitas para a geração do modelo lógico proposto. Na Seção 4 apresenta-se o modelo proposto e um exemplo de sua utilização. Finalmente na Seção 5 têm-se as conclusões e sugestões de continuidade.

2. Trabalhos Correlatos

Em [Vara and Vela and Cavero 2007] é apresentado uma arquitetura MDA (Model Driven Architecture) para o projeto de um BDOR. O trabalho utiliza a notação proposta por [Marcos and Vela and Cavero 2003] para a descrição de um Perfil UML e para apresentar as técnicas de mapeamento do Modelo Independente de Plataforma (PIM) para o Dependente de Plataforma (PSM). Contudo, as técnicas apresentadas não contemplam todos os possíveis relacionamentos existentes entre os elementos do Esquema de um BDOR, tais como: Associação N-ária (três ou mais classes), Classe Associativa etc.

[Feuerlicht and Pokorný and Richta 2009] discutem funções de mapeamento, contudo não acrescenta novidades em relação ao que já foi publicado pelos trabalhos aqui já citados.

Os trabalhos de [Golobsky and Vecchietti 2005] e de [Marcos and Vela and Cavero 2003] e [Vara and Vela and Cavero 2007] propõem o mapeamento do modelo de classes UML para BDOR. [Marcos and Vela and Cavero 2003] propõe também a representação gráfica do modelo lógico OR. Nesta, o diagrama de classes foi estendido para permitir a representação dos elementos de um BDOR.

Realiza-se, por meio da Tabela B-1, uma comparação entre o Modelo Lógico Gráfico proposto por [Marcos and Vela and Cavero 2003] e proposto neste trabalho e destacam-se as diferenças entre as propostas.

Tabela B-1: Estereótipos para o Modelo Lógico Gráfico.
primeira coluna: estereótipos propostos por [Marcos and Vela and Cavero 2003]; segunda coluna: estereótipos utilizados neste trabalho. Terceira coluna: diferenças entre as notações**.**

Marcos et al.	Este Trabalho	Diferença
«udt»	«udt»	Neste trabalho, o UDT definido no Esquema Lógico pode ser utilizado na especificação do domínio de campos e na definição de tabelas tipadas (como especificado na SQL:2003), diferente da proposta de Marcos et al. que utiliza-o apenas no domínio de campos.
«Object Type»	-	No trabalho de Marcos et al propõe-se o estereótipo «Object Type» para a definição de tabelas tipadas que possuem um UDT implicitamente vinculado. A representação de um UDT e de uma tabela tipada por meio de uma única notação não viabiliza o reúso do UDT na definição de outras tabelas tipadas. Por conta disso, optou-se por representar separadamente um UDT e uma tabela tipada. Para vincular a tabela tipada ao UDT deve-se utilizar uma associação marcada com o estereótipo «define».
«Knows»	-	Neste trabalho o estereótipo «Knows» não é incorporado, pois se existissem no modelo lógico referências mútuas entre duas classes, então duas associações «Knows», cada uma para uma das classes, seriam representadas no diagrama, tornando-o visualmente pesado e difícil de ser entendido. No modelo proposto, uma única linha de associação, coerentemente com o modelo de classes UML, é utilizada para representar o relacionamento.
«REF» «Array» «row»	-	O trabalho de Marcos et al propõe os estereótipos «REF», «Array» e «row» para a definição do domínio de atributos no esquema lógico. A especificação do domínio de um atributo por meio de um estereótipo introduz mais uma complexidade gráfica desnecessária no modelo lógico gerado. Além disso, um atributo para concretizar o relacionamento no modelo gráfico pode ser implicitamente representado por meio de relacionamentos e suas multiplicidades. Devido aos argumentos expostos, esses estereótipos não são incorporados no modelo gráfico aqui apresentado.
«row»	«row type»	Neste trabalho utiliza-se o estereótipo «row type» aplicado a classe, ao invés de aplicá-lo em atributos como proposto no trabalho de Marcos et al. Isto possibilita o reuso, pois um mesmo tipo linha poderá ser reutilizado para definir o domínio de atributos em diferentes classes.
-	«typed table»	Utiliza-se dessa notação para representar uma Tabela Tipada.
-	«define»	O estereótipo «define» é utilizado para associar uma tabela tipada ao UDT que a define.
-	«static»	No Modelo aqui proposto, utiliza-se a notação UML para os tipos de métodos definidos pela SQL:2003.
-	«constructor»	
-	«instance»	
«redef»	«overriding»	O estereótipo «redef» foi proposto para representar o conceito de polimorfismo, que na SQL:2003 é indicado pela palavra reservada *overriding*. Aqui se utiliza a definição da SQL:2003.
«def»	-	Como a especificação da SQL:2003 não contém o conceito de método abstrato, considerou-se tal estereótipo desnecessário ao modelo.

Os estereótipos «table», «pk», «fk», «not null», «unique» e «check» são incorporados ao modelo proposto, similarmente ao de Marcos et al, por corresponderem às restrições definidas pela SQL:2003.

Em relação ao mapeamento proposto neste trabalho (Tabelas B-4 e B-5) e as propostas de trabalhos de [Marcos and Vela and Cavero 2003] e [Vara and Vela and Cavero 2007], que serão referenciados como trabalho 1, e de Golobisky [Golobsky and Vecchietti 2005], que será referenciado como trabalho 2, ressaltam-se as seguintes diferenças:

1. Tanto no trabalho 1 quanto no trabalho 2 é sugerido que toda classe origine uma tabela tipada. Considera-se que esta abordagem generalista pode não ser a mais adequada para certos modelos de classes que apresentam particularidades associadas às regras de negócio da aplicação a que se refere. Por exemplo, se for considerado que classes abstratas foram definidas no modelo de classes. Tais classes não devem originar instâncias, portanto, a criação de uma tabela tipada associada a tal classe representará desperdício de espaço, sem falar nos problemas decorrentes da gerência e manutenção da base de dados, que pode apresentar, com o decorrer do tempo, um número significativo de tabelas que não apresentam dados. Por outro lado, investigando possíveis particularidades de um modelo de classes e de uma aplicação, entende-se que a geração do modelo lógico deve ser flexível, permitindo ao desenvolvedor e/ou projetista a decisão se uma classe deve ser representada (Tabela B-4) por um UDT (tipo estruturado), uma tabela tipada, uma tabela convencional (BDR), um tipo linha (*row type*), ou ainda apenas por um UDT (sem a criação de uma tabela tipada). O mapeamento para um tipo linha é sugerido para os casos onde não se pretende definir métodos associados à estrutura.
2. Nos trabalhos 1 e 2, métodos de classes são mapeados para métodos de um UDT. Diferentemente destes, buscando-se uma maior flexibilidade, aqui métodos de classes podem ser mapeados para métodos de UDTs, mas o desenvolvedor deve fazer essa indicação explicitamente. Considera-se que uma classe poderá ser mapeada para uma tabela convencional, para um tipo linha, para uma tabela tipada e/ou para um UDT, conforme discutido anteriormente. Assim, mesmo que o mapeamento corresponda a uma tabela tipada e/ou a um UDT caberá ao desenvolvedor definir quais métodos devem ser mapeados para métodos de objetos na base de dados. Entende-se que esta flexibilidade é fundamental no projeto lógico da base de dados, pois uma série de questões está envolvida nesse processo, dentre eles o desempenho da aplicação. Cabe ao desenvolvedor ou projetista a decisão sobre a forma de mapeamento de métodos.

3. Tecnologias e Modelos Empregados

A utilização de diagramas de classe UML na modelagem Conceitual em BD é uma alternativa ao MER [Navathe and Elmasri 2009] que permite o uso de um único modelo para o registro dos objetos, tanto persistentes quanto transientes. Isto otimiza a fase de desenvolvimento, já que elimina o esforço da construção de um modelo específico para a representação dos objetos persistentes, como também o da verificação da consistência necessária entre os modelos gerados.

Além da opção do uso de diagramas de classe como o modelo conceitual, optou-se também pela extensão do mesmo para a criação de esquemas lógicos gráficos para BDORs. Como tais diagramas já dispõem de elementos para representação de objetos, relacionamentos e restrições intrínsecas da OO, o esforço para a representação de elementos exclusivos de BDORs foi minimizado [OMG 2009].

No esquema da Figura B-1, mostra-se metamodelos e modelos envolvidos na geração do modelo lógico objeto-relacional aqui proposto. No esquema, níveis de abstração associados ao projeto de uma base de dados, são usados para apresentar os modelos e especificações correspondentes aos níveis conceitual e interno da arquitetura ANSI/SPARC. Destaca-se ainda na Figura B-1, o paralelo da arquitetura ANSI/SPARC com a arquitetura MDA.

O desenvolvimento MDA foca primeiro na funcionalidade e comportamento de uma aplicação ou sistema, sem se preocupar com as especificidades da plataforma, ou plataformas, na qual será desenvolvido. Desta forma, MDA separa os detalhes de implementação das funções do negócio. Sendo assim, não é necessário repetir o processo de definição da funcionalidade e comportamento de uma aplicação ou sistema cada vez que uma nova tecnologia surgir. Com a MDA, a funcionalidade e o comportamento são modelados uma única vez. O mapeamento do Modelo Independente de Plataforma (PIM) para um Modelo Dependente de Plataforma (PSM), fornecido pela plataforma MDA, pode ser realizado por ferramentas de modo a facilitar o suporte a novas ou diferentes tecnologias [OMG 2003].

Se for considerado que a partir de um modelo de classe, pode-se obter a geração do modelo lógico (nível conceitual / lógico) e a geração do modelo físico (nível interno), este último representado pelo código SQL específico para criação da base de dados em um SGBD produto (Oracle, Postgre, SQLServer etc.), então pode-se dizer que um ciclo de projeto de uma base dados reflete uma arquitetura MDA.

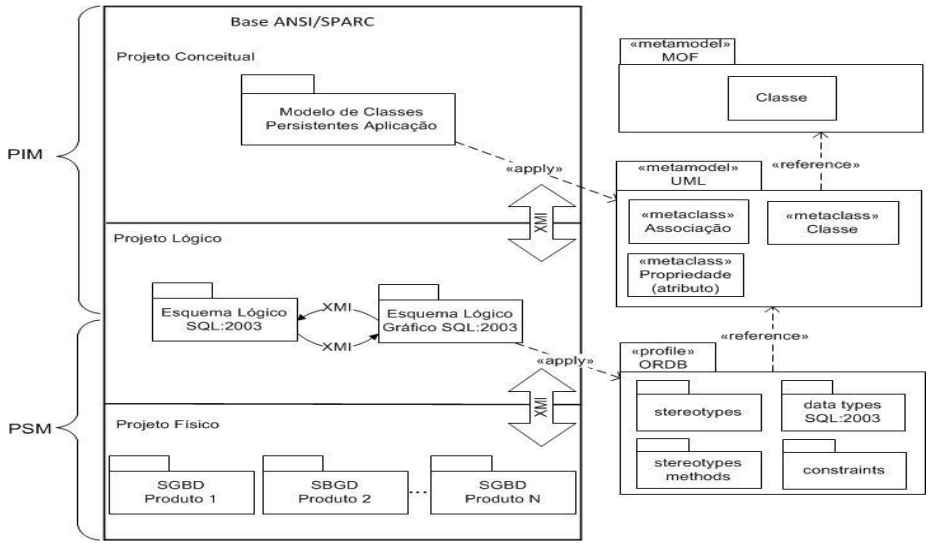

Figura B-2. Metamodelos e modelos envolvidos na geração do modelo lógico gráfico objeto-relacional proposto neste trabalho.

Ainda na Figura B-1, observa-se uma intersecção do PIM com o PSM no nível do Projeto Lógico. O PIM é caracterizado por ser independente de uma tecnologia específica [OMG 2003], nesse modelo o projetista preocupa-se apenas com os aspectos relacionados ao domínio do negócio a ser modelado. No PSM considera-se algum tipo de tecnologia [OMG 2003], sendo que no contexto do projeto de um BDOR o PSM pode ser desenvolvido visando-se algum SGBDOR produto. O nível do Projeto Lógico ilustrado na Figura B-1 não se refere a algum SGBOR produto (tal como o Projeto Físico). Contudo, nesta proposta utiliza-se a SQL:2003 que trata-se de uma especificação padrão para uma tecnologia específica: BDOR. Dado a dificuldade de discernimento entre classificar o nível do Projeto Lógico como sendo um PSM ou um PIM, justifica-se, tal como ilustrado na Figura B-1, a classificação desse nível como sendo a intersecção desses modelos.

O perfil ORDB corresponde à extensão da UML e os elementos definidos no perfil em conjunto com os da UML formam o Modelo Lógico Gráfico utilizado para o desenvolvimento de Esquemas Gráficos dos usuários (lado esquerdo da Figura B-1). Visando à separação lógica dos elementos, o perfil desenvolvido foi organizado em quatro subpacotes: *stereotypes*, *stereotypes methods*, *constraints* e *data types SQL:2003*. Esses subpacotes são detalhados na Seção 4.

O XMI (*XML Metadata Interchange*) é um padrão mantido pela OMG para definição, intercâmbio, manipulação e integração de dados e objetos por meio da Linguagem de Marcação Extensível (XML) [OMG, 2007]. O XMI provê regras por meio das quais um esquema pode ser gerado para qualquer metamodelo baseado em MOF[10] (*Meta-Object Facility*). A UML utiliza como metalinguagem de definição a MOF, sendo assim, é possível converter um Esquema UML em XMI. Um esquema XMI tem como principal benefício à portabilidade entre diferentes ferramentas CASE, desde que estas estejam de acordo com o padrão [OMG, 2007].

Para melhorar a interoperabilidade, propõe-se o emprego de XMI para o mapeamento em todos os níveis de abstração, conforme apresentado na Figura B-1. O esquema conceitual (diagrama de classes da UML) é facilmente traduzido para XMI [OMG 2009] [OMG 2007]. O mesmo acontece com o esquema lógico do SGBDOR. Com isso, tanto o esquema conceitual quanto o lógico em XMI poderão ser transportados entre ferramentas CASE.

4. Modelo Lógico Gráfico SQL:2003

No pacote *stereotypes* (Figura B-1) encontram-se os estereótipos aplicados em classes e relacionamentos do esquema lógico gráfico conforme detalhado na Tabela B-2. Nesta tabela, a primeira coluna corresponde ao nome do estereótipo, a segunda descreve o elemento da UML em que o estereótipo se aplica e a última descreve o significado do estereótipo.

Tabela B-2. Estereótipos do Modelo Lógico Gráfico.

Estereótipo	Elemento Base	Descrição
«table»	Classe	Define uma tabela [Navathe et al 2009].
«udt»	Classe	Define um Tipo Definido Pelo Usuário: pode ser composto por um ou mais atributos, ter métodos e gerar hierarquias supertipo-subtipo
«typed table»	Classe	Define uma Tabela Tipada: uma tabela definida a partir de um UDT.
«row type»	Classe	Define um Tipo Linha: estrutura composta por um ou mais campos[1].
«define»	Associação	Define uma Tabela Tipada a partir de um UDT.

[1] O Tipo Linha diferencia-se do Tipo Definido pelo Usuário nos seguintes aspectos: não suporta generalização-especialização, não possui comportamento (método) e não pode ser utilizado para definir tabelas.

O pacote *constraints* (Figura B-1) refere-se às restrições definidas pela SQL:2003 para colunas de tabelas. Como a informação de restrições deve fazer parte de um projeto lógico de uma base de dados, definiu-se os estereótipos "«pk», «fk», «not null», «unique» e «check»" para as *constraints*, os quais são aplicados sobre o elemento "atributo". O pacote *data types SQL:2003* dá suporte a tipos dos dados específicos definidos pela SQL:2003 e não serão detalhados aqui.

[10] Consulte (OMG, 2006) para obter uma descrição completa a respeito da MOF.

Tabela B-3. Estereótipos para especificação de Métodos.

Estereótipo	Elemento Base	Descrição
«static»	Method	Define um método estático em um UDT.
«constructor»	Method	Define um construtor em um UDT.
«instance»	Method	Define um método de instância, quando nenhum estereótipo for especificado, o componente assume que o método é de instância.
«overriding»	Method	Utilizado para redefinir (sobrescrever) um método herdado.

A SQL:2003 disponibiliza recursos para a definição de métodos em um UDT muito próximos (se não os mesmos) do que se tem no ambiente da aplicação, são eles: métodos de instância, estático e construtores. Estes são representados no modelo lógico proposto por estereótipos, conforme Tabela B-3. Além disso, é possível sobrescrever algum método da classe pai na classe filha possibilitando o polimorfismo, usando para isso a palavra reservada *overriding*. O Perfil proposto deve possuir elementos para identificar (ou diferenciar) os métodos de uma classe no modelo Lógico. Tal notação além de possibilitar uma fácil visualização e compreensão do esquema, possibilita também a distinção dos tipos de métodos, possibilitando dessa forma a geração de código automático para um determinado esquema. A Tabela B-3 apresenta os elementos pertencentes ao pacote *stereotypes methods* (Figura B-1).

As Tabelas B-4 e B-5 apresentam a proposta de mapeamento do modelo de classes UML (modelo conceitual) para o modelo lógico OR.

Tabela B-4. Mapeamento do diagrama de classes (classes, atributos e métodos) para o modelo lógico OR.

Elemento no Diagrama de Classes	**Correspondente no Esquema Lógico de um BDOR**	**Justificativa**
Classe	Tabela (*table*)	O modelo lógico deve ser flexível, permitindo ao projetista a decisão de como uma classe deve ser representada, adequando-se assim às diferentes características dos negócios.
	UDT	
	Tabela Tipada (*typed table*)	
	Tipo linha (*row type*)	
Classe abstrata	UDT	Uma classe abstrata não pode ser instanciada, podendo ser utilizada na definição de uma classe concreta (que pode ser instanciada). Sugere-se a utilização de um UDT sem a criação de uma tabela tipada associada, pois o UDT poderá ser utilizado na definição de outros UDTs, e como não possui tabela tipada associada, não haverá instâncias sendo persistidas
Atributo simples	Tipo Primitivo (*build-in*)	A SQL:2003 dá suporte a diversos tipos primitivos, como o tipo inteiro, real etc. Dessa forma, é possível achar um correspondente em SQL para cada tipo simples definidos pela UML.
Atributo composto	Tipo Linha (*row type*)	O mapeamento para um tipo linha é sugerido para os casos onde não se pretende definir métodos associados à estrutura, tal como os atributos compostos.
Atributo multivalorado	*Array* ou *Multiset*	Essas estruturas multidimensionais são adequadas para armazenar atributos de um mesmo tipo (coleções).
Métodos	Métodos de um UDT	Os UDTs suportam a declaração de métodos em sua definição, dessa forma, sugere-se sua utilização quando existirem métodos específicos a determinada classe. Caso exista uma função comum a outras classes, podem-se definir gatilhos (*triggers*).

Tabela B-5. Mapeamento do diagrama de classes (relacionamentos) para o modelo lógico OR.

<table>
<tr><td colspan="2">Associação</td><td colspan="2">Correspondente no Esquema Lógico de um BDOR</td></tr>
<tr><td rowspan="2">Associação Bidirecional</td><td rowspan="2">Composição/ Agregação/ Associação</td><td>1..1</td><td>Define-se uma Referência cruzada, ou seja, cada classe mantém a referência (REF) para a outra.</td></tr>
<tr><td>1..*[1]</td><td>Define-se também uma referência cruzada, porém a classe agregadora terá um Array ou um Multiset de Referências.</td></tr>
<tr><td colspan="2">Associação Unidirecional</td><td colspan="2">Similares às respectivas associações bidirecionais apresentadas acima, no entanto a referência(s) irá existir em apenas uma tabela.</td></tr>
<tr><td colspan="2">Associação N-ária (três ou mais classes)</td><td colspan="2">Define-se uma tabela ou um UDT com o nome do papel da associação. A tabela ou o UDT (e posteriormente a tabela tipada) deve referenciar as classes envolvidas.</td></tr>
<tr><td colspan="2">Classe Associativa</td><td colspan="2">Similarmente a Associação N-ária, pode-se definir uma tabela ou um UDT para a classe de associação.</td></tr>
<tr><td colspan="2">Generalização-Especialização</td><td colspan="2">Define-se um UDT para cada classe da hierarquia.</td></tr>
</table>

[1]Se for conhecido o valor da multiplicidade *, então se sugere um *Array*, em caso contrário, um *Multiset*.

4.2. Estudo de Caso

Nesta seção, apresenta-se um estudo de caso para o desenvolvimento de um sistema simplificado para uma loja online especializada em produtos de informática. Para esse projeto, utilizou-se a arquitetura apresentada na Seção 3, o Modelo Lógico Gráfico proposto e as técnicas de mapeamento discutidas na Seção 4.

O objetivo desse estudo de caso é prover um exemplo prático de um projeto em BDORs, em que se inicia pela fase de concepção do sistema e estende-se até a fase de geração de código em SQL:2003 e no dialeto SQL utilizado pelo Oracle 11g.

Os seguintes requisitos foram identificados na fase de concepção do sistema:

6) A loja trabalha com a venda de produtos nos seguimentos de software e hardware;
7) Clientes cadastrados, identificados por nome e CPF, podem realizar compras dos produtos disponíveis no site da loja;
8) O sistema deve armazenar o endereço (número e a rua) dos clientes cadastrados;
9) Os produtos no seguimento de software apresentam o número de versão. Já os produtos no seguimento de hardware apresentam a montadora;
10) O sistema deve possibilitar a geração de relatórios tanto para clientes, relacionando as compras realizadas em um determinado período, como para gerência.

De acordo com a arquitetura da Figura B-1, inicia-se o projeto de um BDOR com o desenvolvimento de um esquema conceitual utilizando-se o Diagrama de Classes da UML. O esquema conceitual para o estudo de caso é apresentado na Figura B-2. O mapeamento para o projeto lógico é apresentado na Figura B-3.

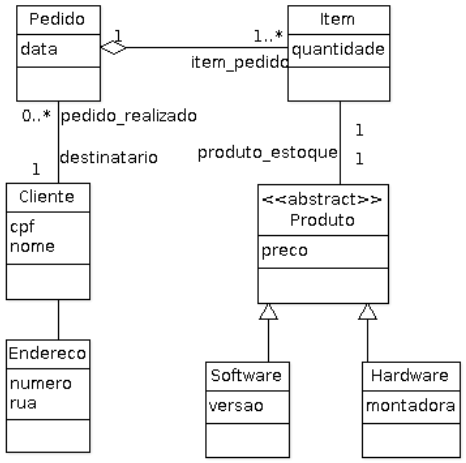

Figura B-2. Exemplo de um Projeto Conceitual.

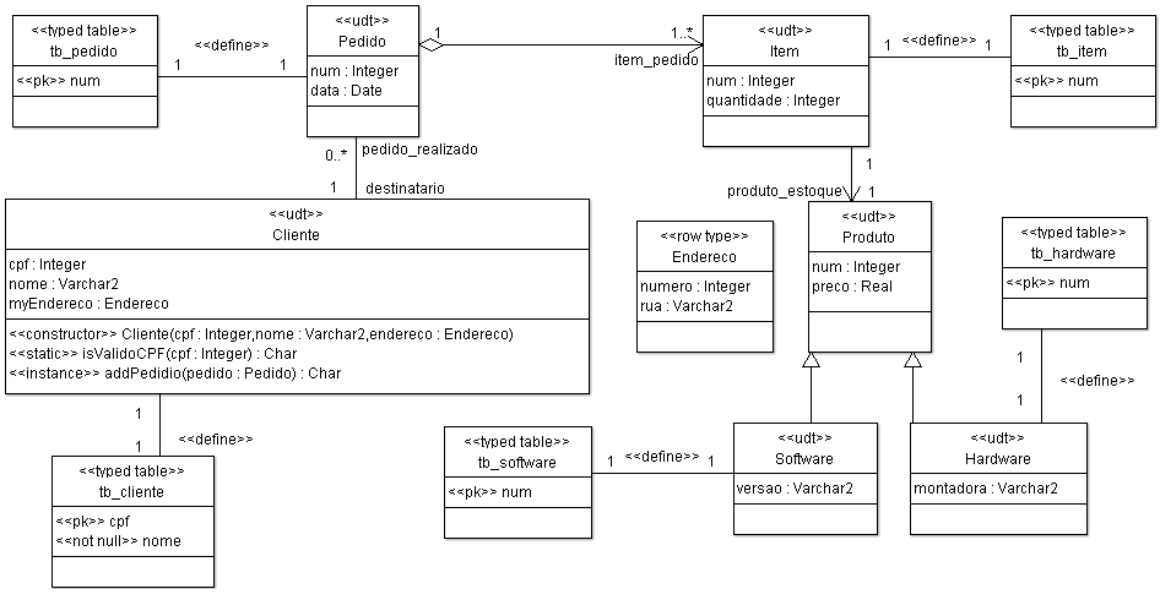

Figura B-3. Exemplo de um Projeto Lógico Gráfico.

Optou-se por gerar o código (Tabela B-6) em SQL:2003, pois a SQL definida pelo padrão pode ser adaptada para qualquer dialeto SQL utilizado por um SGBDOR produto, como o PostgreSQL, Oracle ou SQL Server por exemplo. Para demonstrar a geração do projeto lógico em código SQL de um SGBD produto optou-se pelo Oracle 11g [Oracle 2010], pois ele dá suporte à maioria dos elementos definidos pelo padrão SQL.

O código em SQL:2003 e no dialeto SQL do Oracle 11g são apresentados na Tabela B-6 e encontra-se na última fase do projeto de um BDOR.

Tabela B-6. Código SQL correspondente ao esquema ilustrado na Figura B-3.

SQL:2003	Oracle 11g
CREATE TYPE Produto AS (num INTEGER, preco REAL) NOT FINAL	CREATE TYPE Produto AS OBJECT (num INTEGER, preco REAL)NOT FINAL
CREATE TYPE Software UNDER Produto AS (versao VARCHAR2) FINAL CREATE TABLE tb_software OF Software(num WITH OPTIONS PRIMARY KEY)	CREATE TYPE Software UNDER Produto (versao VARCHAR2(15)) FINAL CREATE TABLE tb_software OF Software(num PRIMARY KEY)
CREATE TYPE Hardware UNDER Produto AS (montadora VARCHAR2)FINAL CREATE TABLE tb_hardware OF Hardware(num WITH OPTIONS PRIMARY KEY)	CREATE TYPE Hardware UNDER Produto (montadora VARCHAR2(30))FINAL CREATE TABLE tb_hardware OF Hardware(num PRIMARY KEY)
CREATE TYPE Item AS (num INTEGER, quantidade INTEGER, produto_estoque REF(Produto))FINAL CREATE TABLE tb_item OF Item(produto_estoque WITH OPTIONS NOT NULL, num WITH OPTIONS PRIMARY KEY)	CREATE TYPE Item AS OBJECT (num INTEGER, quantidade INTEGER, produto_estoque REF Produto)FINAL CREATE TABLE tb_item OF Item(num PRIMARY KEY)
CREATE TYPE Pedido AS (num INTEGER, data DATE, destinatario REF(Cliente), item_pedido REF(Item) MULTISET)FINAL	CREATE TYPE item_nested_table_type AS TABLE OF REF Item CREATE TYPE Pedido AS OBJECT (num INTEGER, data DATE, destinatario REF Cliente, item_pedido item_nested_table_type)FINAL
CREATE TABLE tb_pedido OF Pedido(destinatario WITH OPTIONS NOT NULL, item_pedido WITH OPTIONS NOT NULL, num WITH OPTIONS PRIMARY KEY)	CREATE TABLE tb_pedido OF Pedido(num PRIMARY KEY) NESTED TABLE item_pedido STORE AS item_pedido_nested_table
Endereco ROW(numero INTEGER, rua VARCHAR2)	CREATE TYPE Endereco AS OBJECT (numero INTEGER, rua VARCHAR2(30)) FINAL
CREATE TYPE Cliente AS (cpf INTEGER, nome VARCHAR2(30), myEndereco ENDERECO, pedido_realizado REF(Pedido) MULTISET)FINAL CONSTRUCTOR METHOD Cliente(cpf INTEGER, nome VARCHAR2, endereco ENDERECO) RETURNS SELF AS RESULT, STATIC METHOD isValidoCPF(cpf INTEGER) RETURNS CHAR, INSTANCE METHOD addPedidio(pedido PEDIDO) RETURNS CHAR	CREATE TYPE pedido_nested_table_type AS TABLE OF REF Pedido CREATE TYPE Cliente AS OBJECT (cpf INTEGER, nome VARCHAR2(30), myEndereco ENDERECO, pedido_realizado pedido_nested_table_type, CONSTRUCTOR FUNCTION Cliente(cpf INTEGER, nome VARCHAR2, endereco ENDERECO) RETURN SELF AS RESULT, STATIC FUNCTION isValidoCPF(cpf INTEGER) RETURN CHAR, MEMBER FUNCTION addPedidio(pedido PEDIDO) RETURN CHAR) FINAL
CREATE TABLE tb_cliente OF Cliente(nome WITH OPTIONS NOT NULL, cpf WITH OPTIONS PRIMARY KEY)	CREATE TABLE tb_cliente OF Cliente(nome NOT NULL, cpf PRIMARY KEY) NESTED TABLE pedido_realizado STORE AS pedido_realizado_nested_table

Com relação ao esquema lógico gráfico ilustrado pela Figura B-3 e o seu respectivo código SQL (Tabela B-6), têm-se as seguintes considerações:

1. Devido ao fato de em um UDT não ser possível declarar restrições em seus atributos, estas devem ser declaradas nos atributos apropriados da tabela tipada.
2. O relacionamento de composição entre Pedido (classe UDT todo) e Item (classe UDT parte) é implementado por meio de um *Multiset* de referências na classe todo. O *Multiset* é utilizado, pois a multiplicidade do relacionamento é * (limite superior da multiplicidade não é definido).
3. O Oracle 11g não garante a integridade referencial em *Multisets* e *Arrays* de referências, caso ocorra uma tentativa de acesso a uma referência que não existe o valor nulo será

retornado. Sendo assim, cabe ao projetista a definição de métodos ou *triggers* (gatilhos) para garantir a integridade referencial nessas estruturas.

4. O Oracle 11g define o *Multiset* de forma diferente da definida pela SQL:2003 e conseqüentemente, diferente do modelo lógico gráfico proposto. Observa-se, por exemplo, a classe Pedido; no código em SQL:2003 (Tabela B-6 e na Figura B-3), Pedido possui um *Multiset* de referências denominado item_pedido, já o código correspondente no Oracle 11g a definição de um *Multiset* é feita por meio da criação de um tipo tabela de referências denominado item_nested_table_type. Outra diferença entre o padrão SQL:2003 e o Oracle 11g está na definição de uma tabela tipada a partir de um UDT que possui um *Multiset*. Ao definir-se uma tabela tipada, cada atributo do tipo tabela de referências (*Multiset*) deverá ser explicitamente declarada, observe que na definição da tabela tipada tb_pedido, declarou-se uma tabela aninhada (Nested Table) denominada item_pedido que será armazenada fisicamente com o nome item_pedido_nested_table.
5. A classe Endereco do esquema conceitual foi mapeada para um tipo linha no esquema lógico, pois essa classe não apresenta métodos na sua definição e não haverá necessidade de acessá-la individualmente, ou seja, no estudo de caso o acesso a Endereco está estritamente relacionado ao acesso a Cliente, dessa forma, julgou-se apropriado a utilização de uma estrutura aninhada. Ressalta-se que se definiu o atributo myEndereco em Cliente (Figura B-3) cujo domínio é do tipo Endereco.
6. A classe abstrata Produto foi mapeada para um UDT sem uma tabela tipada associada, pois, como discutido na Seção 4, classes abstratas não podem ser instanciadas, logo a criação de uma tabela para persistir objetos dessa classe corresponderia a desperdício de espaço.
7. As demais classes (Software, Hardware, Item, Pedido e Cliente) foram mapeadas para UDTs com tabelas tipadas associadas, pois essas classes são instanciáveis e seus objetos poderão ser persistidos, além disso, com essa implementação é possível utilizar os recursos disponibilizado pela SQL:2003, são eles: criação de métodos, herança, polimorfismo e criação de referências.
8. Como o Oracle 11g não suporta o tipo linha, optou-se por utilizar um UDT sem a definição de métodos (comportamento), além disso, devido ao fato do tipo linha não poder ser estendido, definiu-se o UDT como final, ou seja, não é possível estendê-lo.

5. Conclusões e Trabalhos Futuros

Neste trabalho propõe-se um modelo para o Projeto Lógico Gráfico para BDOR considerando-se a SQL:2003. Tal modelo consolida-se por meio do desenvolvimento de um Perfil UML, o qual incorpora os elementos existentes na SQL:2003 fazendo uso de estereótipos. Um esquema desenvolvido utilizando o perfil proposto permite uma fácil visualização e compreensão do esquema.

Como continuidade deste trabalho, a ferramenta CASE ArgoUML está sendo estendida. A extensão permitirá tanto a elaboração direta de um projeto lógico gráfico para um BDOR, com base na SQL:2003 e sua tradução automática para o respectivo código SQL, quanto a geração do modelo lógico a partir do modelo de classes UML. A geração automática do código SQL para SGBDs produtos específicos faz parte da proposta. Contudo, para o caso do modelo lógico seja gerado a partir do diagrama de classes é necessário ainda estabelecer uma notação específica para marcar os elementos deste, já que uma classe pode ser mapeada para mais de um elemento no modelo lógico. Isto permitirá que a ferramenta CASE possa identificar as marcações e fazer o mapeamento adequado ao modelo elaborado.

Ainda se faz necessário a investigação de aspectos relacionados à performance dos recursos incorporados à SQL:2003 e também do impacto decorrente desses recursos no desenvolvimento da aplicação e do projeto de BDs.

6. Referências

Golobisky, M. F. and Vecchietti, A. (2005). Mapping UML Class Diagrams into Object-Relational Schemas. Rosario, Argentina: Proceedings of ASSE, p 65-79.

Marcos, E. and Vela, B. and Cavero, J. M. (2003). A Methodological Approach for Object-Relational Database Design using UML. Heidelberg: Springer Berlin, p59-72.

Vara, J.M. and Vela, B. and Cavero, J.M. and Marcos, E. (2007). Model Transformation for Object-Relational Database Development. New York: ACM, p 251-258.

Feuerlicht,G. and Pokorný,J. and Richta, K. (2009), Object-Relational Database Design: Can Your Application Benefit from SQL:2003? Galway, Ireland: Springer, p. 1-13.

Navathe, B. S. and Elmasri, R. (2009), Sistemas de Banco de Dados. São Paulo: Pearson Education.

OMG. MDA Guide Version 1.0.1 - Document Number: omg/2003-06-01. Disponível em: < http://www.omg.org/cgi-bin/doc?omg/03-06-01> (2003).

OMG. Meta Object Facility (MOF) Core Specification - Version 2.0. Disponível em: < http://www.omg.org/spec/MOF/2.0/PDF/> (2006)

OMG. MOF 2.0/XMI Mapping - Version 2.1.1. Disponível em: < http://www.omg.org/cgi-bin/doc?formal/07-12-02> (2007)

OMG.: Unified Modeling Language - version 2.2. Disponível em: < http://www.omg.org/spec/UML/2.2/Infrastructure/PDF/ (2009).

Oracle. Disponível em: <http://www.oracle.com>. Acessado em 20 de setembro de 2010.