Regras de Transformação Baseada na M2T para Geração de Linguagem de Definição de Dados

Technical Report · December 2012

DOI: 10.13140/RG.2.2.34119.39844/1

2 authors:

André Rosa
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
12 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Carlos Eduardo Pantoja
Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
121 PUBLICATIONS **215** CITATIONS
SEE PROFILE

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA

CELSO SUCKOW DA FONSECA

UNIDADE DE ENSINO DESCENTRALIZADA DE NOVA FRIBURGO/RJ

ANDRÉ DE SOUZA ROSA
andre_souza.rosa@hotmail.com

ITALINE DA SILVA GONÇALVES
italine.goncalves@hotmail.com

REGRAS DE TRANSFORMAÇÃO BASEADAS NA M2T PARA GERAÇÃO DE LINGUAGEM DE DEFINIÇÃO DE DADOS

Nova Friburgo

2012

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA

CELSO SUCKOW DA FONSECA

UNIDADE DE ENSINO DESCENTRALIZADA DE NOVA FRIBURGO/RJ

ANDRÉ DE SOUZA ROSA
andre_souza.rosa@hotmail.com

ITALINE DA SILVA GONÇALVES
italine.goncalves@hotmail.com

REGRAS DE TRANSFORMAÇÃO BASEADAS NA M2T PARA GERAÇÃO DE LINGUAGEM DE DEFINIÇÃO DE DADOS

Trabalho de conclusão de curso apresentado ao CEFET Nova Friburgo como requisito parcial para a conclusão do Curso Técnico de Informática

Nova Friburgo

2012

CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA CELSO SUCKOW DA FONSECA

UNIDADE DE ENSINO DESCENTRALIZADA DE NOVA FRIBURGO/RJ

Curso: Técnico em Informática

Ano: 2012

Aluno 1: André de Souza Rosa

Aluno 2: Italine da Silva Gonçalves

Título do TCC: Regras de Transformação Baseadas na M2T para a Geração de Linguagem de Definição de Dados

Orientador: Carlos Eduardo Pantoja

Nota:

Recomendações:

_____
_____
_____
_____
_____
_____

_____

(Docente Orientador)

Nova Friburgo, ____ de _____ de _____

Trabalho de Conclusão de Curso, Núm. XX/AA ISSN: XXXX-XXXX

REGRAS DE TRANSFORMAÇÃO BASEADAS NA M2T PARA GERAÇÃO DE LINGUAGEM DE DEFINIÇÃO DE DADOS

André de Souza Rosa

Italine da Silva Gonçalves

Resumo. Este trabalho propõe uma abordagem utilizando a Arquitetura Orientada a Modelos (*Model Driven Architecture* – MDA) que tem como objetivo automatizar as etapas do processo de modelagem de banco de dados. É definido um metamodelo de acordo com a abordagem MDA que contem os conceitos comuns presentes nas notações de modelagem conceitual voltadas para Sistemas Gerenciadores de Banco de Dados (SGBD) relacionais em sua definição, possibilitando aderência a modelos definidos com base nas distintas notações existentes como Entidade-Relacionamento, *Crow's Foot*, Definição de integração para Modelagem de Informações (*Integration Definition for Information Modeling* - IDEF1X) e a Linguagem de Modelagem Unificada (*Unified Modeling Language* –UML). É criado um conjunto de regras de transformação utilizando a linguagem *MOF Model To Text Transformation Language* (MOFM2T). Essas regras são responsáveis por transformar os modelos, aderentes ao metamodelo que está sendo proposto, em código de Linguagem de Definição de Dados (*Data Definition Language* – DDL), formado do núcleo compatível com os padrões ANSI SQL 92, 99 e 2003, que será utilizado para a implementação da base de dados desejada em um SGBD. A ferramenta implementada para testes da abordagem é um conjunto de *plugins* para o ambiente de desenvolvimento Eclipse, que possibilita a instanciação de notações distintas de modelagem conceitual. O metamodelo foi construído utilizando o Ecore, parte integrante EMF, e as regras de transformação forma implementadas utilizando o Acceleo, gerador de código que utiliza a linguagem MOFM2T.

Palavras-chave: Modelagem Conceitual, Arquitetura Orientada por Modelos, Transformação de Modelos para texto.

Abstract. This work proposes a Model Driven Architecture (MDA) approach that aims to automate the steps of database modeling process. It was defined a metamodel according to the MDA approach which contains common concepts of conceptual modeling notations used in relational databases, enabling its use for distinct modeling notations, such as Entity-Relationship, Crow's Foot, Integration Definition for Information Modeling (IDEF1X) and Unified Modeling Language (UML). It was created a set of transformation rules using MOF Model To Text Transformation Language (MOFM2T), that is capable of transform the acceptable models in the proposed metamodel to the standard ANSI SQL 92, 99 and 2003 DDL code, which will be used to implement the database in the desired Database Management System (DBMS). The tool created for testing of the approach is a set of plugins for Eclipse development environment, where is capable of instantiate different conceptual modeling notations. The metamodel was constructed using Ecore, that is part of EMF and the transformation rules were implemented using Acceleo, which is a code generator that uses the MOFM2T language.

Keywords: Conceptual Modeling, Model-Driven Architecture, Model-To-Text Transformation.

Responsável por publicações:

<Inserir Nome da Responsável Aqui>
Biblioteca do CEFET UnED Nova Friburgo
Av. Governador Roberto Silveira, 1900 – Prado
28.635-000 Nova Friburgo RJ Brasil
Tel. +55 22 2527-1727
E-mail: biblionf@gmail.com

LISTA DE SIGLAS E ABREVIATURAS

ANSI	*American National Standards Institute*
CIM	*Computation Independent Model*
DBMS	*Database Management System*
DDL	*Data Definition Language*
DER	Diagrama Entidade-Relacionamento
DML	*Data Manipulation Language*
EER	Entidade-Relacionamento Estendido
EMF	*Eclipse Modeling Framework*
ER	Entidade-Relacionamento
GMF	*Graphical Modeling Framework*
IDEF1X	*Integration Definition for Information Modeling*
ISO	*International Organization for Standardization*
M2T	*Model To Text*
MDA	*Model-Driven Architecture*
MER	Modelo Entidade-Relacionamento
MOF	*Meta-Object Facility*
NIST	*National Institute of Standards and Technology*
ODBC	*Open Data Base Connectivity*
OMG	*Object Management Group*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	*Structured Query Language*
TCC	Trabalho de Conclusão de Curso
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

Lista de Figuras

Lista de tabelas

Sumário

1 Introdução

Um banco de dados consiste em um conjunto de informações, organizadas de forma lógica, podendo conter diversos tipos de elementos, desde elementos numéricos e textuais a imagens, vídeos e músicas, o que torna seu uso indispensável no armazenamento da grande quantidade de informações disponíveis para que o usuário recupere tais informações e as utilize de forma a sanar suas necessidades (Heuser, 2009).

O gerenciamento das informações na base de dados se dá por meio do Sistema Gerenciador de Banco de Dados (SGBD), que é um conjunto de aplicativos o qual é responsável pela inserção, edição, recuperação e exclusão dos dados na base e deve também fornecer mecanismos que garantam a integridade, a disponibilidade e a confidencialidade dos dados.

A implementação de uma base de dados requer que suas características sejam específicas, bem como quais dados serão armazenados e de que forma isso será feito. Essa fase de especificação do banco, onde o mesmo é projetado, é denominada modelagem de banco de dados. Um projeto de banco de dados é composto por três fases distintas: a modelagem conceitual, o projeto lógico e o projeto físico (Heuser, 2009).

A modelagem conceitual representa a estrutura do banco de dados considerando o ponto de vista do usuário. O projeto lógico representa a estrutura com algumas considerações tecnológicas, ou seja, possui alguns detalhes da implementação do banco. Já o projeto físico é a implementação do banco de dados, considerando todos os detalhes tecnológicos pertencentes ao SGDB utilizado. A transição entre essas fases até a efetiva implementação de um banco de dados em um SGBD é um processo que demanda tempo e esforço por parte do projetista de banco de dados (Heuser, 2009) .

A modelagem conceitual é independente de aspectos tecnológicos, pois é a representação dos dados em um alto nível de abstração (Abreu; Machado, 1999), ela não considera a implementação do banco de dados em si, considera a forma como é criada a estrutura de armazenamento. Utiliza-se, para isso, um diagrama contendo vários elementos com o propósito de demonstrar os dados do banco, divididos entre entidades, atributos e relacionamentos. Os elementos presentes nesses diagramas, seus comportamentos e outras características são definidas de acordo com a notação de modelagem conceitual utilizada. Existem distintas notações para modelagem con-

ceitual de banco de dados relacional, como a de Entidade-Relacionamento (Chen, 1976) , Definição de integração para Modelagem de Informações (*Integration Definition for Information Modeling* - IDEF1X) (Hay, 1999), *Backman* (Hay, 1999), *Crow'sFoot* (Halpin, 1999), *Min-Max* (Hay, 1999) e a Linguagem de Modelagem Unificada (*Unified Modeling Language – UML)*, que não é específica para modelagem de banco de dados, mas é amplamente utilizada (Guedes, 2008).

1.1 Problema

Há soluções que fazem a automação da modelagem conceitual da base de dados até a implementação da mesma no SGBD. O problema é que essas soluções geralmente possuem atrelamento a determinadas notações de modelagem conceitual, que em alguns casos, é uma notação de modelagem de dados específica, uma linguagem de modelagem própria da solução ou modificada de outras (Rosa et al., 2012) .

Também existe o problema de as soluções serem direcionadas, muitas vezes, a um SGBD específico, limitando também o poder de escolha do projetista de banco de dados que deseja utilizar aquela solução, o obrigando a direcionar sua modelagem ao SGBD do qual ela suporta (Rosa et al., 2012).

Alguns exemplos de soluções utilizadas para projetar bancos de dados que possibilitam a automação da modelagem da base de dados que possuem os atrelamentos citados são o DBDesigner (DbDesigner homepage, 2012), que apesar de não ser mais um projeto ativo atualmente, possui uma interface que permite a utilização de até três notações de modelagem conceitual diferentes: a E-R, a EE-R e a de Crow's Foot, porém é direcionada especificamente para o MySQL, a ferramenta Visual Database Tools, que possui uma notação para modelagem conceitual particular e é voltado para MS SQL Server (Visual Database Tools, 2012) e o DBExplorer (DBExplorer, 2012), que é versátil, mas não é livre.

1.2 Objetivo

Esse trabalho propõe uma metodologia construída com base na Arquitetura Orientada a Modelos (*Model-Driven Architecture*-MDA), na qual partindo de uma modelagem conceitual qualquer voltada para SGBD relacional, gera-se código ANSI-SQL DDL de forma automática.

Conforme demonstrado na figura 1, é utilizado um metamodelo, cujo o mesmo foi definido manualmente, observando os conceitos comuns presentes nas notações

de modelagem conceitual utilizadas em bancos de dados relacionais, de forma que, se uma modelagem feita possuir na definição da sua estrutura os mesmos conceitos definidos no metamodelo, ela passará pelo processo de transformação através das regras utilizando a linguagem MOFM2T (OMG, 2008), dando então origem a artefatos de texto de código ANSI-SQL DDL compatível com o núcleo dos padrões 92, 99, 2003, que poderá ser utilizado para a implementação da base de dados em SGBD que sejam compatíveis com o padrão.

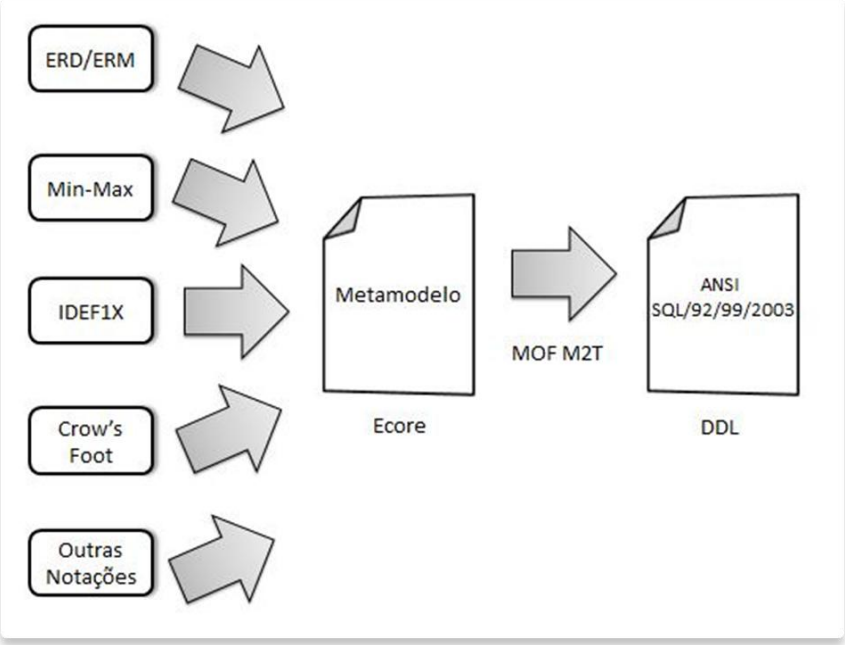

Figura 1 - Abordagem proposta (ROSA et. al, 2012)

Visa-se com esse trabalho, portanto, construir um metamodelo que permita a definição de modelos que contenham características de distintas notações de modelagem conceitual e também a definição de regras de transformação que transformem um modelo definido através de um arquivo no formato *eXtensible Markup Language* (XML) de forma visual, utilizando para isso uma árvore hierárquica, em um artefato de texto contento código de Linguagem de Definição de Dados (*Data Definition Language* – DDL) compatível com o núcleo comum aos padrões SQL 92/99/2003 .

1.3 Contribuição

As contribuições esperadas nesse trabalho são uma metodologia MDA (Gonçalves, I. S. et al.,2012) que tem como objetivo automatizar as etapas do processo de modelagem de banco de dados, dando origem então a outras contribuições subsequentes

como o metamodelo, observando os conceitos comuns existentes entre as notações de modelagem conceitual relacional, conferindo à metodologia a flexibilidade pretendida em relação a modelagem conceitual.

Outra contribuição importante decorrente da utilização da MDA são as regras de transformação de modelo para texto criadas utilizando a linguagem de transformações MOFM2T (OMG, 2008), disponibilizada pela Object Management Group (OMG), que após serem aplicadas sobre um modelo aderente ao metamodelo definido, resultarão na geração do código DDL compatível com o núcleo comum aos padrões SQL 92/99/2003, podendo ser utilizado para implementação da base de dados nos SGBD que sejam aderentes ao padrão.

Será criada uma ferramenta, que será um conjunto de plugins para o ambiente de desenvolvimento Eclipse, onde o metamodelo será construído utilizando o Ecore, parte integrante do EMF, e as regras de transformação implementadas utilizando o Acceleo, gerador de código que utiliza a linguagem MOFM2T.

1.4 Organização do Trabalho

Esse trabalho está estruturado da seguinte forma: o capítulo dois descreve alguns conceitos básicos para entender a proposta ; o capítulo três mostra a abordagem MDA proposta; o capítulo quatro apresenta exemplos utilizando três notações de modelagem conceitual distintas utilizando a abordagem proposta; o capítulo cinco apresenta alguns trabalhos relacionados e o capítulo seis conclui o trabalho.

2 Conceitos

Este capítulo descreve os conceitos básicos que serão utilizados no desenvolvimento da metodologia proposta. Na seção 2.1 será apresentado o conceito de modelagem de dados, na seção 2.2 o conceito de Modelo Conceitual, na 2.3 o conceito de Entidade-Relacionamento, na seção 2.4 será falado sobre a Notação de modelagem Crow's Foot, na 2.5 será falado sobre a notação IDEF1X, na seção 2.6 será falado sobre outras notações. Na sessão 2.7 será falado sobre o Modelo Lógico,. Na seção 2.8 será falado sobre o conceito de Modelo Físico, na seção 2.9 será falado sobre os conceitos da Arquitetura Orientada a Modelos (*Model-Driven Architecture*-MDA) e na 2.10 sobre as Transformações de Modelos.

2.1 Modelagem de Dados

A modelagem de dados consiste em determinar a estrutura do banco de dados, definindo suas características, a fim de facilitar seu entendimento, bem como seu projeto e implementação. O ambiente observado pode ser representado por meio da construção de um modelo, com elementos textuais ou gráficos, os quais serão definidos de acordo com a notação e técnicas empregadas durante a construção do mesmo (Heuser, 2009).

2.2 Modelo Conceitual

O modelo conceitual é constituído, geralmente, por elementos diagramáticos os quais são especificados por notações de modelagem conceitual. Esses elementos são utilizados para representar as informações, que estarão presentes no banco de dados modelado, em um nível de abstração mais alto, ou seja, mais próximo à percepção do usuário, não levando em consideração detalhes tecnológicos, como por exemplo, qual será o SGBD utilizado para a implementação do banco.

2.3 Notação Entidade-Relacionamento

A notação de Peter Chen para modelagem de dados, o Modelo Entidade- Relacionamento (MER), foi criada com base em outras três notações de modelagem: o Modelo de Redes (BACHMAN, 1973), o Modelo de Conjunto de Entidades, que é baseado na Teoria de Conjuntos (Halmos, 1960), e o Modelo Relacional (Codd, 1970), que é ba-

seado na Teoria da Relação, buscando ser um meio em comum entre elas, porém reunindo vantagens existentes nas três (CHEN, 1976).

A técnica diagramática utilizada por Chen para descrever a estrutura do banco de dados é chamada de Diagrama Entidade-Relacionamento (DER) , que incorpora as informações do mundo real, ou seja, do ambiente que se deseja modelar e, através de uma técnica diagramática, representa esse ambiente, transformando o diagrama em uma ferramenta importante para a criação do banco de dados. O modelo possui em sua essência abstrações das informações acerca do ambiente que será modelado. Isso faz com que o mesmo represente esse ambiente de forma mais natural, facilitando o entendimento.

Chen (1976) identifica quatro pontos de vista lógicos de dados e os divide em quatro níveis:

1. Abstração de dados – neste nível acontece a abstração dos elementos do ambiente que se deseja modelar, transformando os mesmos em objetos conceituais mentais. Essa abstração é necessária para definir quais informações são mesmo necessárias e devem estar no banco de dados, parte importante do processo, pois armazenar todas as informações é inviável. Faz-se a distinção dos elementos do ambiente que será modelado transformando-os em entidades e relacionamentos, segundo suas funções, ou seja, seus papéis no mundo real.

2. Organização da estrutura de dados – neste nível devem-se organizar esses dados como entidades e relacionamentos, a fim de transformá-los em uma estrutura. Aqui os objetos conceituais existentes em nossa mente, ganham representações diagramáticas, considerando assim que esses objetos possuem representações conceituais diretas.

3. Diagrama Entidade-Relacionamento - nesse nível, são apresentados alguns assuntos pertinentes à representação e manipulação de dados feita por meio da interação entre os objetos conceituais. O diagrama Entidade-Relacionamento é uma técnica diagramática para a representação dos dados por meio de entidades, relacionamentos e atributos.

4. Análise de outros modelos de dados e suas derivações do modelo entidade-relacionamento - nesse nível, Chen explica um pouco como funcionam os modelos que deram origem à sua notação e faz uma comparação mostrando as vantagens que sua notação vem a apresentar em relação a esses modelos, ressaltando, por exemplo, que o uso do MER ajuda a resolver problemas de ambiguidade de informação.

Uma entidade é algo que existe ou pode existir no ambiente modelado e que possui características próprias de distinção das demais entidades, ou seja, um elemento com uma identificação distinta e significado próprio. O papel de uma entidade em uma relação é representar a função da entidade no relacionamento entre elas. Um exemplo de relação a ser citado é o relacionamento Casamento onde duas entidades do conjunto Pessoa têm seus papéis definidos por Marido e Esposa, como pode ser visto na figura 2.

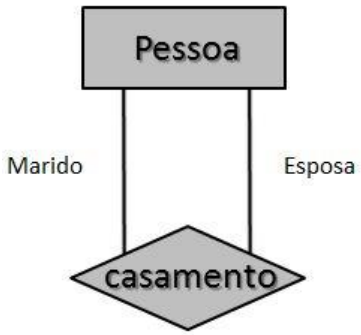

Figura 2 - Relacionamento Casamento (CHEN, 1976)

Um relacionamento é uma conexão, uma associação entre as ocorrências das entidades, podendo ocorrer entre ocorrências de entidades distintas ou entre ocorrências de uma mesma entidade o que caracteriza um auto relacionamento. As informações, como características e propriedades sobre as entidades e seus respectivos relacionamentos, são chamadas de atributos. Os atributos de uma entidade são definidos pelo conjunto de entidades que ela faz parte (CHEN, 1976). O conjunto de entidades é a coletânea de todas as entidades de um tipo entidade em particular. Se o conjunto de entidades Cliente tem como atributo Código, Nome, CPF e Telefone, todas as entidades que pertencem a esse conjunto terão esses mesmos atributos.

As informações sobre entidades e também sobre os relacionamentos são conseguidas através da observação do modelo que se deseja armazenar no banco de dados. As informações são representadas em valores que serão ligados a conjuntos

de valores formando um Atributo, que pode ser formado por um ou mais conjuntos de valores (CHEN, 1976). Esses atributos terão a finalidade de tornar a informação armazenada livre de ambiguidade, já que um mesmo valor pode assumir significados diferentes em relação ao seu contexto. Por exemplo, um número para representação de peso tem significado diferente para hora ou medida de distância.

Quando um atributo é formado por mais de um conjunto de valores pode-se dizer que é um atributo multivalorado (CHEN, 1976). Para exemplificar, temos um atributo Nome, que pode ser formado por primeiro nome e sobrenome. Porém, é aconselhável não utilizar atributos multivalorados quando se está modelando, pois em muitos casos esses atributos podem possuir características específicas, o que poderia causar erro no momento da modelagem, sendo correto torná-lo uma entidade ao invés de um atributo.

Atributo de relacionamentos nasce a partir de informações que resultam da interação entre duas entidades, não dizendo respeito à informação sobre uma única entidade do relacionamento, mas como citado, sobre a informação que a relação entre elas produz. Os atributos de relacionamentos também ajudam a compreender melhor a dependência de dados entre entidades.

Chen (1976) introduziu também o conceito que se conhece como herança entre conjuntos de entidades através do seu conceito de subconjunto de entidades, onde cada entidade que participa do conjunto de entidades Pessoa_Masculina, também pertencerá ao conjunto Pessoa recebendo suas características, como pode ser visto na figura 3.

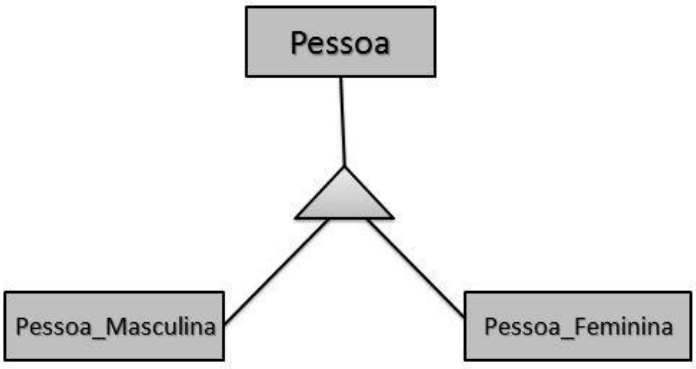

Figura 3 - Subconjunto de Pessoa (CHEN, 1976)

Percebe-se ainda que, a partir do conceito de subconjuntos, existe o princípio da ideia de Especialização. A partir de um conjunto de entidades Funcionário podemos ter os subconjuntos Padeiro e Confeiteiro, cada um, como citado anteriormente, possuindo suas próprias características e obtendo também as características do conjunto de entidades que se encontram nível acima, no caso, de Funcionário, como pode

ser visto na figura 4. O MER permite a observação clara da dependência entre os dados por causa da organização desses dados como entidades e relacionamentos.

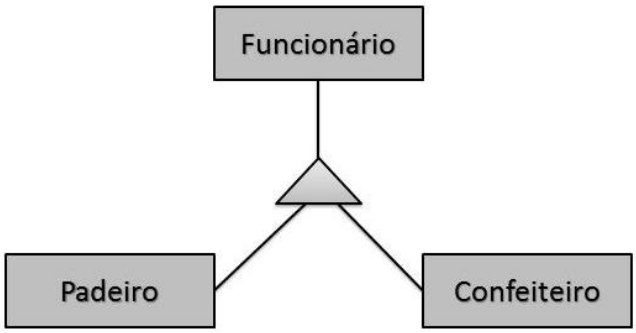

Figura 4 - Especialização de Funcionário (Fonte: Os Autores)

Ao se definir uma entidade em valores, existe a necessidade de estabelecer uma identificação da mesma em seu conjunto, a fim de promover a distinção da mesma das demais entidades de seu conjunto. Isso pode ser feito definindo um ou mais atributos como chave primária (do inglês *Primary Key*, PK) da entidade (CHEN, 1976). Esse valor deve ser único na base de dados e devem ser escolhidos atributos que não sofram alteração com o tempo. Na ausência de um atributo que complete esse perfil, pode-se definir um atributo artificial para a tarefa.

Considerando uma ocorrência da entidade Empregado e seus atributos, para distinguir essa ocorrência de outras, deve-se definir um atributo como PK da ocorrência. Na figura 5, o atributo escolhido para essa tarefa foi o número do empregado, visto que, qualquer um dos outros atributos presentes nessa entidade não serviram como PK, pois são atributos que podem sofrer alterações com o tempo e também podem se repetir.

De acordo com Chen (1976), também é possível identificar uma relação entre entidades através das chaves primárias das mesmas, porém há casos que para identificar uma entidade não é possível utilizar seus próprios atributos, sendo necessário recorrer à relação da qual ela faz parte para identificá-la, pois, teoricamente, todos os relacionamentos servem para identificar as entidades. Também existem chaves primárias formadas pela junção das chaves primárias de outras entidades para a formação de uma nova entidade.

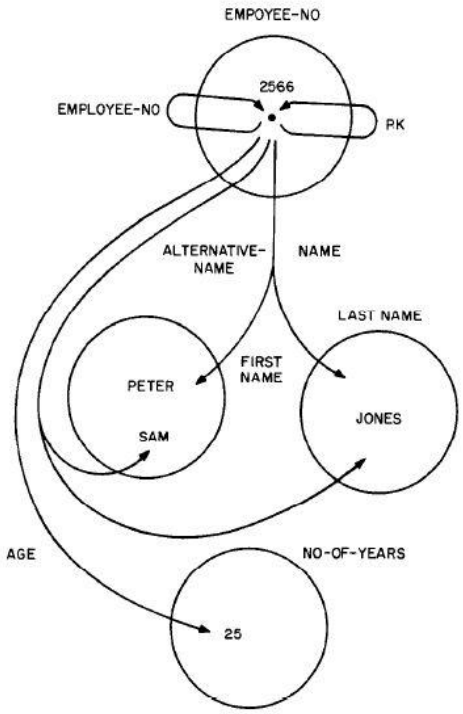

Figura 5 - Representando Entidades por Valores (número-empregado). (CHEN, 1976)

Existem duas formas de classificar uma entidade. Se uma das entidades participantes depende do relacionamento em questão para ser identificada, esse relacionamento é classificado como relação de entidades fraca. Caso a relação não seja utilizada para identificar entidades ela é classificada como relação de entidades regular (CHEN, 1976).

Existem ainda, duas formas de relacionamentos. Se todas as entidades participantes de um relacionamento são identificadas pelos seus próprios atributos, podemos classificá-lo como relação de relacionamento regular. Se alguma entidade no relacionamento é identificada por outro relacionamento, esse relacionamento é classificado como relação de relacionamento fraco (CHEN, 1976).

O MER é independente de um Sistema Gerenciador de Banco de Dados (SGBD) específico (Heuser, 2009), pois no mesmo devem ser considerados os dados independentemente do processamento, ou seja, do software que irá transformá-los. Por isso, durante a fase da modelagem conceitual da base de dados, não se tem como prioridade tecnologias e suas particularidades. Nessa fase, o foco deve ser observação do ambiente a ser representado na base de dados (CHEN, 1976).

2.3.1 O Diagrama Entidade-Relacionamento

O Diagrama de Entidade-Relacionamento (DER) é uma ferramenta de representação gráfica de entidades e relacionamentos. No DER, cada entidade é representada por um retângulo como pode ser visto na figura 6:

Figura 6 - Representação de uma entidade no Diagrama Entidade-Relacionamento de Peter Chen (Fonte: Os Autores)

Em alguns casos, o conjunto de valores e atributos das entidades pode ser definido no DER. Caso seja feito, os atributos serão representados por uma haste contendo um círculo não preenchido na ponta. Definindo a chave primária de cada entidade entre seus atributos sua representação será feita também por uma haste com círculo, porém esse totalmente preenchido, conforme visto na figura 7.

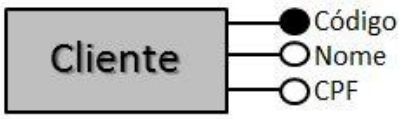

Figura 7 - Representação de uma entidade e seus atributos no Diagrama Entidade-Relacionamento de Peter Chen (Fonte: Os Autores)

Os relacionamentos são representados por um losango contendo o nome da relação como pode ser visto na figura 8:

Figura 8 - Representação de uma relação no Diagrama Entidade-Relacionamento de Peter Chen (Fonte: Os Autores)

A representação dos relacionamentos possui, além do item da figura 7, linhas ligando a entidade ao relacionamento que, por sua vez, é ligado à outra entidade. Nas extremidades das linhas, por parte da entidade, são indicadas as cardinalidades que a relação possui, como pode ser visto na figura 9:

Figura 9 - Representação de um relacionamento no Diagrama Entidade-Relacionamento de Peter Chen (Fonte: Os Autores)

A representação do auto relacionamento é feita por meio de uma linha conectando a entidade ao relacionamento e o mesmo conectado novamente a mesma entidade, como pode ser visto na figura 10:

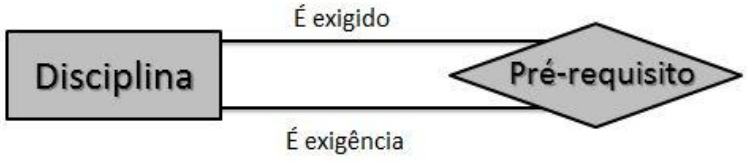

Figura 10 - Representação de um auto relacionamento no Diagrama Entidade-Relacionamento de Peter Chen (Fonte: Os Autores)

A especialização é representada no DER por meio de linha conectando a entidade genérica à um triangulo e o mesmo é conectado às entidades que são especializações da entidade genérica, como pode ser visto na figura 11.

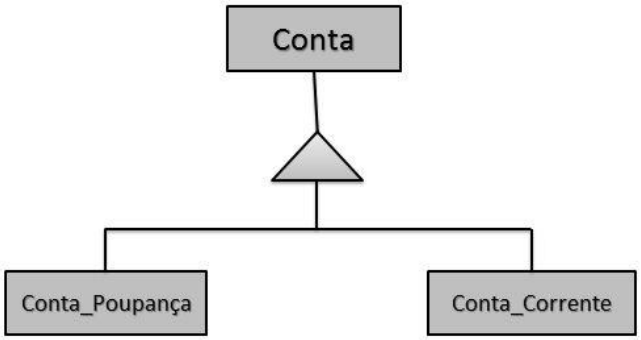

Figura 11- Representação de uma especialização no Diagrama Entidade-Relacionamento de Peter Chen (Fonte: Os Autores)

Segundo Chen (1976), algumas características importantes do DER podem ser identificadas:

- Um relacionamento pode ser definido por mais de uma entidade.
- Um relacionamento pode ser definido por apenas uma entidade.
- Duas entidades podem ter dois relacionamentos diferentes ou mais.

- O diagrama pode distinguir a cardinalidade dos relacionamentos.
- O diagrama pode expressar a dependência existencial entre entidades.

Um relacionamento pode conter mais de dois conjuntos de entidades como é o caso representado na figura pelo relacionamento Forn-Proj-Peç, no qual se relacionam as entidades dos conjuntos Projeto, Peça e Fornecedor assim também como pode um relacionamento acontecer com apenas uma entidade, caso representado na figura 12 pelo relacionamento Componente.

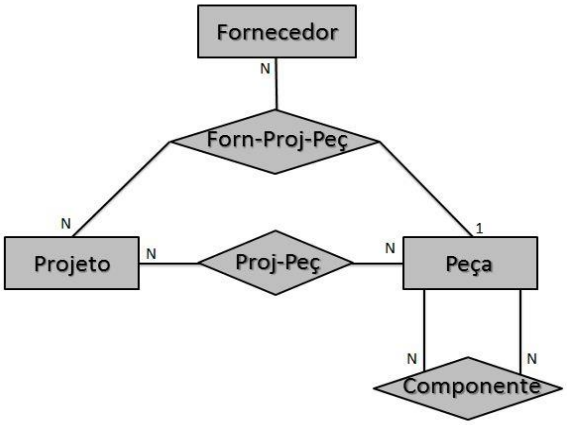

Figura 12 - Modelo Entidade-Relacionamento para análise da informação de uma firma de manufatura. (CHEN, 1976)

Os relacionamentos também podem demonstrar as dependências entre as ocorrências de entidades. Na figura 13, existe o relacionamento entre as entidades Empregado e Dependente, a qual é representada por um retângulo especial, representando uma relação de entidade fraca. A entidade Dependente existe apenas por causa da existência do Empregado, pois se no caso, se o Funcionário deixasse de fazer parte da empresa, os dados de seu dependente já não importariam mais para a empresa.

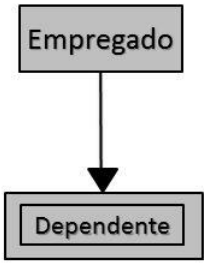

Figura 13 - Modelo Entidade-Relacionamento para análise da informação de uma firma de manufatura. (CHEN, 1976)

A limitação das ocorrências das entidades em um ou mais relacionamentos é denominada cardinalidade. As cardinalidades existentes entre os relacionamentos no MER são as seguintes:

- 1:1: Consideremos o relacionamento entre Gerente e Departamento, no qual, uma ocorrência da entidade Gerente pode estar associada a apenas uma ocorrência da entidade Departamento, como pode ser visto na figura 14. Este relacionamento é denominado um-para-um (1:1) e ocorre quando uma ocorrência de uma entidade A pode estar associada a apenas uma ocorrência da entidade B e uma ocorrência de uma entidade B pode estar associada a apenas uma ocorrência da entidade A.

Figura 14 - Relacionamento um-para-um. (Fonte: Os Autores)

- 1:N: Consideremos o relacionamento entre as entidades Departamento e Empregado, na qual, um Departamento pode ter vários Empregados e um Empregado pode estar em apenas um Departamento, como pode ser visto na figura 15. Esse relacionamento é denominado um-para-muitos (1:N) e ocorre quando uma ocorrência de uma entidade A pode estar relacionada a várias ocorrências de uma entidade B, enquanto a entidade B relaciona-se apenas com uma ocorrência da entidade A.

Figura 15 - Relacionamento Departamento – Empregado. (CHEN, 1976)

- N:N: Consideremos o relacionamento entre Empregado e Projeto, no qual um Empregado pode estar relacionado a várias ocorrências da entidade Projeto e a entidade Projeto pode estar relacionada a várias ocorrências da entidade Empregado, como pode ser visto na figura 16. Este relacionamento é denomi-

nado muitos-para-muitos (N:N) e ocorre quando uma ocorrência de uma entidade A pode estar relacionada a várias ocorrências da entidade B e uma ocorrência da uma entidade B pode estar relacionada a várias ocorrências da entidade A.

Figura 16 - Relacionamento Empregado - Projeto. (CHEN, 1976)

Segundo Chen (CHEN, 1976), seguem algumas etapas para ajudar a definir uma base de dados utilizando o Diagrama de Entidade-Relacionamento:

- Identificação das entidades e relacionamentos importantes no ambiente a ser modelado, ou seja, os elementos que possuem características próprias.
- Identificação das informações semânticas sobre os itens citados acima, como a cardinalidade, por exemplo.
- Definir os valores dos atributos.
- Organizar os dados como entidades e relacionamentos, ciando a relação entre eles, bem como decidir as chaves primárias das entidades.

2.3.2 Integridade de Dados

A integridade de dados diz respeito à fase de modelagem conhecida como modelo lógico ou projeto lógico, onde ocorre a definição das entidades e relacionamentos através de dados que deverão ser armazenadas para a construção da base de dados, organizando as definições de dados levando em conta suas descrições e restrições de integridade, normas que devem ser obedecidas por todos estados válidos na base de dados (Heuser, 2009).

Segundo Chen (1976), para permitir maior confiabilidade no armazenamento e manipulação dos dados, são definida restrições na entrada de dados em relação a seu tipo e extensão de informação. Algumas das possíveis são:

- Limitações em relação a valores permitidos para conjunto de valores: podemos definir quais tipos de valores serão aceitos para determinado conjunto (caractere, Inteiro, etc.), bem como sua variação em relação à quantidade de informação ou espaço disponível para tal (permitir apenas quatro dígitos para determinado número ou uma variação entre uma determinada numeração, de 0 a 50, por exemplo);

- Limitações de valores permitidos para determinados atributos: em alguns casos, de acordo com as regras do negócio que estamos representando, nem todos os valores permitidos são aceitos para determinados atributos. Um exemplo seria a definição de uma quantidade mínima para venda de produtos de uma distribuidora.

Segundo Chen (1976), com o uso do DER, a recuperação dos dados se torna mais fácil, pois o mesmo permite uma visualização clara e objetiva dos elementos. Levando em consideração a dificuldade em manter a integridade dos dados no momento em que se fala em realizar operações de exclusão, alteração ou inserção nos deparamos também com falta de regras para manter essa integridade. O MER presta-se a esta tarefa, pois o mesmo tem por fundamento facilitar a compreensão da informação a ser representada e manipulada.

Por padronização, cada representação visual, ou seja, cada elemento gráfico, tem um conceito único e definido. Isso faz com que ele seja facilmente compreendido por profissionais da área. O modelo é independente de um SGBD específico (Chen, 1976). Essa independência torna mais simples migrar de um modelo lógico para outro, caso seja necessário, assim como sua manutenção.

2.4 Notação de modelagem Crow's Foot

Dentre as notações de modelagem conceitual utilizadas atualmente, encontra-se a notação de *Crow's Foot* que é frequentemente empregada no processo de modelagem de dados. Essa notação também possui recursos gráficos que são usados para representar os elementos que irão compor o banco de dados modelado. A notação é chamada de *Crow's Foot* (Patas de Galinha) porque um dos símbolos utilizados na representação das cardinalidades possui aparência semelhante às patas de tal ave.

Nesta notação, as entidades independentes são representadas por quadrados contendo o seu nome e seus atributos, os últimos podem também conter indicações que os definam como chave primária, como exemplificado na figura 17, ou estrangeira, caso a entidade em questão possua. A indicação de que uma entidade é dependente é feita adicionando traços inclinados nas extremidades do quadrado correspondente à entidade dependente, como pode ser visto na figura 18 (Halpin, 1999).

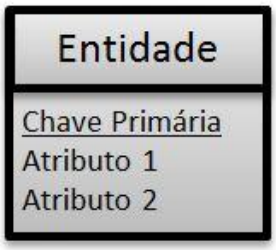

Figura 17 - Representação de entidade Notação de Crow's Foot (Fonte: Os Autores)

Figura 18 - Representação de entidade dependente Notação de Crow's Foot (Fonte: Os Autores)

As associações entre as entidades, ou seja, as relações entre elas são representadas por linhas que fazem a conexão das entidades que pertencem ao relacionamento em questão, os relacionamentos binários, ou seja, entre duas entidades são representados conforme a figura 19, os relacionamentos entre três entidades são representadas na figura 20 e a representação de auto relacionamentos, ou relacionamentos unários, como também é chamado, são representados na figura 21 (Halpin, 1999) .

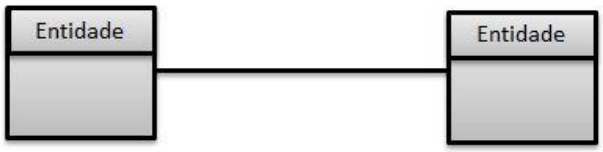

Figura 19 - Relacionamento binário Notação de Crow's Foot (Fonte: Os Autores)

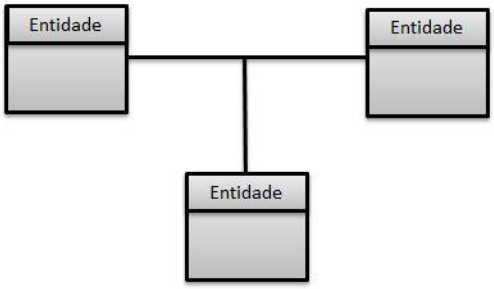

Figura 20 - Relacionamento ternário Notação de Crow's Foot (Fonte: Os Autores)

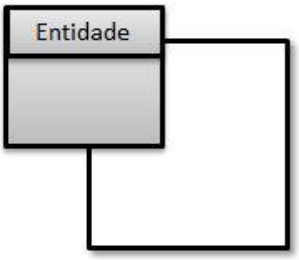

Figura 21 - Relacionamento unário Notação de Crow's Foot (Fonte: Os Autores)

As cardinalidades são representadas por formas distintas localizadas nas extremidades das linhas. A representação de cardinalidade na qual a entidade pode não participar da relação ou participar várias vezes é demonstrada por um conjunto de três linhas, entre as quais uma é inclinada para baixo, uma é horizontal e a outra é inclinada para cima, juntamente com um circulo sem preenchimento, como pode ser visto na figura 22 (Hay, 1999).

Figura 22 - Cardinalidade 0:N Notação de Crow's Foot (Fonte: Os Autores)

A representação de cardinalidade na qual a entidade deve participar ao menos uma vez da relação e que a mesma pode participar várias vezes é demostrada por um conjunto de três linhas, entre as quais uma é inclinada para baixo, uma é horizontal com um traço vertical e a outra é inclinada para cima, como pode ser visto na figura 23 (Hay, 1999)..

Figura 23 - Cardinalidade 1:N Notação de Crow's Foot (Fonte: Os Autores)

A representação de cardinalidade na qual a entidade só deve participar uma vez da relação é demostrada por uma linha horizontal com dois traços verticais, como pode ser visto na figura 24 (Hay, 1999).

Figura 24 - Cardinalidade 1:1 Notação de Crow's Foot (Fonte: Os Autores)

A representação de cardinalidade na qual a entidade pode não participar da relação ou, se houver participação, essa só deve ocorrer uma única vez é demostrada por uma linha horizontal com um traço na vertical juntamente com um círculo sem preenchimento, como pode ser visto na figura 25 (Hay, 1999).

Figura 25 - Cardinalidade 0:1 Notação de Crow's Foot (Fonte: Os Autores)

2.5 Notação IDEF1X

O IDEF1X é uma linguagem para modelagem de dados que se baseia no modelo Entidade-Relacionamento e foi tido como padrão de modelagem norte-americano pelo *National Institute of Standards and Technology* (NIST). Possui recursos textuais e gráficos que são utilizados para representar os elementos presentes no ambiente que está sendo modelado.

As entidades nessa notação são representadas por um retângulo que contém uma linha divisória na horizontal, sendo inscritos acima dessa linha os atributos que serão eleitos chave primária da entidade e abaixo dela os demais atributos que poderão ser utilizados para descrever as chaves estrangeiras, se necessário. As entidades podem ser dependentes ou independentes. No primeiro caso, elas possuem seus cantos com formato arredondado e sua chave primária é formada pela chave de outra entidade, como na figura 26; exemplos do presente caso são as situações em que ocorre a especialização de uma entidade genérica em entidades específicas. No segundo caso, as entidades possuem seus cantos no formato normal do retângulo como na figura 27 (Bruce, 1992).

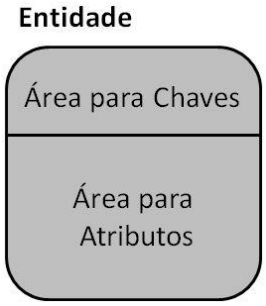

Figura 26 - Representação Entidade Dependente IDEFIX (Fonte: Os Autores)

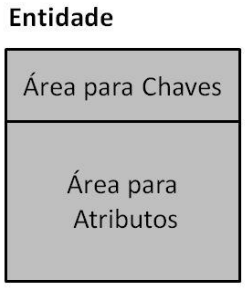

Figura 27 - Representação Entidade Independente IDEF1X (Fonte: Os Autores)

Os relacionamentos entre as entidades são representados por uma linha ligando uma entidade à outra participante da relação, com uma frase verbal indicando o nome da relação. Quando o relacionamento é representado por uma linha contínua que possui em uma de suas extremidades um círculo preenchido, significa que a chave primária da entidade A forma parte da chave primária da entidade B. Esse tipo de relacionamento recebe o nome de Relacionamento Identificador, pois a entidade B

que recebe a chave primária da entidade A depende dela para ser identificada. Um exemplo pode ser visto na figura 28 (Bruce, 1992).

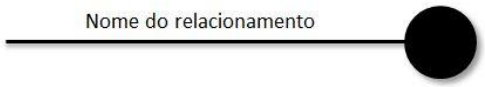

Figura 28 - Representação Relacionamento Identificador IDEF1X (Fonte: Os Autores)

Quando o relacionamento é representado por uma linha tracejada que possui em uma de suas extremidades um circulo preenchido, significa que a chave primária da entidade A encontra-se como chave estrangeira, junto aos atributos e assim não faz parte da formação da chave primária de B, o que torna a mesma independente da entidade A. Esse tipo de relacionamento é chamado de Relacionamento Não-Identificador, pois a entidade B não depende da entidade A para ser identificada, um exemplo pode ser visto na figura 29 (Bruce, 1992).

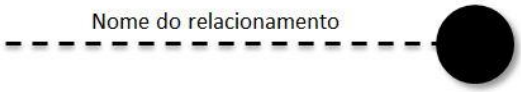

Figura 29 - Representação Relacionamento Não-Identificador IDEF1X (Fonte: Os Autores)

O Relacionamento Não-Identificador pode ser opcional ou mandatório. No primeiro caso, cada instância da entidade B relaciona-se com 0 ou 1 instância da entidade A; nesse caso a linha tracejada recebe um losango na extremidade oposta ao circulo preenchido. No segundo caso, cada instância da entidade B relaciona-se com apenas 1 instância da entidade A, nesse caso não é necessário acrescentar o losango. Um exemplo pode ser visto na figura 30 (Bruce, 1992).

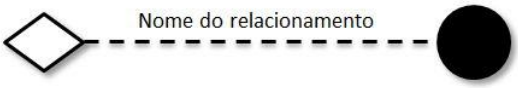

Figura 30 - Representação Relacionamento Não-Identificador Opcional IDEF1X (Fonte: Os Autores)

As cardinalidades são indicadas por símbolos colocados em uma das extremidades junto ao círculo preenchido. Quando nenhum símbolo é colocado , significa que a cardinalidade pode ser 0, 1 ou várias, como pode ser visto na figura 31. Quando é

colocado o símbolo P, significa que a cardinalidade pode ser 1 ou várias, como pode ser visto na figura 32. Quando o símbolo colocado é o Z, significa que a cardinalidade é 0 ou 1, como pode ser visto na figura 33. Se o símbolo for o n, sendo n um número inteiro, significa que a cardinalidade será n, ou seja, a entidade poderá participar da relação n vezes, como pode ser visto na figura 34 (Bruce, 1992).

Figura 31 - Representação Cardinalidade 0, 1 ou várias IDEF1X (Fonte: Os Autores)

Figura 32 - Representação Cardinalidade 1 IDEF1X (Fonte: Os Autores)

Figura 33 - Representação Cardinalidade 0 ou 1 IDEF1X (Fonte: Os Autores)

Figura 34 - Representação Cardinalidade n vezes IDEF1X (Fonte: Os Autores)

Quando em um modelo alguma entidade compartilha um ou mais atributos com outras entidades, dá-se o nome de relacionamento categorizado (generalização/especialização). A entidade A é chamada de categoria discriminadora, ou seja, é ela que compartilha os atributos e as outras entidades (B, C,…, N) são chamadas apenas de categorias. Quando todas as categorias de uma entidade A estão sendo ex-

postas no modelo, utiliza-se um discriminador composto por dois traços horizontais com um círculo não preenchido sobre eles, como pode ser visto na figura 35. Um exemplo desse tipo de relação acontece quando se tem a entidade empregado e como categorias apenas homem e mulher (Bruce, 1992).

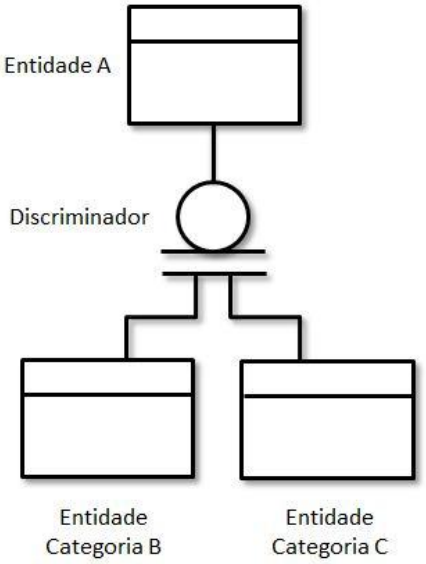

Figura 35 - Representação categorização com todas as categorias expostas (Fonte: Os Autores)

Quando uma entidade A não possui todas as suas categorias expostas no modelo, utiliza-se um discriminador composto por um traço na horizontal com um círculo não preenchido sobre ele, como pode ser visto na figura 36. Um exemplo desse tipo de relação está representado na figura 36, quando se tem a entidade Empresa e como categorias duas entidades que representam apenas duas filiais. Existem mais possibilidades de categorias, considerando que a empresa em questão tenha mais de duas filiais, representar todas elas no modelo seria exaustivo e desnecessário (Bruce, 1992) .

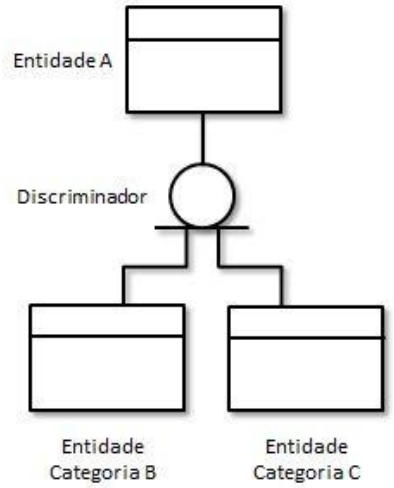

Figura 36 - Representação categorização sem todas as categorias expostas (Fonte: Os Autores)

2.6 Outras Notações

Uma linguagem de modelagem considerada padrão é a UML, muito utilizada na modelagem de software. Esta linguagem possui elementos bem definidos capazes de descrever as partes concretas e abstratas que envolvem a modelagem de um software, elementos esses que são chamados de diagramas que podem descrever tanto a estrutura quanto o comportamento do software. Por esses elementos seguirem certo padrão, torna-se mais fácil o entendimento dos desenvolvedores acerca das partes do projeto (Guedes, 2008). Mesmo que a UML não tenha sido projetada para ser utilizada na descrição de modelos conceituais de bancos de dados, ela também é amplamente utilizada para este fim.

A notação Min-Max refere-se à maneira de representar a cardinalidade utilizada em combinação com ER para determinar o mínimo e máximo de vezes que a entidade participa da relação. Nesta notação, as cardinalidades são representadas por pares de números inteiros perto da representação da entidade que o elemento pertence. A cardinalidade que está relacionada com a limitação mínima, no lado esquerdo da notação e a cardinalidade que está relacionada com a limitação máxima no lado direito, tais como: (1, N), (1, 1) (Hay, 1999).

2.7 Modelo Lógico

O Projeto Lógico de dados é o processo de transformação do Modelo Conceitual para o Modelo Lógico, que é o modelo que representa a estrutura de dados que é vista pelo usuário do SGBD, estando o modelo ligado às particularidades do mesmo. Essa

estrutura em SGBD relacionais é representada por tabelas lógicas, representadas no Modelo Conceitual por um conjunto de entidades (Chen, 1976), relacionamentos e seus atributos. (Heuser, 2009).

Normalmente, o nome do conjunto de entidades dá o nome à tabela lógica que a representa e os atributos mapeados do mesmo, no modelo conceitual, são representados na tabela lógica por colunas com seus respectivos nomes. Cada elemento de um conjunto de entidades que se deseja armazenar no banco de dados dará origem a uma linha, ou registro, na tabela que representa seu conjunto.

As tabelas lógicas não possuem suas linhas dispostas de forma ordenada. Para identificar um registro da tabela ou representar as relações entre elas é utilizado o conceito de chaves que podem ser estabelecidas das seguintes formas:

2.7.1 Chave Candidata

Chave Candidata, ou Chave Alternativa como é chamada em algumas bibliografias, é uma ou o conjunto de várias colunas de uma tabela que poderiam ser escolhidas para identificar e diferenciar uma linha da tabela das demais por possuir unicidade de valores. Além de identificar determinado registro da tabela, a coluna eleita poderá ser referenciada por chaves estrangeiras de tabelas que estejam se relacionando, sendo necessário discernir qual coluna possui valores mais adequados para tal. Caso não sejam eleitas como chave primária, as chaves candidatas deverão possuir seus valores preenchidos obrigatoriamente (Heuser, 2009).

2.7.2 Chave Primária

Segundo Heuser (2009), é chamada de chave primária a chave candidata que foi escolhida para identificar uma linha da tabela. Quando uma chave primária é formada pelo valor de mais de uma coluna ela é chamada de Chave primária Composta. Essa só ocorre quando apenas o valor de uma coluna não é suficiente para identificar uma linha das demais, devido à repetição que esse mesmo valor pode ter em linhas diferentes. O valor ou o conjunto de valores estabelecidos como chave primária não deve se repetir como chave primária de outras linhas da tabela. Na figura 37, é mostrado um exemplo de representação do modelo lógico de uma entidade Cliente:

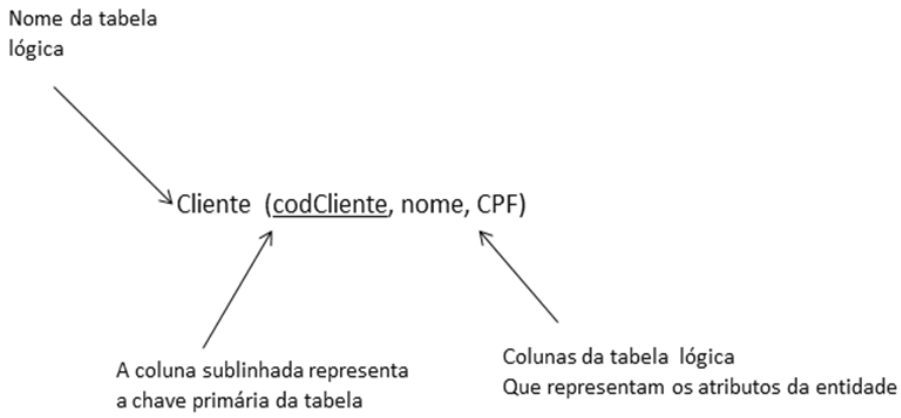

Figura 37 - Representação de uma tabela lógica (Fonte: Os Autores)

Uma chave primária, caso seja composta, deve ser formada somente por valores de colunas que sejam realmente necessárias para a identificação da linha, pois sempre que as colunas estabelecidas estiverem associadas será formada a chave primária. (Heuser, 2009)

2.7.3 Chave Estrangeira

Os relacionamentos que um conjunto de entidades participa no modelo conceitual também influenciam no projeto lógico, pois o vínculo entre entidades criado por determinadas relações deverão ser representadas nas tabelas lógicas. A implementação dos relacionamentos entre as tabelas é feita através da definição de uma ou mais colunas que contenham o(s) valor(es) da chave primária da tabela que se relaciona. A esse conjunto estabelecido se da o nome de Chave Estrangeira (Heuser, 2009).

O surgimento da necessidade da definição de uma chave estrangeira em determinada tabela acontece de acordo com a cardinalidade presente no relacionamento. Como exemplo, se é definido no modelo conceitual como regra de negócio que um cliente pode ter vários telefones e que cada telefone pertença a um único cliente, uma relação de 1:N, pode-se fazer o projeto lógico conforme representado na figura 38. A chave estrangeira está sendo representada na tabela Telefone por uma coluna que contem o nome da chave primária da tabela que faz referência, Cliente. Ela indica que uma linha da tabela Telefone está relacionada a uma e somente uma linha da tabela Cliente.

No exemplo do modelo lógico relacionado à figura 38, pode-se notar algumas particularidades. A chave primária da tabela Aluga é formada pelas chaves estrangei-

ras que fazem referência às instâncias de entidades envolvidas na relação; isso, para que possam ser especificadas quais são as instâncias de cada conjunto envolvido, contendo, além dessas identificações, os atributos; no caso das tabelas, as colunas, que são necessárias para armazenar os dados que provêm dessa relação.

Figura 38 – Exemplo relacionamento 1:N (Fonte: Os Autores)

Cliente (<u>codCliente</u>, nome,CPF)
Telefone (<u>codTelefone</u>, DDD, numero, codCliente*)

No caso de existir um relacionamento onde um elemento de um conjunto de entidades possa ter relação com vários elementos de outro conjunto ao mesmo tempo, uma relação de N:N, será necessário criar uma tabela associativa que represente essa relação, identificando quais são os elementos específicos que estão se relacionando armazenando em si as chaves primárias das instâncias e , caso exista, os atributos que possam surgir dessa relação. Como exemplo, podemos citar na figura 39 a relação entre Cliente e Carro no contexto de um concessionária, onde a regra do negócio define que um cliente pode alugar vários carros e cada carro pode ser alugado por vários clientes em momentos diferentes, sendo desejado guardar o histórico de cada locação:

Figura 39 - Exemplo relacionamento N:N (Fonte: Os Autores)

Cliente (<u>codCliente</u>, nome,CPF)
Carro (<u>codCarro</u>, placa, modelo)
Aluga (<u>codCliente*</u>, <u>codCarro*</u>, dataAluguel, dataEntrega, precoContrato)

Apesar da chave que é utilizada para representar as relações entre tabelas receber o nome de chave estrangeira sempre referenciar uma chave primária, nem

sempre ela irá referenciar uma chave primária de outra tabela, como num caso de auto relacionamento de uma entidade . Nesse caso, a tabela no modelo lógico deverá conter uma chave estrangeira que aponte para a chave primária da mesma (Heuser, 2009). Como exemplo na figura 40 , uma situação onde se deseja guardar no banco de dados um cadastro de pessoas e sinalizar o cônjuge de cada pessoa, caso seja casada:

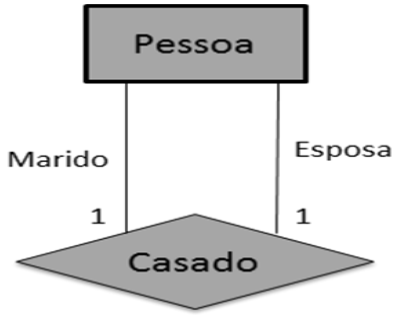

Figura 40 - Exemplo de relacionamento unário (Fonte: Os Autores)

Pessoa (<u>codPessoa</u>, nome, sobrenome, CPF, dataNascimento, codConjuge*)

No relacionamento ternário, a cardinalidade já não se refere ao máximo de instâncias de uma entidade em si, mas sim a pares das mesmas. (Heuser, 2009). Exemplificando, leva-se em conta os conjuntos de entidades A, B e C como participantes de uma relação. A participação de uma instância de A na relação tem sua cardinalidade limitada em 1, enquanto a participação das instâncias de B e C é ilimitada, N. Cada par de instâncias formadas por elementos de B e C estará relacionada a apenas uma instância de A.

Na figura 41, está sendo definido que uma disciplina pode ser lecionada para vários alunos por apenas um professor ou também que, cada aluno estuda várias disciplinas com apenas um professor.

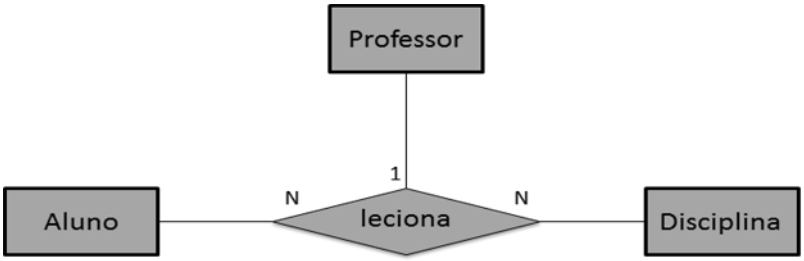

Figura 41 - Exemplo relacionamento ternário (Fonte: Os Autores)

Aluno (matricula, nome, dataNascimento)

Disciplina (codDisciplina, nome, ementa)

Professor (matricula, nome, numRegistro)

Leciona (matAluno*, codDisciplina*, ano, semestre, matProfessor*)

2.7.4 Especialização

Há também circunstâncias em que um conjunto de entidades pode ter um subconjunto, que por sua vez possui suas próprias propriedades e recebendo além delas as propriedades do conjunto que a deriva. A esse conceito se da o nome de Especialização. Esse recurso é utilizado para evitar que em determinados casos se crie vários conjuntos de entidade, e consequentemente de tabelas no modelo lógico, que tenham vários atributos em comum se diferenciando em muitas vezes apenas por uma característica. Como exemplo do caso se tem a variação de Cliente, que pode ser uma pessoa física ou jurídica. Ambas especializações receberão as características do conjunto genérico, Cliente, e terão as suas características particulares. A especialização será demonstrada graficamente por um triângulo isósceles que será ligado aos subconjuntos conforme apresentado na figura 42.

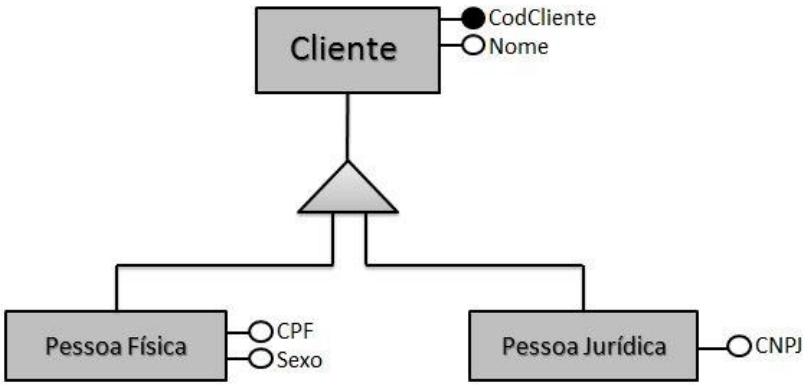

Figura 42 - Exemplo especialização (Fonte: Os Autores)

Para saber de qual instância se trata a especialização, se utiliza de uma chave estrangeira na tabela dos subconjuntos que aponte para a chave primária do conjunto genérico:

Cliente (codCliente, nome)

Pessoa Jurídica (codCliente*,CNPJ)

Pessoa Física (codCliente*, CPF, sexo)

2.7.5 Restrições de Integridade

A integridade de um banco de dados é fundamental para que os dados contidos nele possam ser armazenados, manipulados e recuperados com segurança. A precisão das informações obtidas pelas organizações através dele é crucial para o seu desenvolvimento e na tomada de decisões, portanto devem ser tomadas medidas que garantam essa integridade.

Durante o projeto lógico, com a geração das tabelas serão definidos os conjuntos de valores que serão permitidos atribuir a cada coluna da tabela, sendo esses valores chamados de domínio do campo (Heuser, 2009).

As restrições de integridade são regras que visam a garantir a consistência dos dados no SGBD, sendo essas regras já disponibilizadas pelo SGBD ou podendo também, além dessas, serem definidas outras restrições em relação aos domínios do campo com base nas regras de negócio da organização. (Heuser, 2009)

Das restrições oferecidas pelo próprio SGBD (Heuser, 2009):

- Integridade de domínio: é a especificação dos tipos de valores que os domínios podem assumir para tal coluna como: valor numérico inteiro, data, texto, etc. Alguns SGBD possibilitam a definição de tipos próprios além dos pré-definidos, referentes à própria aplicação que está sendo desenvolvida.
- Integridade de nulidade: é a definição de quais colunas podem ter seus valores vazios ou se devem ter seus valores obrigatoriamente preenchidos. Chaves primárias devem ter seus campos sempre preenchidos.
- Integridade de chaves: garantir a unicidade de valores para as chaves primárias e candidatas dentre todas as linhas da tabela.
- Integridade referencial: define que os valores que serão armazenados como chaves estrangeiras em uma tabela são os mesmos que estão definidos como chaves primárias nas tabelas que fazem referência.

As restrições de integridade definidas além das originais do SGBD, definidas com base nas regras da organização que se deseja representar no banco de dados, são chamados de restrições semânticas (Heuser, 2009).

2.8 Modelo Físico

Nessa etapa da modelagem, ocorre a efetiva implementação do banco de dados, que geralmente já passou pelas etapas de modelagem conceitual e lógica, culminado na modelagem física. A modelagem física envolve a concepção real de um banco de dados de acordo com os requisitos que foram estabelecidos durante a modelagem lógica.A modelagem física lida com a conversão do modelo lógico, ou de negócios, em

um espaço de banco de dados relacional (*Logical Versus Physical Database Modeling*, 2012). Esse modelo incluirá todos os artefatos de banco de dados necessários para criar os elementos presentes no modelo lógico como a estrutura das tabelas, as colunas, chaves primárias e estrangeiras, os relacionamentos entre as tabelas e ainda definições de restrição, tipos de dados, (*Physical Data Model*, 2012) bem como outras particularidades oferecidas por determinados SGBD. Alguns SGBD relacionais possuem diferenças quanto à forma com que os dados são representados e armazenados, alguns possuem elementos que outros não possuem, além de existirem diferenças na forma de representar esses elementos, o que acaba causando incompatibilidade entre muitos SGBD presentes hoje no mercado, pois modelos físicos de um mesmo modelo lógico, podem variar de acordo com o SGBD utilizado.

2.8.1 ANSI SQL

A linguagem utilizada para a definição do modelo físico é a *Structured Query Language* (SQL), que é um padrão *American National Standards Institute* (ANSI) . Essa linguagem permite a gestão dos dados em um SGBD relacional. Embora seja um padrão, existem várias versões diferentes, além de muitos SGBD possuírem uma versão própria de SQL, com comandos particulares, o que causa incompatibilidade entre o SGBD. Segundo Jacobs & Carvalho (2006), as principais versões foram:

- SQL-86, criada em 1986 quando duas organizações: ANSI e *International Standards Organization* (ISO) se juntaram e publicaram um novo padrão para SQL;
- SAA-SQL (*Systems Application Architecture Database Interface*), surgiu quando a IBM em 1987 publicou o seu próprio padrão;
- SQL-89, padrão revisado, também conhecido como SQL1.
- SQL-92, criada em 1992, devido a uma série de incompatibilidades com a entrada de dados do padrão SQL-89, novamente o padrão foi revisado, também conhecido como SQL2;
- SQL-99, criada em 1999, nova linha de SGBD modernos, trazendo novidades como: conceitos objeto-relacional, gerenciamento de integridade, também foi conhecido como SQL3.
- SQL-2003, novos conceitos no gerenciamento de entidades objeto-relacional, suporte a XML (*eXtensible Markup Language*).

As versões SQL-92, SQL-99 e SQL-2003, são as mais utilizadas dentre os SGBD atuais, por isso, as mesmas foram adotadas nesse trabalho e serão comentadas mais a fundo.

2.8.2 SQL 92

Essa versão da linguagem SQL surgiu do aperfeiçoamento das funcionalidades do padrão anterior, além de contar com novas funcionalidades dentre elas:

- Suporte para tipos de dados adicionais (*DATE*, *TIME TIMESTAMP*, *INTERVAL*, cadeia de bits, de comprimento variável de caracteres e sequências de bits, e cadeias de caráter nacional);
- Suporte para operações escalares adicionais, como operações string para concatenação e substring, operações para data e hora e uma forma para descrever expressões condicionais;
- Conjunto adicional de operadores de conjuntos (por exemplo, a união, junção natural, definir diferença e conjunto intersecção);
- Capacidade para definições de domínio no esquema;
- Apoio à recursos de manipulação de esquema (especialmente *DROP* e *ALTER*);
- Definição de um esquema de informações;
- Apoio à execução dinâmica da linguagem SQL;
- Apoio à certas facilidades necessárias para acesso remoto à base de dados (especialmente declarações de conexão de gestão e nomes de esquemas);
- Suporte para tabelas temporárias.

2.8.3 SQL 99

Essa versão da linguagem SQL trouxe novos recursos como novos tipos de dados novos, novos predicados, melhoria semântica, segurança adicional e banco de dados ativo. Dentre os novos tipos de dados estão (Eisenberg; Melton, 1999) :

- *LARGE OBJECT* (LOB) – esse tipo de dado possui as variantes *CHARACTER LARGE OBJECT* (CLOB) e *BINARY LARGE OBJECT* (BLOB). CLOBs comportam-se de forma parecida com strings, mas possuem restrições como não poderem ser usados como chaves primárias, especificados como únicos, utilizados como chaves estrangeiras ou em comparações de igualdade e desi-

gualdade. BLOBs tem comportamento semelhante e não podem ser usados em *GROUP BY* ou *ORDER BY*;

- *BOOLEAN* – esse tipo de dado permite ao SQL gravar no banco os valores verdadeiro, falso e desconhecido;
- *ARRAY* – esse é um tipo composto de dados, que permite armazenar conjuntos de valores diretamente em uma coluna da tabela do banco de dados.

Essa versão recebeu também três novos predicados *LIKE*, *SIMILAR* e *DISTINCT*, permite a criação de *triggers*, entre outros recursos.

2.8.4 SQL 2003

Essa versão possui revisões dos recursos da versão anterior e também novos recursos. Além dos tipos já existentes no SQL-99, foram acrescentados três novos tipos (Lima, 2006):

- *BIGINT* - parecido com os tipos *smallint* e *integer*, porém com maior precisão. Todas as operações do *integer* são suportadas por este tipo. Inteiro de 8 bytes com sinal;
- *MULTISET*: parecido com o tipo *array*, é uma coleção de elementos do mesmo tipo não ordenados e que podem ter valores duplicados;
- XML – tipo criado para armazenamento de documentos XML no banco.

2.8.5 SQL DML e SQL DDL

A linguagem permite criar a estrutura, inserir dados, fazer consultas, modificações e exclusões. A linguagem SQL pode ser dividida em duas partes (SQL Introduction, 2012):

- *Data Manipulation Language* (DML) – contém os comandos de consulta e edição.
    - ***SELECT*** – extrai dados da base de dados, conforme a figura 43;

```
SELECT column_name(s)
FROM table_name
```

Figura 43 - Sintaxe do comando SQL Select (Fonte: Os Autores)

- ***UPDATE*** – edita dados na base de dados, conforme a figura 44;

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Figura 44 - Sintaxe do comando SQL Update (Fonte: Os Autores)

- ***DELETE*** – exclui dados da base de dados, conforme a figura 45 ;

```
DELETE FROM table_name
WHERE some_column=some_value
```

Figura 45 - Sintaxe do comando SQL Delete (Fonte: Os Autores)

- ***INSERT INTO*** – insere novos dados na base de dados, conforme a figura 46.

```
INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)
```

Figura 46 - Sintaxe do comando SQL Insert into (Fonte: Os Autores)

- *Data Definition Language* (DDL) – permite a criação e exclusão de tabelas no banco, a definição das chaves, especificação de links entre as tabelas e a criação de restrições entre as mesmas. Alguns comandos DDL importantes são.

  - ***CREATE DATABASE*** – cria uma nova base de dados, conforme a figura 47.

```
CREATE DATABASE database_name
```

Figura 47 - Sintaxe do comando SQL Create Database (Fonte: Os Autores)

  - ***CREATE TABLE*** – cria uma nova tabela, conforme a figura 48.

```
CREATE TABLE table_name
(
column_name1 data_type,
column_name2 data_type,
column_name3 data_type,
....
)
```

Figura 48 - Sintaxe do comando SQL Create Table (Fonte: Os Autores)

- ***ALTER TABLE*** – modifica a estrutura uma tabela, conforme a figura 49.

```
ALTER TABLE table_name
ADD column_name datatype
```

Figura 49 - Sintaxe do comando SQL Alter Table para adicionar uma coluna à tabela (Fonte: Os Autores)

- ***DROP TABLE*** – exclui uma tabela da base de dados, conforme a figura 50.

```
DROP TABLE table_name
```

Figura 50 - Sintaxe do comando SQL Drop Table (Fonte: Os Autores)

2.9 Arquitetura Orientada a Modelos – Model Driven Architecture (MDA)

Os crescentes avanços tecnológicos, bem como a necessidade de processamento e armazenamento do grande fluxo de informações da atualidade, têm exigido cada vez mais dos softwares disponíveis no mercado para realizar tais tarefas. Por isso, os desenvolvedores de softwares não cessam de buscar novas formas de melhorar a execução dessas tarefas criando aplicativos cada vez mais funcionais e consequentemente complexos, o que transforma sua implementação em uma tarefa árdua.

Outro problema enfrentado pelos desenvolvedores de software é a dificuldade em criar aplicativos de qualidade e com determinada longevidade, além de ter que criar os aplicativos dentro das barreiras impostas pelos recursos disponíveis controlando os custos de implementação do mesmo. No entanto, a longevidade dos aplicativos é diminuída pelo fato de que novas tecnologias surgem a todo o momento e quanto mais elas ficam populares, torna-se mais indispensável que os aplicativos sejam aderentes a essas tecnologias dentro da área de atuação destes (OMG, 2003).

Ainda existe outro ponto a ser questionado: softwares não vivem isolados. Hoje em dia os softwares trabalham em conjunto para realizar as mais diversas tarefas, por isso precisam manter comunicação uns com os outros, o que pode se tornar complicado quando se trata de aplicativos antigos ou muito novos. Sem contar que dependendo da área para que o software é utilizado, existe uma infinidade de informações que o mesmo deve manipular, informações essas que podem mudar constantemente segundo a sua estrutura, forma de processamento e armazenamento.

A Arquitetura Orientada por Modelos (MDA) é uma abordagem para desenvolvimento de software disponibilizada pela Grupo de Gerenciamento de Objetos (OMG) (OMG, 2003), que se baseia na utilização de modelos de diferentes níveis de abstração.

Os modelos são separados em grupos, de acordo com seu nível de abstração, partindo da especificação do sistema mais abstrata, feita através de um modelo independente do meio computacional, até a efetiva implementação física do sistema, passando nesse processo por transformações de modelos, conforme ilustrado na figura 51.

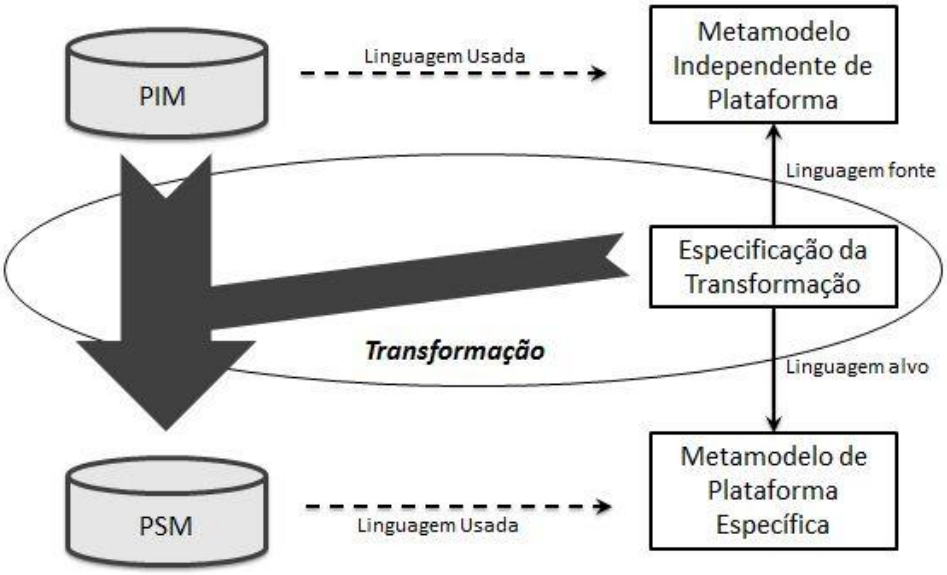

Figura 51 - Transformação do Metamodelo (OMG, 2003)

A MDA permite que os desenvolvedores construam modelos de partes do projeto sem o conhecimento de outros modelos da aplicação e depois os combine para criar a aplicação em si. O uso dos modelos permite que o aplicativo seja mais independente de tecnologia, pois os modelos podem ser combinados entre si e novos mo-

delos de novas tecnologias podem ser adicionados. Isso aumenta a interoperabilidade do projeto e facilita a manutenção (Mellor et al., 2005).

A abordagem MDA é baseada em modelos, no relacionamento entre estes modelos, bem como na transformação de modelos que resultam na aplicação completa. Essa abordagem também possibilita o desenvolvimento de forma mais rápida e consistente, independente de uma plataforma específica. Os modelos podem ser divididos em quatro níveis diferentes de abstração (Mellor et al., 2005):

- **M3 = metametamodelo:** esse nível contém os dados do aplicativo.
- **M2 = metamodelo:** esse nível contém o aplicativo em si.
- **M1 = modelo:** esse nível contém os metadados que capturam a linguagem de modelagem.
- **M0 = dados:** esse nível contém os metadados que descrevem as propriedades que o mesmo pode exibir.

Segundo (Mellor et al., 2005), modelos consistem em conjuntos de elementos que descrevem alguma realidade física, abstrata ou hipotética. É ainda uma simplificação de alguma coisa que nós podemos visualizar, manipular e raciocinar a respeito dela e, portanto, ajuda-nos a entender a complexidade inerente ao sujeito que está sendo estudado.

Um modelo pode ser considerado uma abstração de algo que existe no mundo real e o mesmo é formado por um grupo de informações, onde são descritos os componentes acerca do que está sendo modelado, que podemos chamar de objeto. Por meio dessas informações, é possível extrair regras e conceitos, os quais permitem a reprodução do objeto modelado. Dentro do MDA essas regras e conceitos são chamados de linguagem de modelagem (Mellor et al., 2005), .

Para chegar a um bom modelo é preciso utilizar-se da abstração, pois por intermédio da mesma, é possível incluir no modelo apenas as informações mais importantes sobre o objeto modelado, deixando de fora informações que possuem certa insignificância em relação às demais.

Na MDA criam-se vários modelos, que podem ser confeccionados em diferentes níveis de abstração. Esses modelos depois são interligados a fim de formar de fato, o objeto final. Esses modelos poderão ser combinados com outros modelos a fim de formar outros objetos finais. Além de classificá-los como modelos abstratos ou concretos, podemos classificar os vários modelos como (OMG 2003):

- **Modelo Independente de Computação** (*Computation Independent Model*-CIM) é a representação mais abstrata do sistema, onde não são descritos seus detalhes computacionais, tendo como foco a especificação do sistema que se deseja desenvolver;
- **Modelo Independente de Plataforma** (*Platform Independent Model*-PIM) é menos abstrato que o CIM, contém parte da lógica do sistema e algumas considerações técnicas em relação à sua estrutura; é considerado um refinamento do CIM;
- **Modelo Específico de Plataforma** (*Platform Specifc Model*-PSM) é um refinamento do PIM; contém as especificações da plataforma onde o sistema será implementado.

Um metamodelo é simplesmente um modelo da linguagem de modelagem. Ele define a estrutura, a semântica e as restrições de uma família de modelos (Mellor et al., 2005). Um metamodelo é a descrição de uma linguagem de modelagem a qual contém todos os conceitos pertencentes aos modelos que serão modelados utilizando a linguagem em questão. Ele é utilizado para simplificar a comunicação e o relacionamento entre os modelos.

Segundo (OMG, 2003), um metamodelo é o resultado de um processo de abstração, classificação e generalização sobre o domínio da linguagem de modelagem, então, um metamodelo é um modelo da linguagem de modelagem.

A utilização da MDA no processo elaborado nesse trabalho de transformação de um modelo definido de acordo com determinada modelagem conceitual, para SGBD relacional, em código SQL-DDL, confere grande capacidade de adaptabilidade a mudanças futuras, pois por organizar os modelos de acordo com seu nível de abstração, caso seja necessário tornar o metamodelo definido aderente a uma nova notação de modelagem conceitual, que se encontra no nível de Modelo Independente de Plataforma, as mudanças feitas para tal não afetarão necessariamente as regras de transformação, pois estas estão em outro nível de abstração, de Modelo de Plataforma Específica.

Da mesma forma, não haveria necessariamente mudanças no metamodelo caso fosse preciso gerar código em um padrão diferente utilizando as regras de transformação feitas com a linguagem MOFM2T, sendo as mudanças feitas apenas nesse nível de abstração, o que permite modularidade entre as partes que compõem a MDA facilitando a manutenção e novas atualizações do software criado de acordo com essa arquitetura.

2.10 Transformações de Modelos

Transformações de modelos são os processos pelos quais transforma-se um modelo em outro modelo que retrate o mesmo sistema, de um ponto de vista semelhante ou diferente do modelo anterior, realizada de forma manual ou automática através de regras especificadas (OMG, 2003). Essas transformações podem ocorrer entre modelos de diferentes níveis de abstração, através de duas formas (OMG, 2003):

- **Transformações Modelo-Modelo** são as transformações que manipulam exclusivamente modelos, recebendo um ou mais modelos de entrada e gerando um ou mais modelos como saída. Como exemplo, pode-se citar a transformação de CIM para PIM ou de PIM para PSM.

- **Transformações Modelo-Texto** são as transformações que a partir de um modelo de entrada geram código-fonte de forma direta, sem a transformação para um outro modelo intermediário. Essas transformações recebem um ou mais modelos e geram código-fonte para uma determinada linguagem.

A metodologia utilizada nesse trabalho (Gonçalves, I.S et al.,2012) utiliza transformações de modelo para texto, transformando modelos conceituais definidos utilizando determinada modelagem conceitual para código com o núcleo comum entre os padrões SQL ANSI 92/99/03 DDL.

3 Metodologia MDA Para Modelagem Conceitual de Banco de Dados

Este capítulo demonstra a metodologia proposta em (Rosa, A. et al., 2012) e aperfeiçoada em (Rosa, A. S. et al., 2012) (Gonçalves, I. S. et al., 2012) para modelagem conceitual de banco de dados, apresentando o metamodelo definido de forma que seja aderente às diversas notações de modelagem conceitual para SGBD relacional e, posteriormente, às regras de transformação definidas na linguagem MOFM2T e à forma que elas interagem entre si para gerar o código compatível com o núcleo comum entre os padrões SQL ANSI 92/99/03 DDL.

3.1 Metamodelo

O metamodelo foi definido observando os conceitos que existem em comum nas notações de modelagem conceitual utilizadas para modelagem destinadas a SGBD relacionais, representando esses conceitos através de elementos definidos e das interações entre eles. Assim, caso seja definido um modelo usando uma notação aderente ao metamodelo, ele passará pelas regras de transformação gerando o código DDL desejado.

O *Model*, primeiro elemento do metamodelo, é responsável pelos modelos utilizados na modelagem de base de dados que serão instanciados. O elemento *Database* é responsável pela base de dados que será gerada pela notação de modelagem conceitual. Um banco de dados é composto por, pelo menos, um elemento de entidade, para evitar um banco de dados vazio.

O elemento *Entity* representa uma entidade que tem algumas informações a serem mantidas na base de dados, sendo essas informações, representadas através do metamodelo pelo elemento *Field*. Há também um auto relacionamento chamado *subGroupOf*, que é responsável pela construção de especialização de um grupo de entidades e verificação de elementos de *Entity* para satisfazer o comando de verificação SQL. Um elemento de *Field* tem relações com *PrimaryKey* e *ForeignKey*, elementos-chave a serem implementadas na camada física e também tem relações com os elementos de restrições SQL *TextLimit*, *Integrity*, *DefaultValue* e *NumericLimit*.

Optou-se por representar no metamodelo *PrimaryKey* e *ForeignKey* como componentes individuais em relação à *Field*, ao invés de serem uma especialização do mesmo, embora sejam dependentes da sua existência para serem declarados pelo fato de ocorrem casos de modelagem em que uma *PrimaryKey* de uma entidade tam-

bém pode ser uma *ForeignKey* para outra, não podendo assim, haver uma opção exclusiva, sendo mais coerente modelar para esse caso o metamodelo na forma em que se encontra para melhor representação.

A interação entre as entidades em notações de modelagem de banco de dados é representada por suas relações umas com as outras e, deste modo, surge a necessidade da representação das mesmas através do elemento *Relationship* definido no metamodelo para satisfazer essa condição. Esse elemento está relacionado com elementos *Entity* para determinar quem são os elementos envolvidos nesse relacionamento.

Uma relação possui cardinalidades limitando a participação de cada elemento *Entity* na mesma que pode ser de 1:1, 1: N , N: N, e demais intervalos para limitação de paticipação mínima e máxima no relacionamento, que podem ser especificados pelo projetista através da definição de um elemento *Cardinality*, que possiu o atributo *limit* onde é setado, de forma textual, o valor desejado para a limitação. Para a cardinalidade N:N, uma tabela associativa é gerada a partir da relação e tem *Field*, *ForeignKey* e elementos *PrimaryKey* como um elemento de Entidade. O metamodelo proposto pode ser visto na figura 52.

3.2 Regras de Transformação

As transformações de modelos, quando automatizadas, ocorrem através das ferramentas para esse fim que utilizam regras definidas com uma linguagem de transformação, podendo essas linguagens ser voltadas para transformação de modelos para modelos ou de modelos para texto, que é o padrão assumido nesse trabalho através da linguagem MOFM2T (OMG, 2008). As regras estão organizadas em *templates* que, além de realizar suas funções, alguns deles, são responsáveis por evocar outros *templates* que continuam o processo de transformação. Em seguida serão apresentadas as regras de transformação que, após aplicadas ao modelo instanciado na ferramenta proposta, dão origem ao código DDL correspondente a esse modelo.

3.2.1 Template ModelToText

Esse template é o ponto de entrada para o início da transformação do modelo instanciado, sendo responsável por iniciar a criação do código DDL de definição da base de dados, recebendo a modelagem que foi definida através da árvore hierárquica e iniciando a evocação dos demais *templates* que irão gerar a parte do código DDL que estão encarregados.

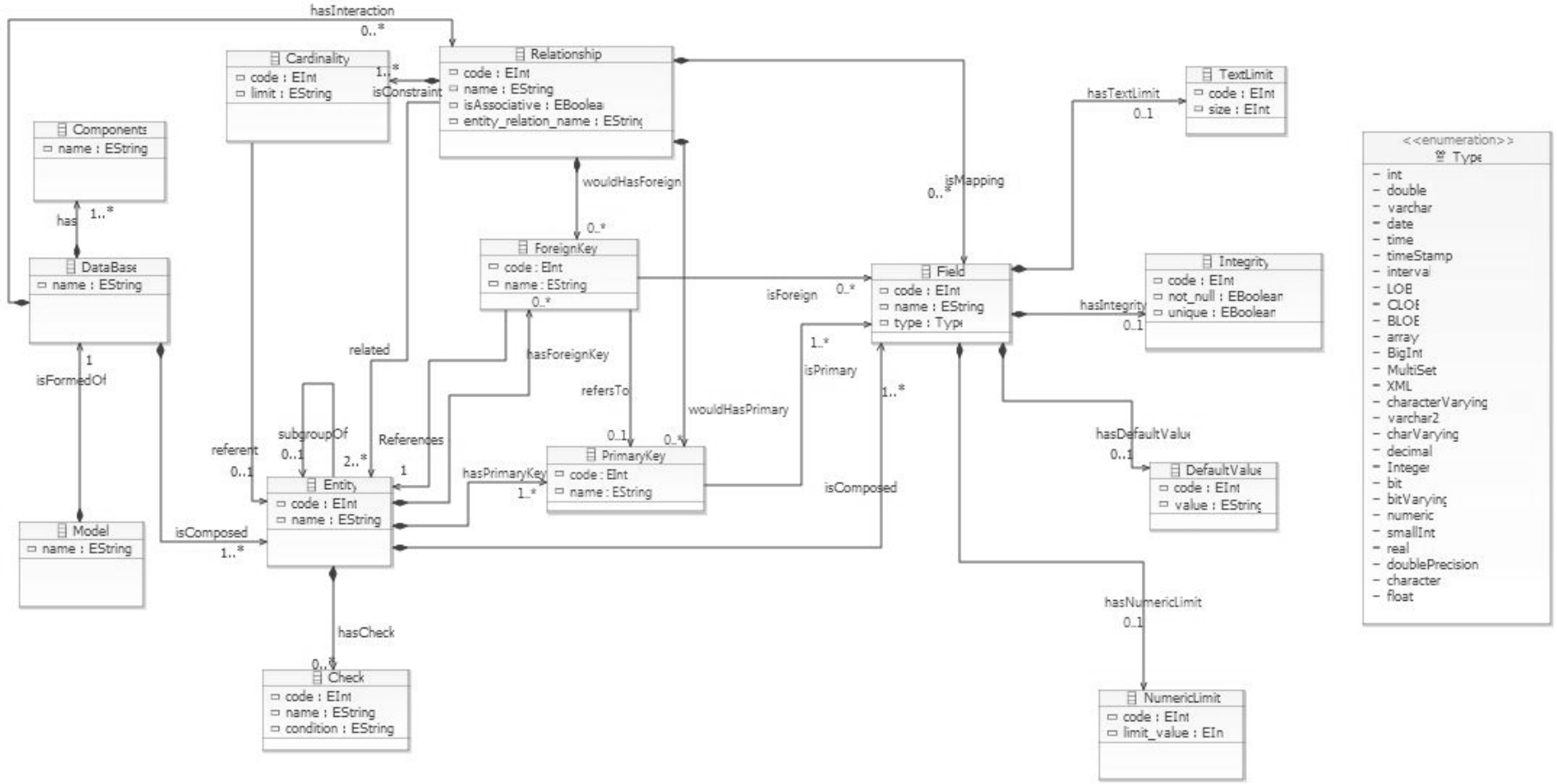

Figura 52 - Metamodelo Proposto (Fonte: Os Autores)

O *template ModelToText* recebe um elemento *Model* e, a partir dele, cria um arquivo no formato .txt, onde é armazenado o código gerado durante todo o processo de transformação, que recebe como nome o texto presente no atributo *name* do objeto *Model*. Esse *template* evoca o template *ToDataBase*, dando continuidade ao processo de transformação, conforme demonstra a figura 53.

```
[template public ModelToText(aModel : Model)]
[comment @main/]

[file (aModel.name+'.txt', false, 'UTF-8')]
    [ToDataBase(aModel.isFormedOf)/]
[/file]
[/template]
```

Figura 53 - Template ModelToText (Fonte: Os Autores)

3.2.2 Template ToDataBase

Esse *template* foi construído para ser o ponto da transformação onde se dará início à geração do código DDL dos componentes da base de dados definida através da modelagem feita. Ele recebe um elemento *DataBase*, e a partir dele, imprime o código *CREATE DATABASE* nomeando a base de dados segundo o texto que estiver presente no atributo *name* do elemento. Esse *template* evoca os *templates PrintPrecedenceA*, *PrintPrecedenceB* e *CreateAssociativeTable*, que vão gerar o código DDL de todos os componentes necessários à base de dados que será definida, conforme demonstrado figura 54.

```
[template public ToDataBase(aDataBase : DataBase)]
CREATE DATABASE [aDataBase.name/];

[PrintPrecedenceA(aDataBase.name,aDataBase.isComposed)/]
[PrintPrecedenceB(aDataBase.name, aDataBase.isComposed)/]
[CreateAssociativeTable(aDataBase.name,aDataBase.hasInteraction)/]
[/template]
```

Figura 54 - Template ToDataBase (Fonte: Os Autores)

3.2.3 Template PrintPrecedenceA

Algumas entidades possuem chaves estrangeiras para outras entidades e, assim, deve-se tomar cuidado com a ordem de criação das tabelas, para evitar que seja criada uma entidade que possua uma ou mais chaves estrangeiras para tabelas que

ainda não existem, sendo então esse o propósito de criação do *template PrintPrecedenceA*.

Conforme exibido na figura 55, esse *template* recebe o nome da base de dados e um conjunto de elementos *Entity*, fazendo uma iteração no conjunto de elementos e verificando se a entidade em questão possui ou não chaves estrangeiras e, caso ela possua, sua tabela não é criada nesse momento, ficando a cargo de outro *template*. Caso ela não possua chaves estrangeiras, o *template PrintEntityWithoutForeign* é evocado para geração de código da mesma, sendo feita essa verificação para todos os elementos do conjunto recebido.

```
[template public PrintPrecedenceA(aDataBaseName : String,
aEntity : OrderedSet(Entity))]
[for (iEntity : Entity | aEntity)]
[if (iEntity.hasForeignKey->isEmpty())]
[PrintEntityWithoutForeign(aDataBaseName,iEntity)/]
[/if]
[/for]
[/template]
```

Figura 55 - Template PrintPrecedenceA (Fonte: Os Autores)

3.2.4 Template PrintPrecedenceB

PrintPrecedenceB é o segundo *template* evocado pelo *template ToDataBase*, que foi criado com um propósito parecido com o do *template PrintPrecedenceA*, porém agora o foco da verificação é identificar as entidades que possuam uma ou mais chaves estrangeiras para outras entidades para gerar o seu respectivo código, já que todas as que não possuem já tiveram seu código gerado no *template* evocado anteriormente.

Como está sendo exemplificado na figura 56, o *template* recebe como parâmetro o nome da base de dados e um conjunto de elementos *Entity*, faz uma iteração no conjunto de elementos e verifica se a entidade em questão possui ou não chaves estrangeiras e, caso possua, o *template PrintEntityWithForeign* é evocado para gerar seu respectivo código. Esse processo é feito para todos os elementos do conjunto recebido.

3.2.5 Template CreateAssociativeTable

O *template CreateAssociativeTable* é construído com o objetivo de verificar se, entre os relacionamentos definidos na modelagem, algum relacionamento dará origem

à criação de uma tabela associativa na geração do código DDL da base de dados e, caso exista essa necessidade, fazer a geração do respectivo código da tabela.

```
[template public PrintPrecedenceB(aDataBaseName: String,
 aEntity : OrderedSet(Entity))]
[for (iEntity : Entity | aEntity)]
[if (iEntity.hasForeignKey->notEmpty())]
[PrintEntityWithForeign(aDataBaseName,iEntity)/]
[/if]
[/for]
[/template]
```

Figura 56 - Template PrintPrecedenceB (Fonte: Os Autores)

Esse *template* recebe o nome da base de dados e um conjunto de elementos *Relationship*, faz a iteração no conjunto e verifica se o atributo *isAssociative* do elemento *Relationship* em questão possui o valor setado como *true*; caso possua, o *template* imprime no código gerado a estrutura de definição da tabela que representa esse relacionamento, imprimindo a linha *CREATE TABLE* e a nomeia colocando, primeiramente, o nome da base de dados seguido de um ponto e logo após colocando o texto presente no atributo *name* do elemento. Caso o atributo não possua valor setado como *true*, significa que o relacionamento não dará origem a uma tabela associativa, sendo desnecessária a criação de uma nova tabela para esse fim.

Para verificar se o relacionamento dá origem a uma tabela associativa, ou não, também pode ser utilizada a verificação das cardinalidades associadas ao elemento *Relationship*, porém foi optado pela utilização do atributo *isAssociative* por facilitar a implementação do código.

A verificação do relacionamento através da cardinalidade foi testada através da implementação de um *template,* que teria o objetivo de verificar as cardinalidades associadas; caso duas das cardinalidades fossem definidas como N, seria criada a tabela associativa. Porém, durante a implementação do código, ao fazer a iteração através das cardinalidades envolvidas para diferencia-las umas das outras e armazenar seu valor definido para comparar com as demais, foi encontrada uma dificuldade em relação a manipulação de variáveis na linguagem. Pois, as variáveis na linguagem MOFM2T devem ter seu valor atribuído no momento de sua declaração, sendo esse imutável após sua definição e só são reconhecidas dentro de um bloco que deve ser inicialmente estabelecido.

A abordagem que se conseguiu chegar para o *template* que fazia a verificação das cardinalidades utilizava no mínimo dois laços de repetição distintos, o que dificultou a definição dos blocos para a declaração de variáveis que armazenariam as cardi-

nalidades encontradas para comparação de valores. Diante da dificuldade de se chegar a um resultado efetivo para a verificação através das cardinalidades associadas foi então adotada a verificação do atributo *isAssociative*.

Para gerar o código DDL que irá descrever as características da tabela associativa, o *template CreateAssociativeTable* evoca os *templates PrintRelationCharacteristics*, *PrintRelationFields*, *PrintPrimary* e *PrintForeignKeyFieldsN_N*, como exibido na figura 57.

```
[template public CreateAssociativeTable(aDataBaseName : String, aRelations
: OrderedSet(Relationship))]
[for (iRelation : Relationship | aRelations)]
    [if (iRelation.isAssociative = true)]
        CREATE TABLE [aDataBaseName/].[iRelation.name/]
        (
        [PrintRelationCharacteristics(iRelation.related)/]
        [PrintRelationFields(iRelation)/]
        [PrintPrimary(iRelation.wouldHasPrimary)/]
        [PrintForeignKeyFieldsN_N(iRelation)/]
        );
    [/if]
[/for]
[/template]
```

Figura 57- Template CreateAssociativeTable (Fonte: Os Autores)

3.2.6 Template PrintEntityWithoutForeign

O *template PrintEntityWithoutForeign* foi construído com o objetivo de gerar o código relativo à definição de dados das entidades que não possuem chave estrangeira para outras entidades.

PrintEntityWithoutForeign recebe como parâmetro o nome da base de dados e um conjunto de elementos *Entity*. O *template* faz uma iteração no conjunto de elementos e, para cada elemento *Entity* encontrado, cria a estrutura da tabela que representa a entidade em questão, imprimindo a linha *CREATE TABLE*, o nome da base de dados seguido de um ponto e, logo após, nomeia a tabela colocando o texto presente no atributo *name* do elemento.

O *template PrintEntityWithoutForeign* evoca alguns *templates* para gerar a descrição das características da entidade, sendo eles os *templates ToField*, *PrintPrimary* e por último verifica se a entidade possui interação com algum elemento *Check*; caso tenha, ele evoca também o *template PrintCheck*, caso não possua, esse último *template* não é evocado, confome a figura 58.

```
[template public PrintEntityWithoutForeign(aDataBaseName : String, aEntity : Entity)]
[for (iEntity : Entity | aEntity)]
CREATE TABLE [aDataBaseName/].[iEntity.name/](
[ToField(iEntity.isComposed)/]
[PrintPrimary(iEntity.hasPrimaryKey)/]
[if (iEntity.hasCheck<>null)][PrintCheck(iEntity.hasCheck)/][/if]
);
[/for]
[/template]
```

Figura 58 - Template PrintEntityWithoutForeign (Fonte: Os Autores)

3.2.7 Template PrintEntityWithForeign

PrintEntityWithForeign foi construído com o propósito de gerar o código referente à definição de dados das entidades que possuem chave estrangeira para outras entidades.

Como está sendo exibido na figura 59, esse *template* recebe o nome da base de dados e um conjunto de elementos *Entity* como parâmetro, faz uma iteração no conjunto de elementos e, para cada elemento *Entity* encontrado e cria a estrutura da tabela que representa a entidade em questão. Para isso, o *template* imprime a linha *CREATE TABLE*, faz a nomeção da tabela colocando primeiramente, o nome da base de dados, seguido de um ponto e, logo após, coloca o texto presente no atributo *name* do elemento.

O *template* evoca, para definição de dados da tabela que representa a entidade a ser definida, os *templates ToField*, *PrintPrimary* e *PrintForeignKey* e por último, verifica se a entidade possui interação com algum elemento *Check*; caso possua, ele evoca também o *template PrintCheck*, caso não possua, esse último *template* não é evocado.

```
[template public PrintEntityWithForeign(aDataBaseName : String, aEntity : Entity)]
[for (iEntity : Entity | aEntity)]
CREATE TABLE [aDataBaseName/].[iEntity.name/](
[ToField(iEntity.isComposed)/]
[PrintPrimary(iEntity.hasPrimaryKey)/]
[PrintForeign(iEntity.hasForeignKey)/]
[if (iEntity.hasCheck<>null)][PrintCheck (iEntity.hasCheck)/][/if]
);
[/for]
[/template]
```

Figura 59 - Template PrintEntityWithForeign (Fonte: Os Autores)

3.2.8 Template ToField

O *template ToField* foi construído para gerar a parte do código DDL referente aos atributos definidos para descrição das características de determinada entidade.

Ele recebe um conjunto de elementos *Field* como parâmetro, faz uma iteração no mesmo e cria o código composto, respectivamente, pelos textos presentes nos atributos *name* e *type* do elemento em questão. Em seguida, verifica se o elemento possui interação com algum elemento *NumericLimit*; caso possua, o *template PrintNumericLimit* é evocado. Após isso, verifica se o elemento possui interação com algum elemento *TextLimit*; caso possua, o *template PrintTextLimit* é evocado e da mesma forma também ocorre a verificação para constatar se o elemento *Field* possui interação com algum elemento *Integrity*; caso possua, o *template PrintIntegrity* é evocado e, por último, será verificado se o elemento *Entity* possui interação com algum elemento *Check;* caso possua, o *template PrintCheck* é evocado, como demonsta a figura 60.

```
[template public ToField(aFields : OrderedSet(Field))]
[for ( iField : Field | aFields ) ]
    [iField.name/] [iField.type/]
    [if (iField.hasNumericLimit <> null)]
        [PrintNumericLimit(iField.hasNumericLimit)/]
    [/if]
    [if (iField.hasTextLimit <> null)]
        [PrintTextLimit(iField.hasTextLimit)/]
    [/if]
    [PrintIntegrity(iField.hasIntegrity)/]
    [if (iField.hasDefaultValue <> null)]
        [PrintDefaultValue(iField.hasDefaultValue)/]
    [/if],
[/for]
[/template]
```

Figura 60 - Template ToField (Fonte: Os Autores)

3.2.9 Template PrintNumericLimit

PrintNumericLimit foi criado para possibilitar a definição de um limite em relação à quantidade de algarismos utilizados para definição de um atributo numérico que componha uma entidade.

Como está sendo exibido na figura 61, esse *template* recebe um elemento *NumericLimit* como parâmetro que possui um campo chamado *limit_value*, onde é atribuído um valor número inteiro que será utilizado para fazer a limitação de quantidade de algarismos que podem ser utilizados para representação de valor do atributo de tipo numérico que está relacionado na geração do código DDL.

```
[template public PrintNumericLimit(aNumeric : NumericLimit)]
([aNumeric.limit_value/])
[/template]
```

Figura 61 - Template PrintNumericLimit (Fonte: Os Autores)

3.2.10 Template PrintTextLimit

PrintTextLimit foi criado para definir o número máximo de caracteres que podem ser utilizados na atribuição do valor de um determinado campo de texto no código DDL relacioando a um atributo.

O *template* recebe um elemento *TextLimit* como parâmetro, que possui um campo *size* onde é setado o valor limite de caracteres que podem ser utilizados em um texto a ser armazenado em determinado atributo, acrescentando esse limite entre parênteses junto do tipo textual que o atributo possui na geração do seu código DDL, como está sendo mostrado na figura 62.

```
[template public PrintTextLimit(aText: TextLimit)]
([aText.size/])
[/template]
```

Figura 62 - Template PrintTextLimit (Fonte: Os Autores)

3.2.11 Template PrintIntegrity

Esse *template* foi criado para definir no código DDL se é ou não permitido nulidade de valor à determinado atributo e se esse atributo possui unicidade de valor entre os registros ta tabela que pertence.

Como mostra a figura 63, *PrintIntegrity* recebe como parâmetro um elemento *Integrity* e verifica, primeiramente, se o atributo *not_null* possui valor igual a *true*. Caso possua, é acrescentado o texto *NOT NULL* à descrição de características do atributo que o elemento *Integrity* está relacionado. Em seguida, verifica se o atributo *unique* do elemento *Integrity* possui valor igual a *true* e, caso possua, também é acrescentado o texto *UNIQUE* à descrição de características do atributo que o elemento *Integrity* está relacionado através do código DDL.

```
[template public PrintIntegrity(aIntegrity : Integrity)]
[if(aIntegrity.not_null=true)]NOT NULL [/if]
[if ( aIntegrity.unique=true)]UNIQUE[/if]
[/template]
```

Figura 63 - Template PrintIntegrity (Fonte: Os Autores)

3.2.12 Template PrintDefaultValue

O *template PrintDefaultValue* foi construído para possibilitar na geração do código DDL a preparação de atribuição de valores *default* a campos desejados no momento da inserção de dados futuras, de forma que, caso não seja passado um valor para o campo que não se deseja permitir nulidade, o valor *default* é assumido evitando, assim, erros na base de dados.

O *template* recebe um elemento *DefaulValue*, que contem um atributo de nome *value* onde o valor default que se deseja atribuir ao campo relacionado a esse elemento é inserido. É feito então o acréscimo na geração do código DDL do campo relacionado à descrição do valor padrão que o mesmo possui, sendo inserido o texto *DEFAULT*, seguido do valor pré-determinado, como apresentado na figura 64.

```
[template public PrintDefaultValue(aDefault : DefaultValue)]
DEFAULT '[aDefault.value/]'
[/template]
```

Figura 64 - Template PrintDefaultValue (Fonte: Os Autores)

3.2.13 Template PrintCheck

É possível limitar um intervalo para os valores que podem ser atribuídos a um atributo de uma entidade, no caso, a uma coluna da tabela da base de dados. Isso é feito através do acréscimo de uma restrição limitativa direcionada ao atributo desejado. Esse *template* foi construído para fazer o acréscimo da restrição limitativa direcionada ao atributo que se deseja validar no código DDL da entidade que possui essa necessidade.

O *template* recebe um conjunto de elementos *Check*, faz uma iteração no conjunto e para cada elemento, acrescenta no código DDL da entidade relacionada respectivamente, o texto *CONSTRAINT*, o texto contido no atributo *name* do elemento, o texto *CHECK* e, entre parênteses, o valor contido no atributo *condition* do elemento, como mostra a figura 65.

```
[template public PrintCheck(aCheck : OrderedSet(Check))]
[for (iCheck : Check | aCheck) ]
    CONSTRAINT [iCheck.name/] CHECK ([iCheck.condition/])
[/for]
[/template]
```

Figura 65 - Template PrintCheck (Fonte: Os Autores)

3.2.14 Template PrintCandidateKey

PrintCandidateKey foi construído com o objetivo de gerar o código DDL das chaves candidatas existentes na tabela que representa um relacionamento associativo, onde essas chaves são as chaves primárias das entidades que participam da relação. De acordo com a figura 66, o *template* recebe um elemento *Entity* e verifica quais são as chaves primárias da mesma, ou seja, verifica quais elementos *PrimaryKey* possuem interação com o elemento em questão, evocando então o *template ToField* para gerar o código DDL dos campos eleitos como chave primária na entidade participante da relação.

```
[template public PrintCandidateKey(aEntity : Entity)]
[for (iPrimary : PrimaryKey | aEntity.hasPrimaryKey)]
    [ToField(iPrimary.isPrimary)/]
[/for]
[/template]
```

Figura 66 - Template PrintCandidateKey (Fonte: Os Autores)

3.2.15 Template PrintPrimary

O *template PrintPrimary* foi criado para gerar o código DDL relativo às chaves primárias das tabelas que estarão presentes na base de dados.

O *template* recebe um conjunto de elementos *PrimaryKey*, faz iteração no conjunto e, para cada elemento, cria o código composto, respectivamente, pelo texto *CONSTRAINT* , o texto presente no atributo *name* do elemento, o texto *PRIMARY KEY* e, novamente, porém dessa vez entre parênteses, o texto do atributo *name* do elemento, como mostra a figura 67.

```
[template public PrintPrimary(aPrimaryKeys : OrderedSet(PrimaryKey))]
[for ( iPrimaryKey : PrimaryKey | aPrimaryKeys ) ]
    CONSTRAINT [iPrimaryKey.isPrimary.name/]
    PRIMARY KEY([iPrimaryKey.isPrimary.name/]),
[/for]
[/template]
```

Figura 67 - Template PrintPrimary (Fonte: Os Autores)

3.2.16 Template PrintForeign

O *template PrintForeign* foi criado para gerar o código DDL referente às chaves estrangeiras das tabelas criadas para a interação entre as mesmas na base de dados. Como mostra a figura 68, esse *template* recebe um conjunto de elementos *ForeignKey*, faz iteração no conjunto e verifica se o atributo *name* do elemento em ques-

tão não possui valor nulo; caso não possua, cria o código composto, respectivamente, pelo texto *CONSTRAINT*, o texto presente no atributo *name* do elemento, o texto *FOREIGN KEY* e, novamente, porém dessa vez entre parênteses, o texto do atributo *name* do elemento.

```
[template public PrintForeign(aForeignKeys : OrderedSet(ForeignKey))]
[for ( iForeignKey : ForeignKey | aForeignKeys ) separator (', ')]
    [if (iForeignKey.name <> null)]
        CONSTRAINT [iForeignKey.name/]
        FOREIGN KEY ([iForeignKey.isForeign.name/])
        [PrintReferences(iForeignKey)/]
    [/if]
[/for]
[/template]
```

Figura 68 - Template PrintForeignKey (Fonte: Os Autores)

3.2.17 Template PrintForeignKeyNpN

O *template PrintForeignKeyNpN* foi criado para gerar o código DDL das chaves estrangeiras das tabelas associativas criadas a partir de alguns relacionamentos entre entidades.

O *template* recebe um elemento *Relationship* e, a partir dele, acessa cada elemento do conjunto de elementos *ForeignKey* relacionado e cria o código composto, respectivamente, pelo texto *CONSTRAINT*, o texto presente no atributo *name* do elemento *ForeignKey*, o texto *FOREIGN KEY*, acrescenta novamente, porém dessa vez entre parênteses, o texto do atributo *name* do elemento *ForeignKey* e evoca o *template PrintReferences,* conforme está sendo exibido na figura 69 .

```
[template public PrintForeignKeyFieldsN_N ( aRelation: Relationship)]
[for (iForeign : ForeignKey | aRelation.wouldHasForeign)]
    CONSTRAINT [iForeign.name/]
    FOREIGN KEY ([iForeign.refersTo.isPrimary.name/])
    [PrintReferences(iForeign)/]
[/for]
[/template]
```

Figura 69 - Template PrintForeignKeyFieldsN_N (Fonte: Os Autores)

3.2.18 Template PrintReferences

Esse *template* é responsável pela criação do código DDL das referencias das chaves estrangeiras das tabelas. O *template* recebe um elemento *ForeignKey* e cria o código composto, respectivamente, pelo texto REFERENCES, o nome da entidade que a chave estrangeira está referenciando, ou seja, da tabela referenciada e, entre

parênteses, o nome da chave primária que está sendo referenciada, como ilustra a figura 70.

```
[template public PrintReferences(aForeign : ForeignKey)]
REFERENCES [aForeign.References.name/] ([aForeign.refersTo.isPrimary.name/])
[/template]
```

Figura 70 - Template PrintReferences (Fonte: Os Autores)

3.2.19 Template PrintRelationCharacteristics

Esse *template* é responsável por evocar o *template* de criação das chaves candidatas das tabelas associativas criadas a partir de alguns relacionamentos entre entidades. O *template* recebe um conjunto de elementos *Entity*, faz iteração no mesmo e evoca o *template PrintCandidateKey* para cada elemento encontrado, conforme mostra 71.

```
[template public PrintRelationCharacteristics(aEntity : OrderedSet(Entity))]
[for ( iEntity : Entity | aEntity)]
    [PrintCandidateKey(iEntity)/]
[/for]
[/template]
```

Figura 71 - Template PrintRelationCharacteristics (Fonte: Os Autores)

3.2.20 Template PrintRelationFields

Esse *template* é responsável por evocar o *template* de criação do código DDL referente aos atributos presentes nas tabelas associativas criadas a partir de alguns relacionamentos. O *template* recebe um elemento *Relationship* como parâmetro e verifica se o atributo *isMapping* do elemento não possui valor nulo, caso não possua, é evocado o *template ToField,* conforme a figura 72.

```
[template public PrintRelationFields(aRelationship : Relationship)]
[if (aRelationship.isMapping<>null)]
    [ToField(aRelationship.isMapping)/]
[/if]
[/template]
```

Figura 72 - Template PrintRelationFields (Fonte: Os Autores)

4 Exemplos utilizando a metodologia

Nesse capítulo, serão apresentados três exemplos de transformações realizados através da ferramenta, utilizando como base para teste os exemplos de modelagem conceitual segundo as notações ER, Crow's Foot e IDEF1X .

A ferramenta para testes foi um conjunto de *plugins* desenvolvido pelos autores desse trabalho, construído na plataforma de desenvolvimento Eclipse, utilizando o Eclipse Modeling Framework (Steinberg et al.,2008) para criar o metamodelo com base na abordagem MDA. O gerador de códigos Acceleo (Acceleo: MDA generator-Home) foi usado para permitir a implementação de regras de transformação e para gerar o código SQL .

É possível instanciar o modelo desejado na ferramenta utilizando um arquivo XML que conterá as definições do modelo. O arquivo XML pode ser manipulado visualmente no EMF através de uma árvore hierárquica.

4.1 Exemplo A

A figura 73 representa a modelagem conceitual feita de acordo com a notação ER e a figura 74 representa esse modelo após ser instanciado na ferramenta, exposto visualmente através de uma árvore hierárquica contendo os elementos relativos às características presentes na modelagem.

O domínio observado na modelagem da figura 73 se trata de uma fábrica onde devem se manter dados sobre os funcionários (Employee), sobre os dependentes dos funcionários (Dependent), projetos (Project), partes utilizadas pelos projetos (Part e Supplier) e a interação entre essas entidades.

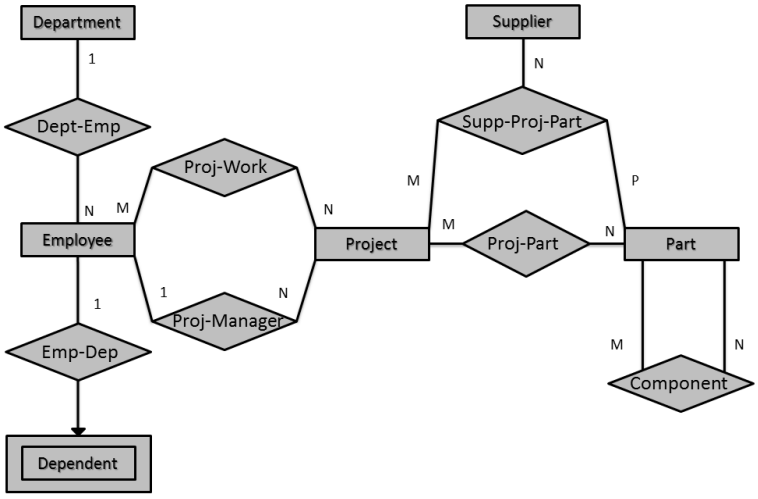

Figura 73 - Exemplo usando a notação Entidade-Relacionamento (CHEN, 1976).

Figura 74 - Representação do modelo conceitual na árvore hierárquica (Fonte: Os Autores).

O código gerado a partir do modelo, após passar pelas regras de transformação, é mostrado abaixo:

```
CREATE DATABASE ManufacturingFirm;

CREATE TABLE ManufacturingFirm.Department(
DepartmentID int (7) NOT NULL UNIQUE,
CONSTRAINT DepartmentID PRIMARY KEY(DepartmentID)        );

CREATE TABLE ManufacturingFirm.Supplier(
SupplierID int (7) NOT NULL UNIQUE,
CONSTRAINT SupplierID PRIMARY KEY(SupplierID));

CREATE TABLE ManufacturingFirm.Part(
PartID int (7) NOT NULL UNIQUE,
CONSTRAINT PartID PRIMARY KEY(PartID));

CREATE TABLE ManufacturingFirm.Employee(
EmployeeID int (7) NOT NULL UNIQUE,
CONSTRAINT EmployeeID PRIMARY KEY(EmployeeID),
CONSTRAINT CodeDept FOREIGN KEY (DepartmentID)REFERENCES Department (DepartmentID));

CREATE TABLE ManufacturingFirm.Dependent(
DependentID int (7) NOT NULL UNIQUE,
CONSTRAINT DependentID PRIMARY KEY(DependentID),
CONSTRAINT CodeEmployee FOREIGN KEY (EmployeeID)REFERENCES Employee (EmployeeID));

CREATE TABLE ManufacturingFirm.Project(
ProjectID int (7) NOT NULL UNIQUE,
CONSTRAINT ProjectID PRIMARY KEY(ProjectID),
CONSTRAINT CodeEmployee FOREIGN KEY (EmployeeID)REFERENCES Employee (EmployeeID));
```

4.2 Exemplo B

Da mesma forma como apresentado no modelo anterior, a figura 75 representa a modelagem conceitual feita com base na notação de Crow's Foot e a figura 76 representa esse modelo após ser instanciado na ferramenta através da árvore hierárquica contendo os elementos utilizados na sua modelagem.

O domínio representado na figura 75 trata um curso a distância onde devem ser mantidos na base de dados sobre os estudantes (Distance Student), sobre o curso frequentado (Course), sobre o registro de notas do estudante (Transcript), sobre os departamentos do curso onde os alunos estão alocados (Department) e da relação que surge entre os estudantes e o curso (Registration date).

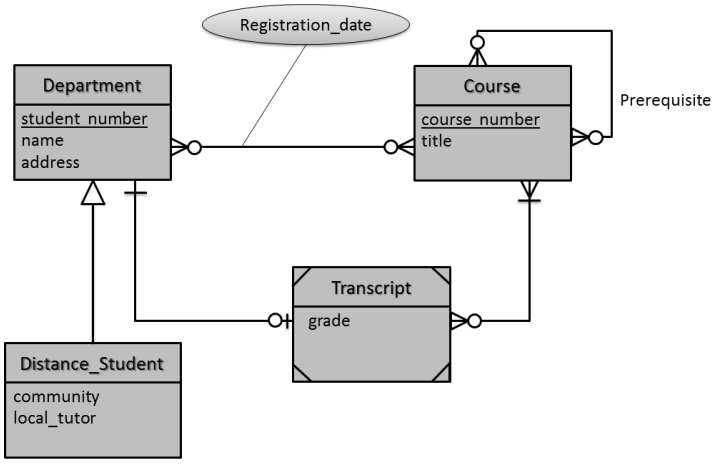

Figura 75 - Exemplo usando a notação Crow's Foot (ROSA et. al, 2012).

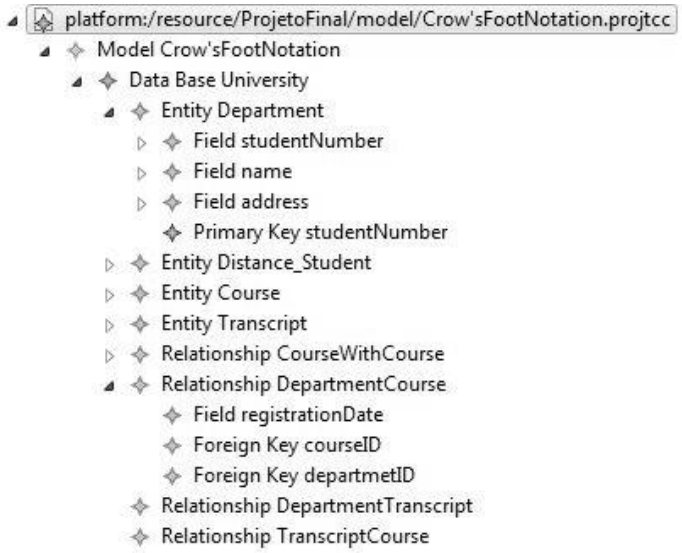

Figura 75 - Representação do modelo conceitual na árvore hierárquica (Fonte: Os Autores).

O código gerado a partir do modelo, após passar pelas regras de transformação, é mostrado abaixo:

```
CREATE DATABASE University;

CREATE TABLE University.Department(
studentNumber int (7) NOT NULL UNIQUE,
name varchar (20) ,
address varchar (45) ,
CONSTRAINT studentNumber PRIMARY KEY(studentNumber));

CREATE TABLE University.Distance_Student(
id_Distance_Studentint (7) NOT NULL UNIQUE,
community varchar (50) ,
local_tutor varchar (50),
CONSTRAINT id_Distance_Student PRIMARY KEY(id_Distance_Student));


CREATE TABLE University.Course(
courseNumber int (7) NOT NULL UNIQUE,
title varchar (20) ,
CONSTRAINT courseNumber PRIMARY KEY(courseNumber));

CREATE TABLE University.Transcript(
id_Transcript int (7) NOT NULL UNIQUE,
grade varchar (20)
CONSTRAINT id_Transcript PRIMARY KEY(id_Transcript));

CREATE TABLE University.CourseWithCourse(
courseNumber int (7) NOT NULL UNIQUE,
CONSTRAINT courseID FOREIGN KEY (courseNumber) REFERENCES Course (courseNumber)
CONSTRAINT prerequisiteID FOREIGN KEY (courseNumber) REFERENCES Course (courseNumber));

CREATE TABLE University.DepartmentCourse(
studentNumber int (7) NOT NULL UNIQUE,
courseNumber int (7) NOT NULL UNIQUE,
registrationDate varchar (10),
CONSTRAINT courseID FOREIGN KEY (courseNumber) REFERENCES Course (courseNumber),
CONSTRAINT departmetID FOREIGN KEY (studentNumber) REFERENCES Department (studentNumber));
```

4.3 Exemplo C

A figura 77 representa um exemplo da modelagem feito com base na notação IDEF1X e a figura 78 representa esse modelo exposto através da árvore hierárquica da ferramenta.

O domínio representado na modelagem da figura 77 trata um gerenciador de emails onde deverão ser mantidos dados sobre as mensagens enviadas (Email Message), sobre os usuários (*User*), sobre os termos de uso e a aceitação dos mesmos (*Terms* e *TermsAgreed*) e da caixa de entrada do usuário (*Recipient*).

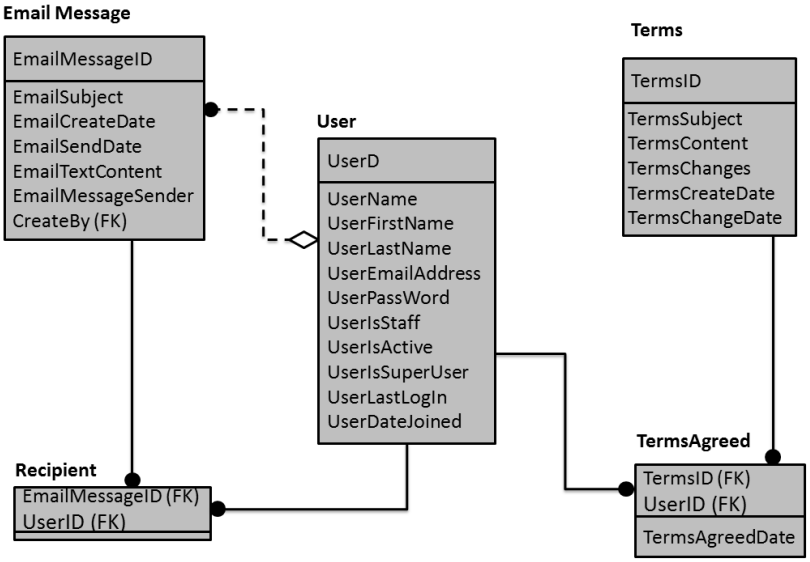

Figura 76 - Exemplo usando a notação IDEF1X (ROSA et. al, 2012)..

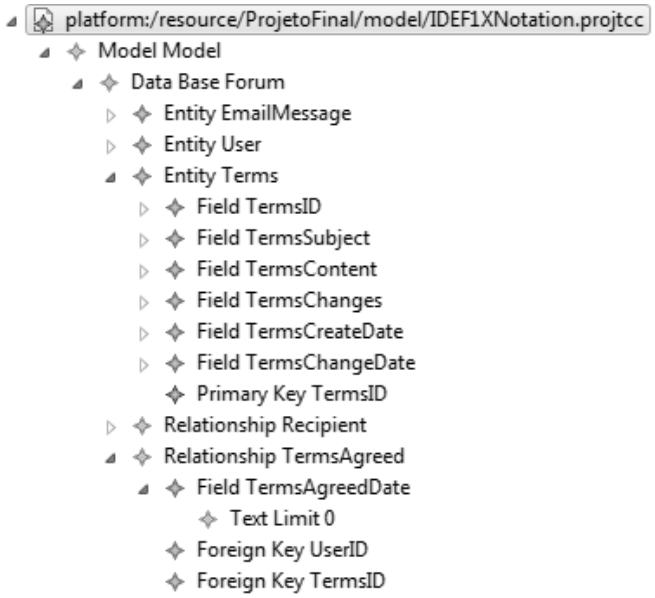

Figura 77 - Representação do modelo conceitual na árvore hierárquica (Fonte: Os Autores).

O código gerado a partir do modelo, após passar pelas regras de transformação, é mostrado abaixo:

```
CREATE DATABASE Forum;

CREATE TABLE Forum.User(
UserID int (7) NOT NULL UNIQUE,
UserFirstName int (30) ,
UserLastName varchar (30) ,
UserEmailAddress varchar (30) ,
UserPassWord varchar (30) ,
UserIsAtaff int (30) ,
UserIsActive int (30) ,
UserIsSuperUser int (30) ,
UserLastLogIn varchar (30) ,
UserDateJoined varchar (30) ,
CONSTRAINT UserID PRIMARY KEY(UserID));
```

```
CREATE TABLE Forum.Terms(
TermsID int (7) NOT NULL UNIQUE,
TermsSubject varchar (30) ,
TermsContent varchar (30) ,
TermsChanges varchar (30) ,
TermsCreateDate varchar (30) ,
TermsChangeDate varchar (30) ,
CONSTRAINT TermsID PRIMARY KEY(TermsID));

CREATE TABLE Forum.EmailMessage(
EmailMessageID int (7) NOT NULL UNIQUE,
EmailSubject varchar (50) ,
EmailCreateDate varchar (50) ,
EmailSendDate varchar (50) ,
EmailTextContent varchar (50) ,
EmailMessageSender varchar (50) ,
CreateBy varchar (50) ,
CONSTRAINT EmailMessageID PRIMARY KEY(EmailMessageID),
CONSTRAINT CreateBy FOREIGN KEY (CreateBy)REFERENCES User (UserID));

CREATE TABLE Forum.Recipient(
EmailMessageID int (7) NOT NULL UNIQUE,
UserID int (7) NOT NULL UNIQUE,
CONSTRAINT EmailMessageID FOREIGN KEY (EmailMessageID) REFERENCES EmailMessage (EmailMessageID)
CONSTRAINT UserID FOREIGN KEY (UserID) REFERENCES User (UserID));

CREATE TABLE Forum.TermsAgreed(
TermsID int (7) NOT NULL UNIQUE,
UserID int (7) NOT NULL UNIQUE,
TermsAgreedDate varchar (10) ,
CONSTRAINT UserID FOREIGN KEY (UserID) REFERENCES User (UserID)
CONSTRAINT TermsID FOREIGN KEY (TermsID) REFERENCES Terms (TermsID));
```

5 Experimentos

O código DDL gerado na ferramenta possui um núcleo com aspectos comuns aos padrões ANSI SQL 92/99/2003, isso permite que o modelo físico de dados, criado através da ferramenta possa ser implementado em SGBD que também sejam aderentes à esses padrões. Porém, alguns SGBD adotam instruções SQL diferentes do padrão, instruções com extensões próprias. Isso faz com que um modelo físico de banco de dados, quando transportado entre SGBD diferentes, na maioria das vezes, precise de significativas modificações. Os vendedores de SGBD criam extensões para melhorar o desempenho e tornar mais fácil a manipulação dos dados. Eles se preocupam em: criar soluções rápidas, tornar a implementação mais atrativa e ter vantagens em relação aos concorrentes (Lima, 2006).

Isso faz com que um modelo físico de banco de dados criado através da ferramenta proposta precise de modificações quando implementado em alguns SGBD. Embora o código gerado seja aderente aos padrões, se o SGBD em que for implementado não for adente ou possuir alguma modificação em relação ao padrão, o código pode precisar de alterações.

O modelo apresentado na figura 79 foi utilizado nos experimentos e a figura 80 representa o modelo instanciado na árvore hierárquica da ferramenta. O código gera-

do foi implementado em alguns SGBD disponíveis, os quais possuem suas diferenças e particularidades.

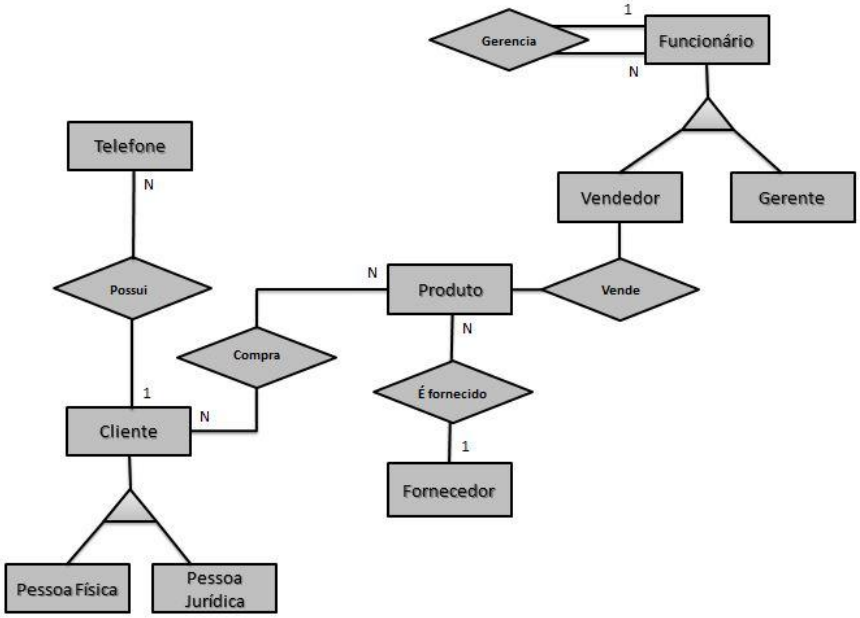

Figura 78 - Exemplo utilizado nos experimentos (Fonte: Os Autores).

Figura 79 - Modelo instanciado na ferramenta (Fonte: Os Autores).

O código gerado a partir do modelo, após passar pelas regras de transformação, é mostrado abaixo:

```
CREATE DATABASE Loja;

CREATE TABLE Loja.Telefone(
idTelefone int  NOT NULL UNIQUE,
DDD int  ,
numero int  ,
CONSTRAINT idTelefone PRIMARY KEY(idTelefone));

CREATE TABLE Loja.Produto(
idProduto int  NOT NULL UNIQUE,
nome varchar (50) ,
tipo varchar (25) ,
precoVenda double  ,
precoCompra double  ,
CONSTRAINT idProduto PRIMARY KEY(idProduto),
CONSTRAINT precoVenda CHECK (precoVenda>precoCompra));

CREATE TABLE Loja.Fornecedor(
idFornecedor int  NOT NULL UNIQUE,
nome varchar (50) ,
CNPJ varchar (20) ,
endereco varchar(50) ,
CONSTRAINT idFornecedor PRIMARY KEY(idFornecedor)        );

CREATE TABLE Loja.Cliente(
idCliente int  NOT NULL UNIQUE,
nome varchar (50) ,
endereco varchar (50) ,
idTelefone int  ,
CONSTRAINT idCliente PRIMARY KEY(idCliente),
CONSTRAINT idTelefone FOREIGN KEY (idTelefone)REFERENCES Telefone (idTelefone));

CREATE TABLE Loja.PessoaFisica(
idPessoaFisica int  NOT NULL UNIQUE,
idCliente int  ,
sexo varchar (10) DEFAULT 'M',
CPF varchar (15) ,
CONSTRAINT idPessoaFisica PRIMARY KEY(idPessoaFisica),
CONSTRAINT idCliente1 FOREIGN KEY (idCliente)REFERENCES Cliente (idCliente));

CREATE TABLE Loja.PessoaJuridica(
idPessoaJuridica int  NOT NULL UNIQUE,
idCliente int  ,
CNPJ varchar (20) ,
CONSTRAINT idPessoaJuridica PRIMARY KEY(idPessoaJuridica),
CONSTRAINT idCliente2 FOREIGN KEY (idCliente)REFERENCES Cliente (idCliente));

CREATE TABLE Loja.Funcionario(
idFuncionario int  NOT NULL UNIQUE,
nome varchar (50) ,
sexo varchar (10) DEFAULT 'M',
endereco varchar (50) ,
idGerente int  ,
CONSTRAINT idFuncionario PRIMARY KEY(idFuncionario),
CONSTRAINT idGerente FOREIGN KEY (idGerente)REFERENCES Funcionario (idFuncionario));

CREATE TABLE Loja.Vendedor(
idVendedor int  NOT NULL UNIQUE,
NomeSetor varchar (20) ,
idFuncionario int  ,
CONSTRAINT idVendedor PRIMARY KEY(idVendedor),
CONSTRAINT idFuncionario1 FOREIGN KEY (idFuncionario)REFERENCES Funcionario (idFuncionario));

CREATE TABLE Loja.Atendente(
idAtendente int  NOT NULL UNIQUE,
nomeDepartamento varchar (50) ,
idfuncionario int  ,
CONSTRAINT idAtendente PRIMARY KEY(idAtendente),
```

```
CONSTRAINT idFuncionario2 FOREIGN KEY (idfuncionario)REFERENCES Funcionario (idFuncionario));

CREATE TABLE Loja.compra (
idProduto int  NOT NULL UNIQUE,
idCliente int  NOT NULL UNIQUE,
dataCompra varchar (10) ,
idCompra int  ,
CONSTRAINT idCompra PRIMARY KEY(idCompra),
CONSTRAINT idProduto FOREIGN KEY (idProduto) REFERENCES Produto (idProduto),
CONSTRAINT idCliente3 FOREIGN KEY (idCliente) REFERENCES Cliente (idCliente));

CREATE TABLE Loja.vende(
idVendedor int  NOT NULL UNIQUE,
idProduto int  NOT NULL UNIQUE,
dataVenda varchar (10) ,
idVenda int  ,
CONSTRAINT idVenda PRIMARY KEY(idVenda),
CONSTRAINT idProduto1 FOREIGN KEY (idProduto) REFERENCES Produto (idProduto),
CONSTRAINT idVendedor FOREIGN KEY (idVendedor) REFERENCES Vendedor (idVendedor));
```

Os resultados dos experimentos são mostrados baixo, bem como as modificações necessárias para que a modelo possa ser implementado no SGBD em questão sem problemas.

5.1 SGBD Firebird 2.1

Durante os experimentos feitos no SGBD Firebird, percebeu-se que o código gerado na ferramenta proposta possui incompatibilidades quanto ao fato de que não é necessário referenciar o banco de dados no create table, quanto à forma de declaração de campos do tipo double e quanto à forma de declaração de unicidade.

Tabela 1 - Incompatibilidades entre o código gerado pela ferramenta e o SGBD Firebird (Fonte: Os Autores).

Exemplo de código criado pela ferramenta	Exemplo de código aceito no Firebird
CREATE TABLE Loja.Produto	CREATE TABLE Produto
precoVenda double	precoVenda double precision
idProduto int UNIQUE	CONSTRAINT idProduto UNIQUE (idProduto)

As incompatibilidades encontradas acima foram solucionadas das seguintes maneiras:

- O nome do banco poderia ser retirado manualmente do código DDL antes da implementação do modelo no SGBD em questão.
- O tipo double poderia ser substituído manualmente, no momento da modelagem, pelo tipo double precision que também está presente entre as opções possíveis de tipos de dados da ferramenta proposta.
- A descrição do campo poderia ser feita através da ferramenta, porém defini-lo como único, dependeria da intervenção manual do usuário no código gerado.

Depois de feitas essas alterações, o modelo pôde ser implementado no SGBD Firebird sem problemas.

5.2 SGBD PostgreSQL 9.1

Durante os experimentos feitos no SGBD PostgreSQL, percebeu-se que o código gerado na ferramenta proposta possui incompatibilidades quanto à forma da declaração de criação das tabelas e quanto à forma de declaração de campos do tipo double.

Tabela 2 - Incompatibilidades entre o código gerado pela ferramenta e o SGBD PostgreSQL (Fonte: Os Autores).

Exemplo de código criado pela ferramenta	Exemplo de código aceito no PostgreSQL
CREATE TABLE Loja.Produto	CREATE TABLE Produto
precoVenda double	precoVenda double precision

As incompatibilidades encontradas acima foram solucionadas das seguintes maneiras:

- O nome do banco poderia ser retirado manualmente do código DDL antes da implementação do modelo no SGBD em questão.
- O tipo double poderia ser substituído manualmente, no momento da modelagem, pelo tipo double precision que também está presente entre as opções possíveis de tipos de dados da ferramenta proposta.

Depois de feitas essas alterações, o modelo pôde ser implementado no SGBD PostgreSQL sem problemas.

5.3 SGBD SQLServer 2012

Durante os experimentos feitos no SGBD SQLServer, percebeu-se que o código gerado na ferramenta proposta possui incompatibilidades quanto à forma da declaração de criação das tabelas.

Tabela 3 - Incompatibilidades entre o código gerado pela ferramenta e o SGBD SQLServer (Fonte: Os Autores).

Exemplo de código criado pela ferramenta	Exemplo de código aceito no SQLServer
CREATE TABLE Loja.Produto	CREATE TABLE Produto

A incompatibilidade encontrada acima foi solucionada da seguinte maneira:

- O nome do banco poderia ser retirado manualmente do código DDL antes da implementação do modelo no SGBD em questão.

Depois de feita essa alteração, o modelo pôde ser implementado no SGBD SQLServer sem problemas.

5.4 SGBD MySQL 5.2.39CE Revision 8757

Durante os experimentos feitos no SGBD MySQL, não foi encontrada nenhuma incompatibilidade quanto ao exemplo utilizado.

5.5 Limitações

Após a implementação e testes da ferramenta construída com base na abordagem adotada nesse trabalho, ficaram constatadas algumas limitações. Porém, algumas se devem pelo fato de que algumas questões a serem resolvidas não fazem parte do escopo desse trabalho e outras não estão relacionadas à abordagem adotada, mas a fatores externos que acabam influenciando na utilização da ferramenta implementada.

Não foi implementado nesse trabalho um mecanismo que auxilie o projetista da base de dados a não deixar de definir alguns elementos importantes para a modelagem como chaves primárias das entidades, a definição de qual atributo a chave primária ou estrangeira referencia, a definição de limitação de caracteres ou formato numé-

rico para determinado atributo ou a atribuição de valores para preencher determinado campo.

Não foi implementada, nas regras de transformação, uma forma de distinção que permita a utilização de palavras reservadas na nomeação de tabelas no código DDL gerado. Houveram tentativas dessa implementação no desenvolvimento das regras, porém foram encontradas incompatibilidades quando o código DDL gerado foi aplicado em alguns SGBD durante a fase de experimentos pelo fato de não seguirem o padrão ANSI/ISO a risca, o que causou a tomada da decisão de não fazer ainda essa implementação para que o cumprimento do prazo de entrega do trabalho não fosse comprometido.

Como exposto nos experimentos feitos do código DDL gerado nos distintos SGBD, apesar de ter sido estabelecido através das regras que o código gerado assumiria o formato de um núcleo comum dos padrões SQL 92, SQL 99 e SQL 2003 foi encontrada a necessidade de edição manual do código para que se tornasse compatível às características do SGBD em questão. Isso por que os SGBD acrescentam características próprias no código DDL para definição das estruturas das tabelas que não fazem parte do padrão ANSI SQL, o que dificulta a compatibilidade direta de um código DDL gerado de acordo com o padrão em relação aos distintos SGBD. Essa limitação não parte da abordagem adotada em si, mas sim pelo fato dos SGBD não aderirem completamente o padrão ANSI SQL para definição de dados.

6 Trabalhos Relacionados

Existem algumas ferramentas relacionadas com a modelagem de banco de dados que são usadas por causa da automação e praticidade oferecidas para auxiliar o processo de desenvolvimento, no entanto, algumas dessas ferramentas são pagas e ligadas à SGBD específicos.

O DBDesigner (DbDesigner homepage, 2012), apesar de não ser mais um projeto ativo, é uma ferramenta livre que possui modelagem gráfica, criação e manutenção de base de dados em um único ambiente, porém é limitada pelo fato de ser voltada exclusivamente para o SGBD MySQL.

O Oracle Designer (Oracle Designer, 2012) oferece opções de modelagem conceitual como o diagrama de ER, diagrama de funções, análise e design de objetos, mas é destinado para o Oracle DBMS, que é pago.

O CA Erwin (SLIK Software, 2012) é compatível com a maioria dos principais SGBD no mercado como MySQL, ODBC, Oracle, Postgree e SQL Server, porém tem como notação de modelagem conceitual a notação gráfica do IDEF1X e também é uma ferramenta paga.

O Toad Data Modeler (Quest Software, 2012) possibilita a criação e alteração de modelos de dados, criação de modelos de dados lógicos e físicos, comparação e sincronização de modelos, geração de SQL/DDL, criação e modificação de scripts e engenharia reversa e direta de sistemas de bancos de dados; é uma ferramenta paga.

O Database Design Tool (DDT) (GNU Win II homepage, 2012) é uma ferramenta *open source* que permite a criação de projetos de banco de dados, os quais podem ser desenhados e exportados para instruções SQL, também é possível importar descrições SQL e visualizá-las com o DDT.

O ModelRight (ModelRight, 2012) possibilita a visualização gráfica de bancos de dados, a aplicação de restrições complexas e o gerenciamento de exibições de bancos de dados, além da validação das decisões de designer e geração de scripits. É uma ferramenta paga e encontra-se diponível em seis versões diferentes as quais devem ser utilizadas de acordo com o SGBD escolhido.

7 Trabalhos Futuros

Em trabalhos futuros, pretende-se desenvolver um ambiente gráfico para as notações de modelagem aderentes à ferramenta, usando possivelmente o *Graphical Modeling Framework* (GMF), o que facilitará ainda mais o uso da ferramenta. Pretende-se construir representações gráficas dos elementos presentes nas notações de modelagem conceitual relacionais, o que poderá possibilitar a criação do modelo conceitual diretamente na ferramenta.

Os elementos que compõem a modelagem poderão ser criados respeitando os conceitos e características de cada notação, utilizando o metamodelo proposto nesse trabalho e poderão ser instanciados pelo usuário, de acordo com a notação de modelagem conceitual desejada. Uma vez instanciado o modelo desejado, poderá ser possível obter o código DDL referente ao mesmo, que poderá ser um código ainda não direcionado a um SGBD específico, mas visando o padrão ANSI/ISO.

Poderão ser implementadas extensões para resolver o problema das características específicas dos SGBD que impedem que em um primeiro momento o código gerado seja utilizado sem necessidade de edição nos SGBD distintos, facilitando assim ainda mais a implementação da modelagem feita.

Ainda pretende-se estudar como incorporar uma ferramenta de tecnologia de recuperação de informações para identificar os requisitos do sistema baseado numa análise textual e automaticamente instanciar o metamodelo proposto neste artigo. O usuário poderá especificar os requisitos do sistema através de um texto comum e a ferramenta irá interpretar esse texto e criar os mode-los, ou seja, a partir da descrição textual do sistema desejado, a ferramenta irá gerar automaticamente o modelos conceitual, depois disso, o modelo gráfico poderá ser autogerado assim como o código DDL no padrão ANSI SQL.

8 Conclusão

A proposta inicial desse trabalho era elaborar uma metodologia baseada na Arquitetura Orientada a Modelos voltada para modelagem de dados, com o propósito de contribuir com a resolução do problema de atrelamento a alguns SGBD específicos encontrado nas ferramentas existentes no mercado, o qual é causador de limitações quanto às opções de SGBD que as modelagens feitas utilizando essas ferramentas poderiam ser destinadas, estando muitas vezes limitadas a um único SGBD específico ou, nos casos que as ferramentas se mostram mais abrangentes, não serem de licença pública.

Para a resolução do problema detectado, foi proposta a definição de um metamodelo reunindo os conceitos de modelagem conceitual da notação Entidade-Relacionamento e o estabelecimento de regras de transformação de modelo para texto, utilizando a linguagem de transformação MOFM2T, que teriam, inicialmente, como objetivo transformar um modelo aderente ao metamodelo em código ANSI SQL DDL no padrão 92, aumentando assim a possibilidade de direcionar a modelagem feita utilizando da metodologia a SGBD que fossem compatíveis com o padrão SQL 92, não ficando assim mais restrita a utilização de um SGBD específico.

Contudo, durante a definição do metamodelo, enquanto se observava os conceitos presentes na notação de Entidade-Relacionamento, percebeu-se que, pelo fato de a metodologia adotada não tratar ainda da parte gráfica de representação da notação de modelagem conceitual, mas sim dos conceitos envolvidos na utilização da mesma, o escopo do projeto poderia ser facilmente estendido, pois existem vários pontos em comum entre a notação de Entidade-Relacionamento e outras notações que são utilizadas para modelagem de dados, voltadas para implementação em SGBD relacional como as notações de Crow's Foot, IDEF1X.

A existência de aspectos comuns entre as notações de modelagem conceitual torna a modelagem feita utilizando os conceitos dessas notações aderentes ao metamodelo definido, possibilitando a transformação dos modelos definidos nos seus respectivos códigos DDL. Essa possibilidade contribui para resolução do problema de atrelamento à uma notação de modelagem conceitual específica, o que também acontece com algumas das ferramentas existentes.

Em relação às regras de transformação, a princípio o foco era defini-las voltadas exclusivamente para o padrão SQL 92, por ser o mais antigo e consequentemente com os recursos mais comuns disponíveis entre os SGBD que são utilizados atualmente, mas também percebeu-se que poderiam ser feitas algumas modificações nes-

sas regras, adicionando, além dos recursos do SQL 92, alguns recursos compatíveis com as versões mais recentes, como a do SQL 99 e SQL 2003.

A metodologia adotada para desenvolvimento desse trabalho, a Arquitetura Orientada a Modelos, tem como uma de suas características a modularidade entre os diferentes tipos de níveis de abstração de modelos envolvidos e das regras de transformação que foram definidas, o que facilitou a implementação das mudanças feitas como a extensão do escopo do projeto em relação as demais notações de modelagem conceitual além da Entidade-Relacionamento durante o trabalho.

Caso seja necessário fazer alterações no metamodelo para implementar novos conceitos de modelagem ainda não abordados, seria necessário verificar qual seria o impacto da mudança sobre o mesmo e sobre as regras. Porém, a modularização presente na estrutura das regras facilita a modificação das já existentes. Sendo criadas novas regras, deverão ser verificadas em qual momento do processo de transformação elas se enquadram, fazendo a evocação das novas regras no momento oportuno por parte das já existentes.

Da mesma forma, caso se queira mudar a implementação das regras de transformação para direcionar a modelagem feita a um novo padrão adotado, não seria necessário mudar também a forma de definição de modelos estabelecida no metamodelo, pois ambas as partes se encontram em níveis diferentes de abstração. Ademais, as regras que são voltadas para implementação do modelo em uma plataforma específica e o metamodelo que é definido sobre um ponto de vista independente de plataforma.

Constatou-se que os principais SGBD utilizados não têm consenso na utilização de um padrão específico das versões existentes do ANSI-SQL, e também atribuem à modelagem relativa ao projeto físico, detalhes que tornam um código feito seguindo apenas um padrão ANSI-SQL adotado para DDL, sem levar em conta particularidades do SGBD em questão, incompatível em alguns pontos no momento de sua implementação, pois possuem algumas características próprias a serem satisfeitas.

Por fim, chegou-se à conclusão de que a ferramenta criada para realizar a transformação de modelos para códigos DDL com base na metodologia proposta nesse trabalho facilita o trabalho do projetista de base de dados por dar a ele poder de escolha em relação a notação de modelagem conceitual que pode ser adotada, e também, por gerar um código DDL seguindo o padrão ANSI/ISO , salvo pequenas edições manuais necessárias no momento que variam de acordo com o SGBD que se deseja implementar a base de dados.

9 Bibliografia

(OMG), O. M. G. MOF Model to Text Transformation Language 1.0. 2008.

ABREU, M.; MACHADO, F. N. R. Projeto de Banco de Dados: Uma Visão Prática. Erica, 1999.

Acceleo: MDA generator - Home. .Disponível em: <http://www.acceleo.org/pages/home/en>. Acesso em: 9/10/2012.

BRUCE, T. A. Designing quality databases with IDEF1X information models. Dorset House Pub., 1992.

CHEN, P. P.-S. The entity-relationship model—toward a unified view of data. ACM Trans. Database Syst., v. 1, n. 1, p. 9–36, 1976.

CODD, E. F. A relational model of data for large shared data banks. Commun. ACM, v. 13, n. 6, p. 377–387, 1970.

Database Design Tool, GNU Win homepage. Disponível em: < http://gnuwin.epfl.ch/apps/DDT/en/>. Acesso em 21/11/2012.

DbDesigner homepage. .Disponível em: <http://dbdesigner.sourceforge.net/>. Acesso em: 10/6/2012.

DBExplorer. .Disponível em: <http://dbexplorer.com/>. Acesso em: 3/9/2012.
EISENBERG, A.; MELTON, J. SQL: 1999, formerly known as SQL3. SIGMOD Rec., v. 28, n. 1, p. 131–138, 1999.

GONÇALVES, I. S.; ROSA, A. S.; PANTOJA, C. E.; LAZARIN, N. M. Uma metodologia para modelagem conceitual de banco de dados integrada utilizando orientação a modelos. In: VI FECTI- Feira de Ciência, Tecnologia e Inovação do Estado do Rio de Janeiro ,2012, Rio de Janeiro, Brasil. (Aceito)

GUEDES, G. T. A. UML - Uma Abordagem Prática. Novatec, 2008.

HALPIN, T. Entity Relationship Modeling from an ORM perspective: Part 1. ,1999.

HAY, D. C. A comparison of Data Modeling Techniques. Essential Strategies. ,1999. Disponível em: <http://essentialstrategies.com/documents/comparison.pdf>. .
HEUSER, C. A. Projeto de Banco de Dados. 6º ed. Bookman, 2009.

JACOBS, R. D.; CARVALHO, J. V. DDL, Lidando com as diferenças das instruções SQL nos diferentes SGBD's. Disponível em: <http://www.sirc.unifra.br/artigos2006/SIRC-Artigo25.pdf>. Acesso em: 9/10/2012.
Logical Versus Physical Database Modeling. .Disponível em: <http://www.developer.com/tech/article.php/641521/Logical-Versus-Physical-Database-Modeling.htm>. Acesso em: 9/10/2012.

MELLOR, S. J.; SCOTT, K.; UHL, A.; WEISE, D. MDA Destilada: Princípios de Arquitetura Orientada por Modelos. Ciência Moderna Ltda, 2005.
ModelRight homepage. Disponível em: < http://www.modelright.com/ >. Acesso em: 21/11/2012.

OMG. Model Driven Architecture (MDA) Guide. 2003.

Oracle Designer. .Disponível em: <http://www.oracle.com/technetwork/developer-tools/designer/overview/index-082236.html>. Acesso em: 19/9/2012.
Physical Data Model. .Disponível em: <http://www.1keydata.com/datawarehousing/physical-data-model.html>. Acesso em: 9/10/2012.

ROSA, A., GONÇALVES, I. and PANTOJA, C.E. A MDA Approach for Database Modeling. Lecture Notes on Software Engineering, v. 1, n. 1, p. 26-30, 2013.

ROSA, A.S., GONÇALVES, I.S., MORI, N., PANTOJA, C.E. Uma abordagem orientada a modelos para modelagem conceitual de banco de dados. In: Anais Sulcomp. Congresso Sul Brasileiro de Computação 2012, Criciúma, Santa Catarina, Brasil.

SLIK Software. .Disponível em: <http://www.sliksoftware.co.nz/products/dbexplorer/index.htm>. Acesso em: 3/9/2012.

STEINBERG, D.; BUDINSKY, F.; MERKS, E.; PATERNOSTRO, M. Emf: Eclipse Modeling Framework. Pearson Education, 2008.
Visual Database Tools. .Disponível em: <http://msdn.microsoft.com/pt-br/library/y5a4ezk9.aspx>. Acesso em: 3/9/2012.

Toad Data Modeler, Quest Software. Disponível em: < http://www.quest.com/toad-data-modeler/> . Acesso em: 21/11/2012.

Apêndice 1

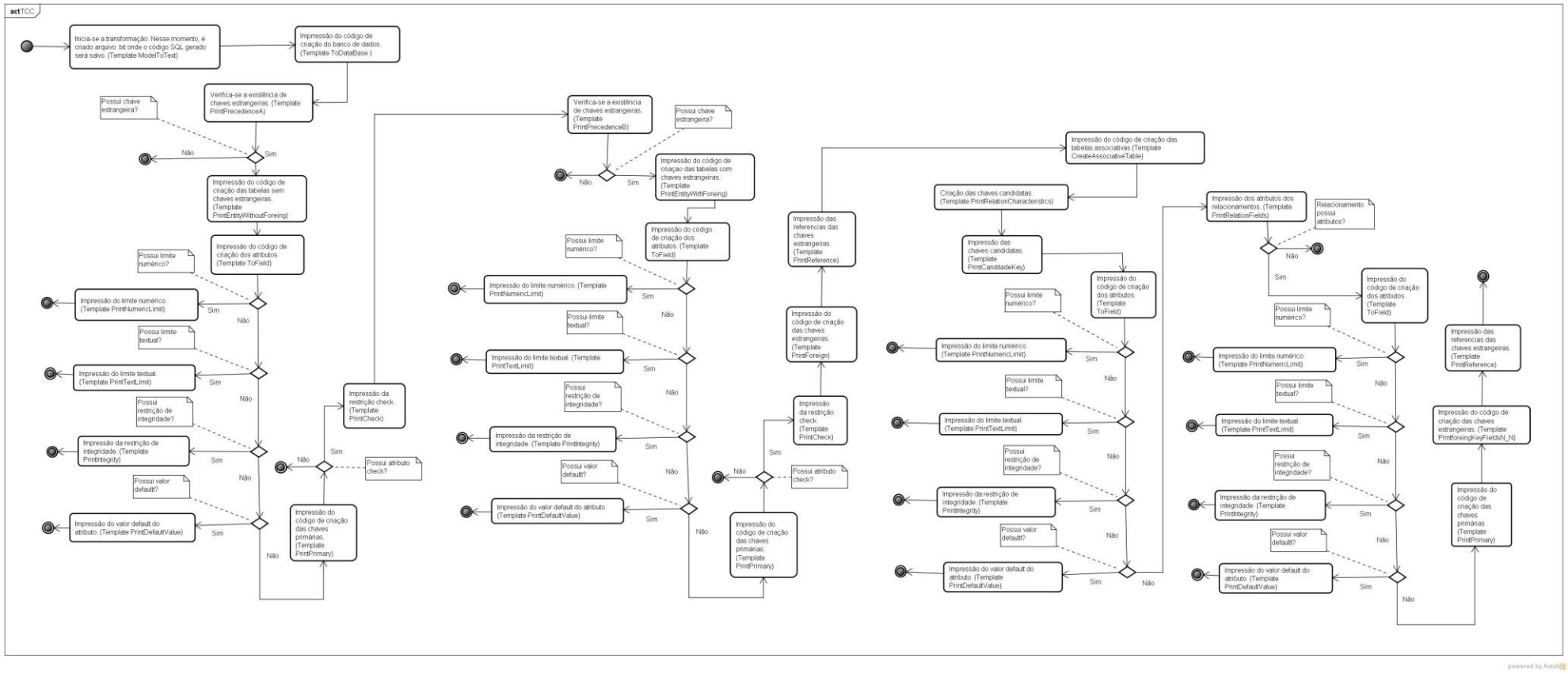

Figura 80 - Diagrama de Atividades