Meta modelagem o Modelo de Entidade-Relacionamento Aprimorado

Resumo.

Um metamodelo oferece uma sintaxe abstracta para distinguir entre os modelos válidos e inválidos. Isto é, um metamodel é tão útil para uma linguagem de modelação de uma gramática é para uma linguagem de programação. Neste contexto, embora a Entidade-Relacionamento (EER) Modelo avançado é o "de facto" linguagem de modelagem padrão para banco de dados de projeto conceitual, com o melhor de nosso conhecimento, existem apenas duas propostas de metamodelos TCE, que não fornecem uma completa apoio para a notação de Chen. Além disso, nem uma discussão sobre a engenharia utilizada para especificar essas metamodelos é apresentado nem uma análise comparativa entre eles é feita. Com o objetivo de superar esses inconvenientes, vamos mostrar uma visão detalhada e prática de como formalizar o modelo EER por meio de um metamodelo que (i) cobre todos os elementos da notação da Chen, (ii) define regras bem formação correcta necessários para a criação de sintaticamente esquemas TCE corretas, e (iii) pode ser usado como um ponto de partida para criar Computer Aided Software Engineering (Case) ferramentas para modelagem EER, metadados intercâmbio entre essas ferramentas, execute SQL automático / DDL geração de código, e / ou ampliar (ou reutilização parte) o modelo EER. A fim de mostrar a viabilidade, expressividade e utilidade do nosso metamodelo (EERMM nomeado), desenvolvemos uma ferramenta CASE (EERCASE chamado), que foi testado com um exemplo prático que abrange todos os construtores da TCE, confirmando que o nosso metamodelo é viável , útil, mais expressiva do que aqueles relacionados e correctamente definida. Além disso, analisamos o nosso trabalho contra os relacionados e apresentar nossas considerações finais.

1.INTRODUÇÃO

Linguagens de modelagem são usados para criar modelos que visam elevar o nível de abstração e ocultar detalhes de implementação. A fim de evitar modelos inválidos, a sintaxe de uma linguagem de modelagem deve definir regras de boa formação que especificam o que é um modelo válido. Neste contexto, é importante ressaltar que a sintaxe abstrata de uma linguagem de modelagem é formalizada por meio de um metamodelo, que também serve como uma base para o intercâmbio de modelos com outras ferramentas. Por esta razão, o termo "meta" é utilizado porque a especificação de linguagem de modelação é um nível mais elevado do que os modelos usuais. Em sua forma mais simples, podemos dizer que um metamodelo é um modelo conceitual de uma linguagem de modelagem. Ou seja, ele descreve os conceitos de uma linguagem de modelagem, as suas propriedades e as ligações jurídicas entre elementos de linguagem. Portanto, metamodelos são artefatos obrigatórios e importantes na especificação de linguagens de modelagem. Em outras palavras, metamodelos são entidades fundamentais para designers e desenvolvedores de ferramentas de modelo, porque precisamente definir como ferramentas e modelos devem trabalhar em conjunto [Kelly e Tolvanen de 2008].

et ai. 2008; Combi et ai. 2008]; (Ii) há uma numerosa Computer Aided Software Engineering (CASE) ferramentas que visam apoiar a modelagem EER (por exemplo, EER / Studio XE2, PowerDesigner, ERWin e SmartDraw); (Iii) os currículos mais ciência da computação da universidade inclui-lo [ACM / IEEE 2008], e (iv) a maioria dos livros de banco de dados [Elmasri e Navathe 2010; Silberschatz et al. 2010; Connolly e Begg 2009; Garcia-Molina et ai. 2008] dedicar um ou mais capítulos de apresentá-la. No entanto, com o melhor de nosso conhecimento, não há metamodelos que fornecem suporte completo para a notação de Chen (incluindo suas extensões - cf. Figura 1). Neste contexto, uma vez que (i) um metamodelo é como uma gramática para uma linguagem de modelagem, (ii) o modelo EER é o padrão "de facto" para banco de dados modelagem conceitual, e (iii) não há metamodel EER que cobre efetivamente tudo construtores de

notação da Chen (o mais utilizado notação EER), argumentamos que a especificação de um metamodelo EER é tão útil para a comunidade de banco de dados como a especificação do metamodelo UML é para a comunidade de engenharia de software.

Com o objetivo de superar a lacuna anterior, neste artigo, apresentamos uma visão detalhada e prática de como formalizar o modelo EER por meio de um metamodelo que (i) suporta todos os construtores de notação de Chen, (ii) define regras de boa formação que impedir a concepção de esquemas TCE inválidos, e (iii) pode ser usado como um ponto de partida para (a) criar ferramentas CASE EER, (b) metadados intercâmbio entre essas ferramentas, (c) permitir que o SQL automático / DDL geração de código, e / ou (d) estender (ou reutilizar parte do) Modelo EER para abordar novos recursos (por exemplo, espacial e / ou modelagem temporal).

O restante deste artigo é organizado da seguinte forma. Na Seção 2, Escribe brevemente as principais linguagens de modelagem para o projeto conceitual de bancos de dados. A seguir, na Seção 3, discutimos os trabalhos conexos. Depois disso, na Seção 4, apresentamos como o nosso metamodelo foi definida, as suas regras de boa formação e uma análise comparativa entre o nosso trabalho e os mais importantes. Por sua vez, na Seção 5 mostramos uma idéia geral de como o nosso metamodelo foi usado para construir uma ferramenta CASE EER e, usando esta ferramenta, nós modelamos um exemplo prático que abrange todos os construtores da TCE. Finalmente, na secção 6 nós fornecemos algumas conclusões e indicações de trabalhos futuros.

2. linguagens de modelagem para o desenho conceitual DE BASES DE DADOS

Existem muitas notações que podem ser utilizados para a concepção da base de dados [Song et al. 1995]. Alguns deles permitir relações binárias, no máximo, e por este motivo são conhecidos como notações binários, enquanto outros permitem que as relações n-árias e são conhecidos como as notações n-ário. Em geral, a maioria das ferramentas CASE usar notações binários. No entanto, estas notações (i) não permitem atributos em relacionamentos - o que não parece natural, porque os relacionamentos podem ter propriedades descritivas e (ii) não permitem relações entre três ou mais entidades - que também é inconveniente, porque alguns relacionamentos são intrinsecamente n-ária e, embora haja transformações de uma relação n-ária de relacionamentos n-binário, o resultado muitas vezes tem semânticas diferentes, a menos que acompanhadas de restrições de integridade [Hartmann 2003; Jones e Canção de 1996; Song et al. 1995]. Por estas razões, notações n-árias são semântica e conceitualmente mais expressiva do que as notações binários [Song et al. 1995], e são mais frequentemente usados na academia. Independentemente de a notação usada, existem duas convenções diferentes para se referir ao lugar onde papéis, cardinalities, e participações são especificados em um relacionamento, ou seja, "look-do outro lado "(ou seja, do lado oposto da entidade) e" olhar-aqui "(ou seja, do mesmo lado da entidade) [Hartmann 2003] [Song et al. 1995].

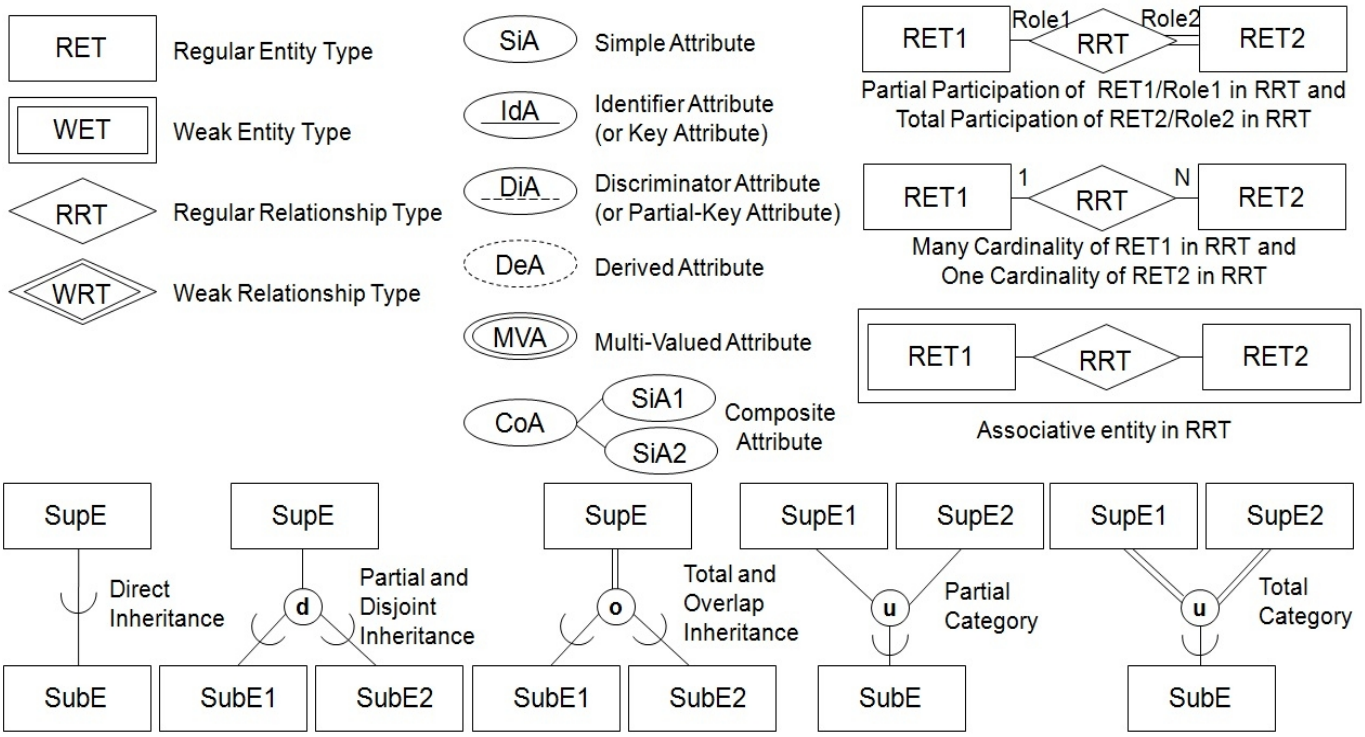
Entre as notações para o desenho conceitual de um banco de dados, o modelo EER eo diagrama de classes (CD) da Unified Modeling Language (UML) [Rumbaugh et al. 2004], são os mais utilizados. Recentemente, três estudos de investigação [Lucia et al. 2008] [Lúcia et ai. 2010] [Bavota et ai. 2011] realizar comparações entre o modelo EER eo CD e relatar os resultados interessantes. Lúcia et ai. [2008] e Lucia et al. [2010] sugerem que a notação CD é geralmente mais compreensível do que a notação EER.

No entanto, Bavota et ai. [2011] concluir que o entendimento do EER Modelo é significativamente superior ao CD se "atributo composto", "atributo multi-valor", e "entidades fraco" são necessárias em um determinado modelo. Ressaltamos que os autores ainda não têm levado em conta tais construções como "entidades associativas", "atributos nas relações", eo "imprecisão de olhar em frente-notações (tais como CD)" que não representam efetivamente a semântica de participação restrição sobre as relações com grau superior a dois [Hartmann 2003; Dullea et ai. 2003; Song et al.

1995]. Portanto alguns construtores TCE importantes ainda precisam ser avaliados. Esperamos que, se a avaliação é realizada tendo em conta todo o conjunto de construtores, mais vantagens para o modelo EER são revelados. Para resumir esta discussão, nós pensamos que qualquer comparação entre o modelo EER eo CD pode fornecer resultados para debate e discussão ea escolha de uma notação especial, em vez de outro é uma questão de preferência ou de contingência do projeto.

3. Trabalhos Relacionados

Apesar das fraquezas de CD (cf. secção 2), muitos vendedores UML manifestaram o desejo de usar o CD para modelagem de dados e acabou definindo seus próprios perfis específicas da ferramenta para cada um. Como resultado, não há nem um padrão aceito nem interoperabilidade dos modelos desenvolvidos usando esses perfis / ferramentas [OMG 2003]. Com o objetivo de superar essa limitação, o Object Management Group (OMG) está revisando sua especificação chamado Common Warehouse Metamodel (CWM) [OMG 2001], a fim de fornecer mais metamodelos normativos (incluindo um metamodelo para o modelo EER) chamado Information Management Metamodel (IMM) [OMG 2009]. Mostramos na Fig. 2 (com base na OMG [2001]) e A Fig. 3 (com base na OMG [2009]) os metamodelos EER de CWM e IMM, respectivamente.

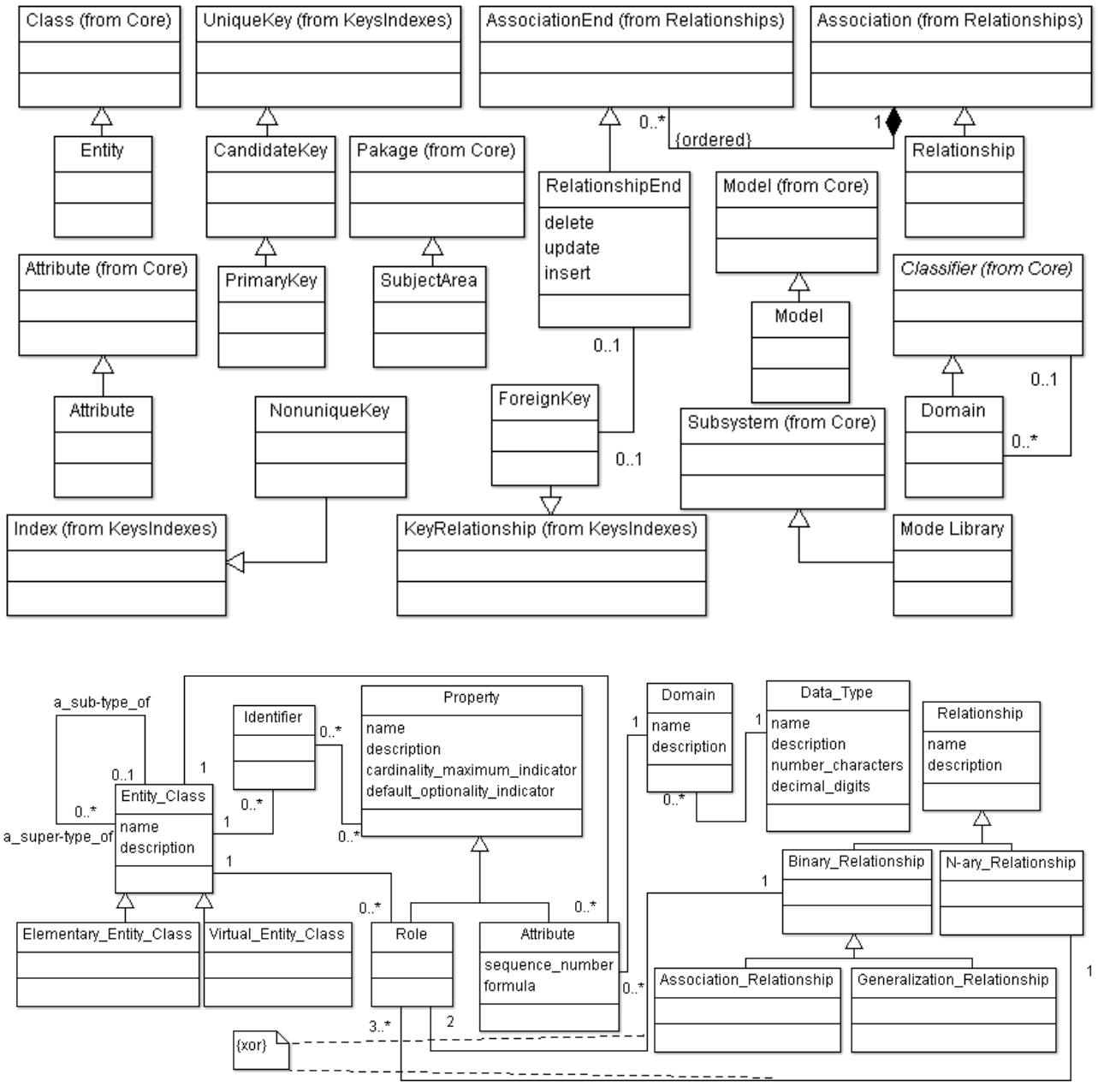


Na Fig. 2, o metamodelo CWM estende o metamodelo UML e especifica as metaclasses para os principais conceitos do modelo EER (ou seja, Entidade, relacionamento e Atributo). Considerando os construtores EER mostrados na Fig. 1, pode-se notar que o metamodelo CWM não inclui metaclasses nem meta-atributos para Weak Entidade / Relacionamento, Atributo Composite, Discriminator Atributo, e Categoria. Além disso, vale ressaltar que os construtores Associativo entidade, atributo de valores múltiplos, atributo derivado, e relacionamento com atributo, embora não diretamente representadas na figura 2, pode ser mapeada usando a notação UML. Ou seja: (i) uma entidade associativa pode ser representado como uma classe de associação; (Ii) um atributo com valores múltiplos podem ser representadas usando parênteses "[]"; (Iii) um atributo derivado pode ser representado utilizando uma barra inicial "/", e (iv) um atributo em relação também pode ser representado utilizando uma classe de associação.
Os outros construtores / construções do modelo EER, ou seja, Relacionamento Unário, relacionamento Cardinality, Participação do relacionamento, relacionamento N-ary, Papel, herança

simples, herança múltipla, Total / Parcial herança, e Disjoint / Sobreposição de herança, embora não representados na Fig . 2, que também pode ser capturado por CWM, uma vez que estes construtores / construções são definidos no metamodelo UML. É importante ressaltar que o CD é uma linguagem de propósito geral. Portanto uma extensão do metamodelo UML para o projeto de banco de dados requer a definição de algumas restrições em Object Constraint Language (OCL) para suprimir elementos UML que estão fora do contexto de banco de dados (ou seja, elementos desnecessários que podem introduzir erros) e assegurar regras de boa formação. No entanto, nenhuma restrição OCL está especificado na metamodel EER de CWM, que é uma limitação importante. Ainda mais, o metamodelo CWM define o metaclass ForeignKey, que é um construtor para modelagem de banco de dados relacional. Ou seja, este é um outro ponto negativo, porque este metaclass não é um construtor do modelo EER.

O metamodelo EER do IMM (ver Fig. 3), ao contrário do metamodelo CWM, nem se estende metaclasses UML para a especificação de seu metamodelo nem se mistura com os construtores da TCE de banco de dados relacionais.

No entanto, tendo em conta também os construtores TCE apresentados na Fig. 1, o IMM não especifica metaclasses, meta-atributos, ou construções para Associativo Entidade, Categoria, Discriminator Atributo, Atributo Composite, atributo de valores múltiplos, Atributo no relacionamento, Especialização



Relacionamento, e herança múltipla. Portanto, ainda não há um metamodelo que fornece um suporte completo para a notação de Chen.

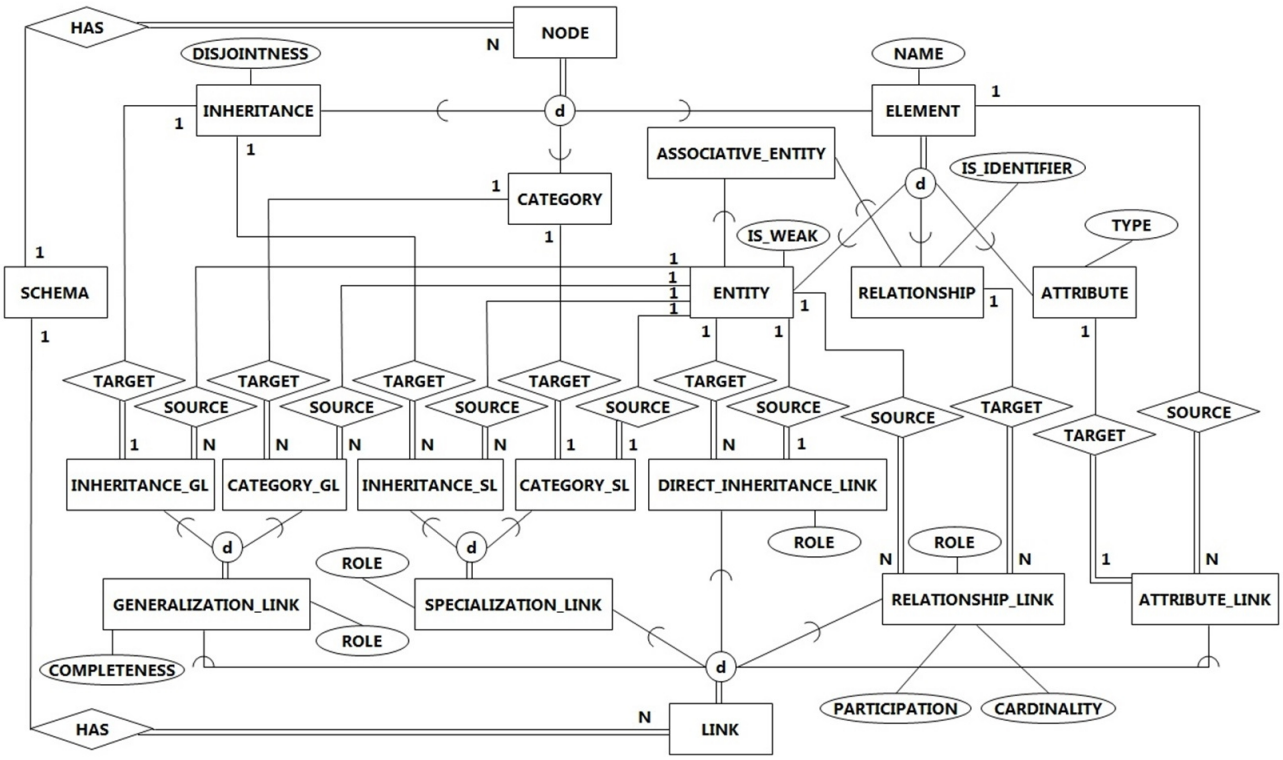
4. metamodelagem DO MODELO EER

O modelo EER é um diagrama node-link [Irani et al. 2001; Ware e Bobrow de 2004], onde nós representam Entidades, Entidades Associativas, atributos, relações, heranças, e categorias, ao passo que as ligações denotar Atributo Link, Relacionamento Link, Generalização Link, e Especialização Link (ver a parte inferior da Fig. 4). Então, com a abstração nó-link ea notação EER em mente, uma visão conceitual do nosso metamodelo, chamado reforçada Entidade-Relacionamento MetaModel (EERMM), é apresentada na figura 4 (ver a parte superior).

De acordo com a Figura 4, nosso metamodelo tem três principais meta-entidades: Esquema, Nó e Link. Schema é a meta-entidade raiz que corresponde à área de desenho de um esquema EER. Por esta razão, esquema pode ter muitas instâncias de Node e muitos casos de Link, que não pode existir (participação total) sem um esquema. Além dos principais meta-entidades, nosso metamodelo tem três meta-entidades especializadas para Node: Elemento, herança, e da categoria, que cobrem os principais construtores do modelo EER.

A meta-Elemento entidade tem um nome chamado meta-atributo (ou seja, uma etiqueta) e é especializada em três meta-entidades: Entidade, relacionamento, e Atributo, que são usados para capturar conceitos do mundo real, como esses conceitos estão ligados uns aos outros, e as propriedades destes conceitos / relações, respectivamente. Note-se que (i) a meta-entidade Entidade tem uma meta-atributo chamado is_Weak que é um valor booleano (ou seja, is_Weak = "True" define uma entidade fraca e is_Weak = "false", define uma entidade regular); (Ii) a relação meta-entidade tem uma meta-atributo chamado is_Identifier que é também um booleano valor (ou seja, is_Identifier = "True" define uma relação de identificação e is_Identifier = "false", define uma relação regular); (Iii) o atributo meta-entidade tem uma meta-atributo chamado tipo que especifica se uma instância de atributo é "comum", "identificador", "discriminador", "derivado" ou "valores múltiplos", exclusivamente; e (iv) o meta-entidade Associative_Entity é tanto uma entidade e um Relacionamento (ou seja, é uma especialização destas meta-entidades).

Na seqüência, o Herança meta-entidade capta o conceito de "herança" e tem a meta-atributo dis-operações combinadas, que define se uma herança é mutuamente exclusiva ("disjunção") ou não ("sobreposição"). Por fim, a Categoria meta-entidade capta o conceito de "categoria" (também chamado de "tipo de união"). Note-se que, por definição [Elmasri e Navathe 2010], uma categoria é separado. Então, em nosso metamodelo nós não modelar a meta-atributo disjunção para uma categoria.



	EER CONSTRUCTORS	EER META-ENTITIES
Nodes		ENTITY
		RELATIONSHIP
		ATTRIBUTE
		ASSOCIATIVE_ENTITY
	d o	INHERITANCE
	u	CATEGORY
Links	1 N 1 N	RELATIONSHIP_LINK
		ATTRIBUTE_LINK
		INHERITANCE_SL
		CATEGORY_SL
		INHERITANCE_GL
		CATEGORY_GL
		DIRECT_INHERITANCE_LINK

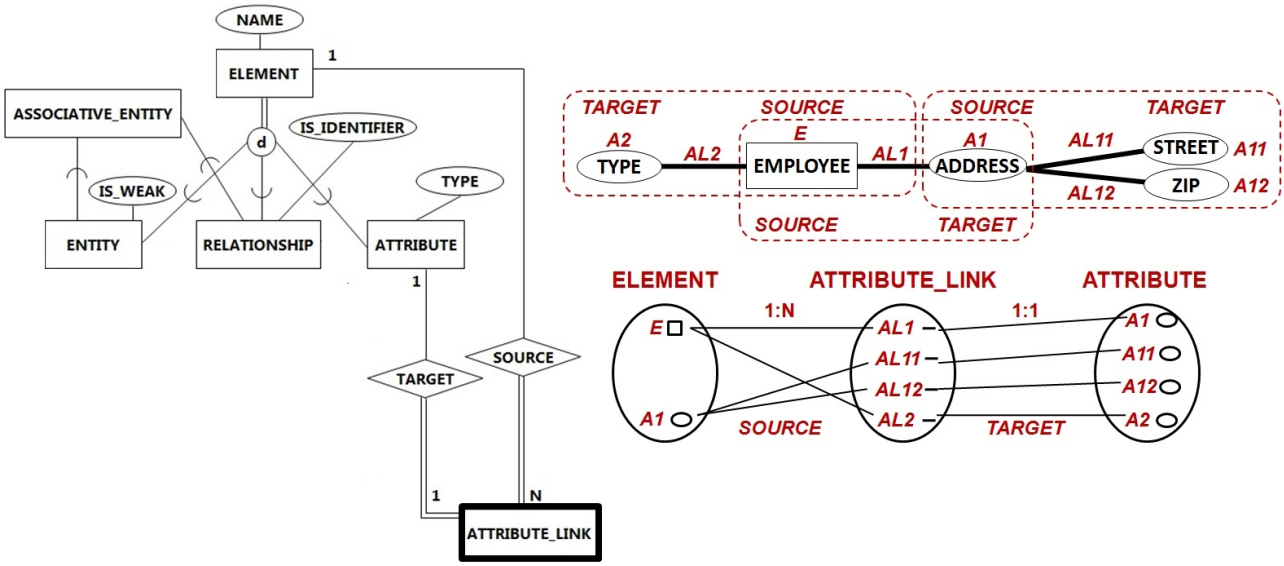
Além das Nó meta-entidades, nosso metamodelo tem cinco meta-entidades especializadas para Link, nomeadamente Attribute_Link, Relationship_Link, pecialization_Link, Generalization_Link, e Direct_Inheritance_Link, que tratam das ligações para um atributo, um relacionamento, uma especialização, uma generalização, e uma direta herança, respectivamente. Estes meta-entidades são apresentados a seguir:

A meta-Attribute_Link entidade tem uma relação de origem com o meta-entidade elemento de tal forma que uma instância de Attribute_Link tem apenas uma instância de elemento como fonte, mas uma instância de elemento pode ser fonte de muitos casos de Attribute_Link. Além disso, a meta-Attribute_Link entidade tem uma relação alvo com o atributo meta-entidade de tal forma que uma instância de Attribute_Link tem apenas uma instância de Atributo como alvo e uma instância de atributo pode ser alvo de apenas uma instância de Attribute_Link. Ou seja, uma instância de elemento pode ser fonte de muitos casos de Attribute_Link (ou seja, muitos atributos em entidades,

muitos atributos em relacionamentos e muitos atributos compostos), mas uma instância de atributo pode ser alvo de apenas uma instância de Attribute_Link (ou seja, um atributo não pode ser vinculado com mais de um elemento). A fim de esclarecer essa discussão, na Fig.5 vamos nos concentrar no Attribute_Link (representado na linha de espessura) e, nesta figura, apresenta-se parte do nosso metamodelo que aborda esse tipo de link e um fragmento de um esquema EER com a sua diagrama que ilustra ocorrência cardinalidades das relações origem e de destino. A meta-Relationship_Link entidade também tem uma relação de origem com a Entidade meta-entidade.

Nesta relação uma instância de Relationship_Link tem apenas uma instância de Entidade como fonte, mas uma instância de entidade pode ser fonte de muitos casos de Relationship_Link. Além disso, a meta-Relationship_Link entidade tem uma relação com o alvo de Relacionamento meta-entidade para que uma instância de Relationship_Link tem apenas uma instância de relacionamento como alvo. No entanto, uma instância de relacionamento pode ser alvo de muitos casos de Relationship_Link. Isto é, um exemplo de Relationship_Link conecta um exemplo de Entidade com um exemplo de relação, mas um exemplo de entidade ou um exemplo de relação pode ser ligado a muitos casos de Relationship_Link (ou seja, uma entidade pode ser ligada a várias relações e um relacionamento pode ser n-ário). Além disso, a meta-Relationship_Link entidade tem três meta-atributos: (i) a participação - que determina se todos (ou seja, "total") ou apenas alguns (ou seja, "parcial") instâncias de uma entidade vai participar do relacionamento; (Ii) cardinalidade - que especifica se o número máximo de instâncias de uma entidade em um relacionamento é "um" ou "muitos"; e (iii) papel - é um texto que pode ser usado para descrever a função de uma entidade de uma relação. Na Fig. 6 nos concentramos em Relationship_Link (representado na linha de espessura). Nesta figura é mostrada a parte do nosso metamodel que cobrem o meta-entidade Relationship_Link, bem como uma parte de um esquema EER e o seu diagrama de ocorrência que exemplificam os cardinais de as relações de fonte e de destino. Outra meta-entidade de nosso metamodelo é Specialization_Link. Esta meta-entidade tenha a função de meta- atributo que corresponde a um texto para descrever a função de um sub-entidade numa especialização.

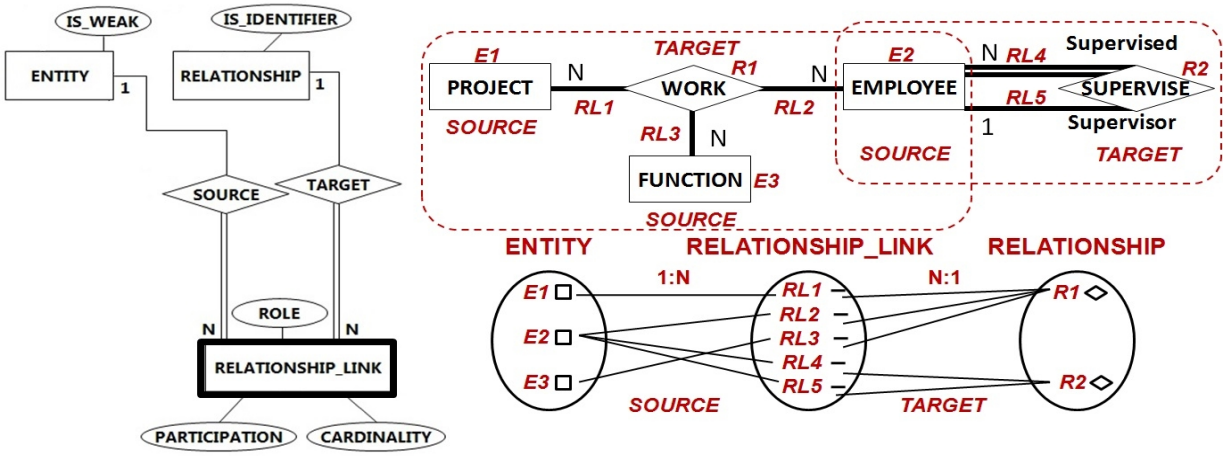
Specialization_Link é especializada em duas meta-entidades: Inheritance_SL (Herança Especialização link) e Category_SL (Categoria Especialização Link), que são usados para definir um link especialização entre um sub-entidade e uma herança ou entre um sub-entidade e uma categoria, respectivamente .Note-se que: (i) um exemplo de Inheritance_SL Category_SL ou tem apenas um exemplo de como fonte de entidade. No entanto, enquanto um exemplo de entidade pode ser fonte de muitos casos de Inheritance_SL (isto é, um sub-entidade pode ser uma especialização de muitas heranças - uma herança múltipla), uma instância

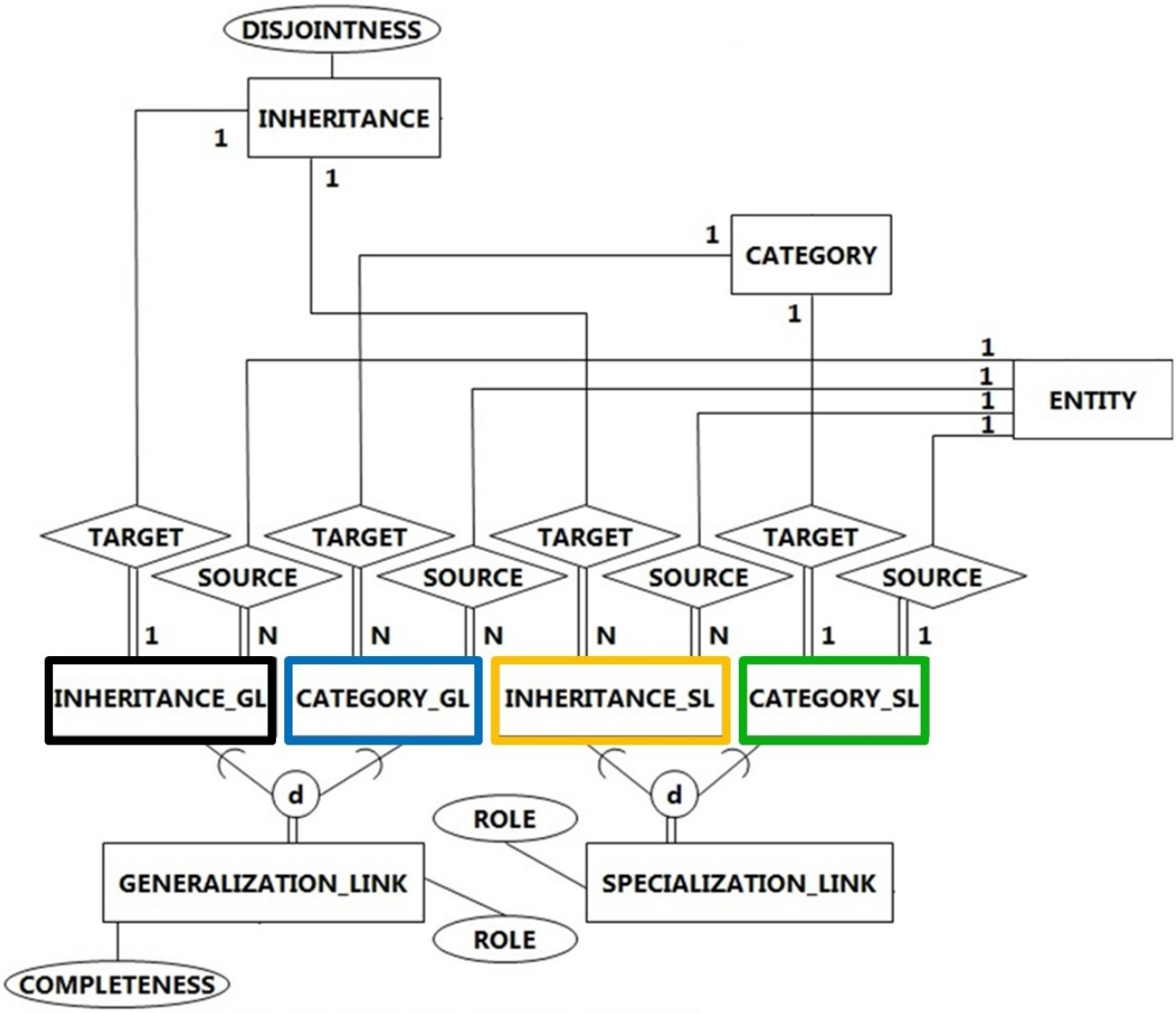


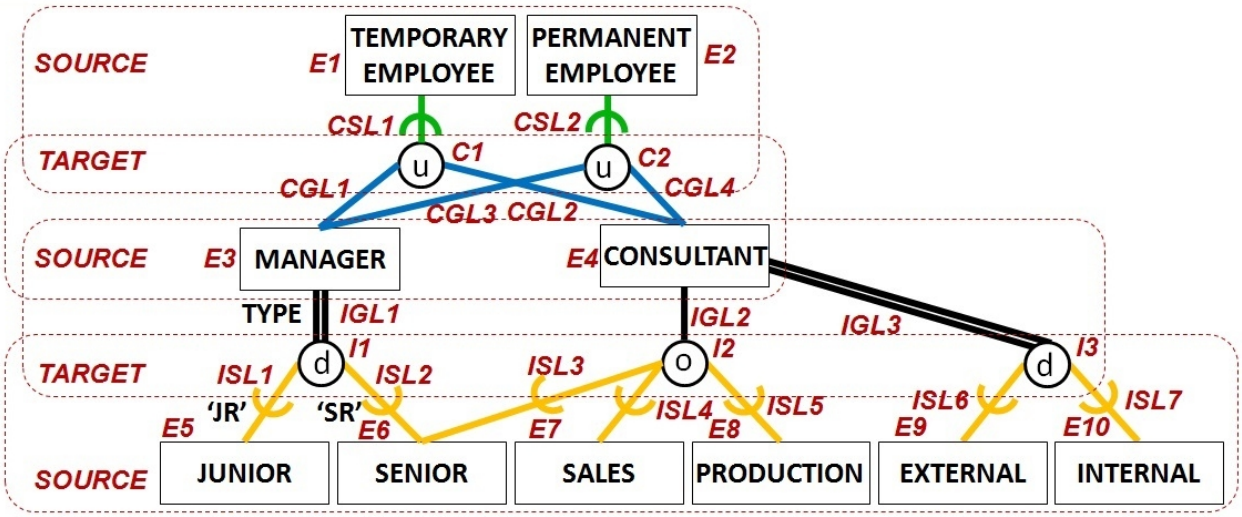
de entidade pode ser fonte para apenas uma instância de Category_SL (ou seja, um sub-entidade pode ser uma especialização de uma única categoria) e (ii) uma instância de Inheritance_SL ou Category_SL tem apenas uma instância de herança ou da categoria como alvo, respectivamente. No entanto, apesar de uma instância de herança pode ser alvo de muitos casos de Inheritance_SL (ou seja, uma herança simples com muitos sub-entidades), uma instância de Categoria, por definição [Elmasri e Navathe 2010], pode ser alvo de apenas uma instância de Category_SL (ie, uma categoria só pode ter um sub-entidade).

Da mesma forma para a Specialization_Link, o meta-entidade Generalization_Link tem também a função de meta-atributo correspondente a um texto que pode ser usado para descrever a função de um super-entidade em uma generalização. Além disso, Generalization_Link tem a integralidade meta-atributo, que especi-fies se a generalização é completamente definido ("total") ou não ("parcial"). Generalization_Link é especializada em duas meta-entidades: Inheritance_GL (Generalização Herança link) e Category_GL (Categoria Generalização Link), que são usados para definir um link de generalização entre uma super-entidade e uma herança, ou entre um super-entidade e uma categoria, respectivamente. Note-se que: (i) um exemplo de Inheritance_GL Category_GL ou tem apenas um exemplo de como fonte de entidade, mas um exemplo de En-dade pode ser fonte de muitos casos de Inheritance_GL (ou seja, uma super-entidade pode ser uma generalização de muitas heranças) ou Category_GL (ou seja, um super-entidade pode participar de muitas categorias) e (ii) uma instância de Inheritance_GL ou Category_GL tem apenas uma instância de herança ou da categoria como alvo, respectivamente. No entanto, enquanto uma instância de herança pode ser alvo de apenas uma instância de Inheritance_GL (ou seja, uma herança pode ter apenas um super-entidade), uma instância de Categoria pode ser alvo de muitos instância de Category_GL (ie, uma categoria pode ter muitos super-entidades. Na verdade, ele deve ter, pelo menos, dois super-entidades). Semelhante às duas últimas figuras, na fig. 7 exibimos uma fração do nosso metamodelo e um pedaço de um esquema EER com seus quatro diagramas de ocorrência que ilustram as cardinalidades das relações origem e de destino. Na Fig. 7 estamos nos concentrando em Inheritance_SL, Category_SL, Inheritance_GL e Category_GL (retratado em linhas grossas e usando cores diferentes). Diferentemente do Inheritance_SL eo Inheritance_GL, a meta-entidade Direct_Inheritance_Link conecta duas entidades sem usar um nó herança. A Direct_Inheritance_Link é impor-tante, porque às vezes é necessário para modelar uma herança múltipla de tal forma que um sub-entidade herda diretamente de dois ou mais super-entidades ou modelar uma herança com apenas um sub-entidade. Note-se que, em ambos os casos é um erro para definir as propriedades disjunção e integralidade de uma herança com apenas um sub-entidade. Por esta razão, um Direct_Inheritance_Link não têm essas propriedades.

No que respeita às relações origem e de destino, o meta-Direct_Inheritance_Link entidade tem uma relação de fonte com a Entidade meta-entidade de tal forma que uma instância de Direct_Inheritance_Link tem apenas uma instância de Entidade como fonte e uma instância de entidade pode ser fonte para apenas uma instância do Di -

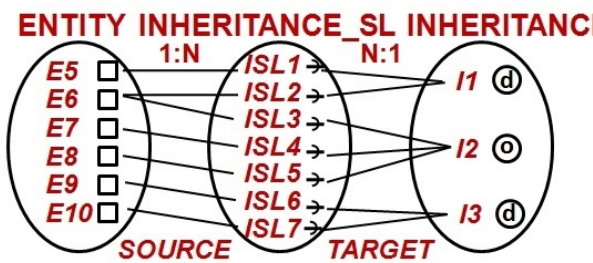


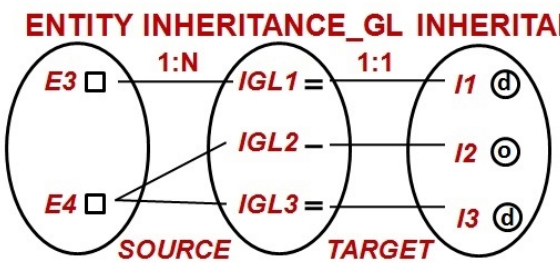

DISJOINTNESS
INHERITANCE
CATEGORY
ENTITY
TARGET
SOURCE
INHERITANCE_GL
CATEGORY_GL
INHERITANCE_SL
CATEGORY_SL
d
ROLE
GENERALIZATION_LINK
SPECIALIZATION_LINK
COMPLETENESS


SOURCE
TARGET
TEMPORARY EMPLOYEE
PERMANENT EMPLOYEE
E1
E2
CSL1
CSL2
C1
C2
CGL1
CGL2
CGL3
CGL4
MANAGER
CONSULTANT
E3
E4
TYPE
IGL1
IGL2
IGL3
I1
I2
I3
ISL1
ISL2
ISL3
ISL4
ISL5
ISL6
ISL7
'JR'
'SR'
E5
E6
E7
E8
E9
E10
JUNIOR
SENIOR
SALES
PRODUCTION
EXTERNAL
INTERNAL


ENTITY
CATEGORY_SL
CATEGORY
1:1
1:1
E1
E2
CSL1
CSL2
C1
C2
SOURCE
TARGET


ENTITY
CATEGORY_GL
CATEGORY
1:N
N:1
E3
E4
CGL1
CGL2
CGL3
CGL4
C1
C2
SOURCE
TARGET


ENTITY
INHERITANCE_SL
INHERITANCE
1:N
N:1
E5
E6
E7
E8
E9
E10
ISL1
ISL2
ISL3
ISL4
ISL5
ISL6
ISL7
I1
I2
I3
SOURCE
TARGET


ENTITY
INHERITANCE_GL
INHERITANCE
1:N
1:1
E3
E4
IGL1
IGL2
IGL3
I1
I2
I3
SOURCE
TARGET

rect_Inheritance_Link. Isto é, uma entidade (super-entidade) pode ter apenas uma Direct_Inheritance_Link, e um Direct_Inheritance_Link pode ser ligado com apenas uma super-entidade. Esta regra de boa formação é importante porque dois ou mais Direct_Inheritance_Link, ao mesmo super-entidade, pode ser modelado (sem perda semântica), utilizando uma sobreposição de disjunção ou herança. Portanto, a fim de evitar essa redundância modelagem, nosso metamodelo só permite uma Direct_Inheritance_Link pelo super-entidade. Além disso, a meta-Direct_Inheritance_Link entidade também tem uma relação alvo com a Entidade meta-entidade. Nessa relação, uma instância de Direct_Inheritance_Link tem apenas uma instância de Entidade como alvo, mas uma instância de entidade pode ser alvo de muitos instância de Direct_Inheritance_Link (ou seja, uma herança múltipla). Enfim, um Direct_Inheritance_Link também tem o papel meta-atributo. Similar aos últimos três figuras, na fig. 8 mostramos um fragmento de nosso metamodelo e um pedaço um esquema EER e sua ocorrência diagrama que ilustra as cardinalidades das relações origem e de destino. Na Fig. 8 estamos nos concentrando em Direct_Inheritance_Link (representado na linha de espessura).

Finalmente, note que a ligação não pode existir sem um nó (participação total), mas um nó pode ser instanciado sem uma conexão com um link (participação parcial), porque entendemos que nós um de-signatário primeiros modelos e depois de modelos links para ligado os nós. Além disso, todas as heranças são disjuntos (mutuamente exclusivos) e total (completamente definida), e todos os meta-atributos, excepto o papel meta-atributo, são de preenchimento obrigatório. Na próxima seção, apresentamos as principais regras de boa formação que são automaticamente suportados pelo nosso metamodelo.
Regras 4.1 de boa formação

Um metamodelo (ou sintaxe abstrata) de uma linguagem de modelagem descreve os conceitos, as relações entre eles e as regras estruturantes que limitam a combinação desses conceitos de acordo com as regras de domínio de uma linguagem de modelagem. Isto é, um metamodel é útil para fornecer uma exacta
definição de todos os conceitos de modelagem e as regras de boa formação necessários para a criação de modelos sintaticamente corretas [Kelly e Tolvanen de 2008]. Neste contexto, nós enumerar as principais regras de boa formação que estão intrinsecamente assegurados pelo nosso metamodelo. Ou seja, estas regras de boa formação provêm directamente de nosso metamodelo e, por essa razão, eles podem ser verificados contra (da mesma forma que quando é necessário verificar se um modelo fornece suporte para garantir regras de negócio).

(1) Um nó não pode ser conectado a outro nó sem uma ligação, e vice-versa. Ou seja, nem um nó pode ser conectado diretamente a outros nós nem um link pode ser conectado diretamente a outros links;

(2) uma ligação não pode ser criado sem um nó como uma fonte e outra como um alvo. Isto é, uma ligação não pode ser representado isolado na área do desenho;

(3) A Relationship_Link nem pode conectar uma Entidade para outros nem relação aos outros. Ou seja, Relationship_Link não pode existir entre as entidades ou entre os relacionamentos;

(4) uma herança ou uma categoria não pode ter um atributo e um atributo não pode ser ligado a mais de um elemento. Isto é, um atributo só pertence a uma entidade, uma Associative_Entity, um relacionamento ou outro atributo (atributo composto);

(5) uma herança pode ser separado ou sobreposição, total ou parcial e têm muitas Specialization_Links, mas não pode ter mais de um Generalization_Link. Em outras palavras, uma herança pode ser especializado em muitos sub-entidades. No entanto, uma herança deve ser

generalizada para apenas um super-entidade;

(6) A Categoria nem pode ter sobreposição nem mais de uma Specialization_Link, mas pode ter muitos Generalization_Link. Isto é, uma categoria é separado, tem apenas um sub-entidade, mas pode ter muitos super-entidades;

(7) Uma entidade não pode ser fonte de mais de um Category_SL. Ou seja, um sub-entidade pode ter apenas um Category_SL.

8) Uma entidade não pode ser fonte de mais de um Direct_Inheritance_Link. Ou seja, um super-entidade pode ter apenas um Direct_Inheritance_Link.

4.2 Análise Comparativa

Nesta secção, apresentamos os resultados da análise comparativa do nosso metamodelo (EERMM) contra os metamodelos de CWM e IMM. A tabela 1 resume os principais pontos discutidos nas secções 3 e 4, e apresenta os resultados de nossa análise. Neste sentido, podemos ver que (i) nem CWM nem IMM fraco apoio Entidade / Relacionamento, atributo composto, atributo discriminador, e categoria; (Ii) CWM apoia indirectamente: entidade associativa, com vários valores atributo, atributo derivado, e relacionamento com o atributo; (Iii) IMM também não suporta Associativo Entidade, relacionamento com o atributo, herança múltipla, o total de herança / parcial e disjuntos / herança sobreposição; e (iv) O nosso metamodelo abrange todos os construtores da TCE. Em resumo, como destacado ao longo deste artigo, os metamodelos EER de OMG claramente não são capazes de fornecer um suporte completo para o modelo EER de acordo com a notação de Chen, de tal forma que IMM (a mais recente) só cobre 55% das características mostradas em A Tabela 1 e CWM (o precursor do IMM), embora possa cobrir até 80% das funções (considerando construções indirectos derivados da notação UML), é um perfil UML sem regras OCL assegurar modelos válidos. Isto é, tendo em conta estas deficiências, temos boas razões para propor um novo meta-modelo, em vez de modificar as propostas existentes, principalmente porque não temos garantias de que o trabalho necessário para definir regras OCL ou para adicionar os recursos em falta é menos complexo do que o esforço necessário para definir EERMM. Além disso, o metamodel modificado pode ser mais complexo do que EERMM.

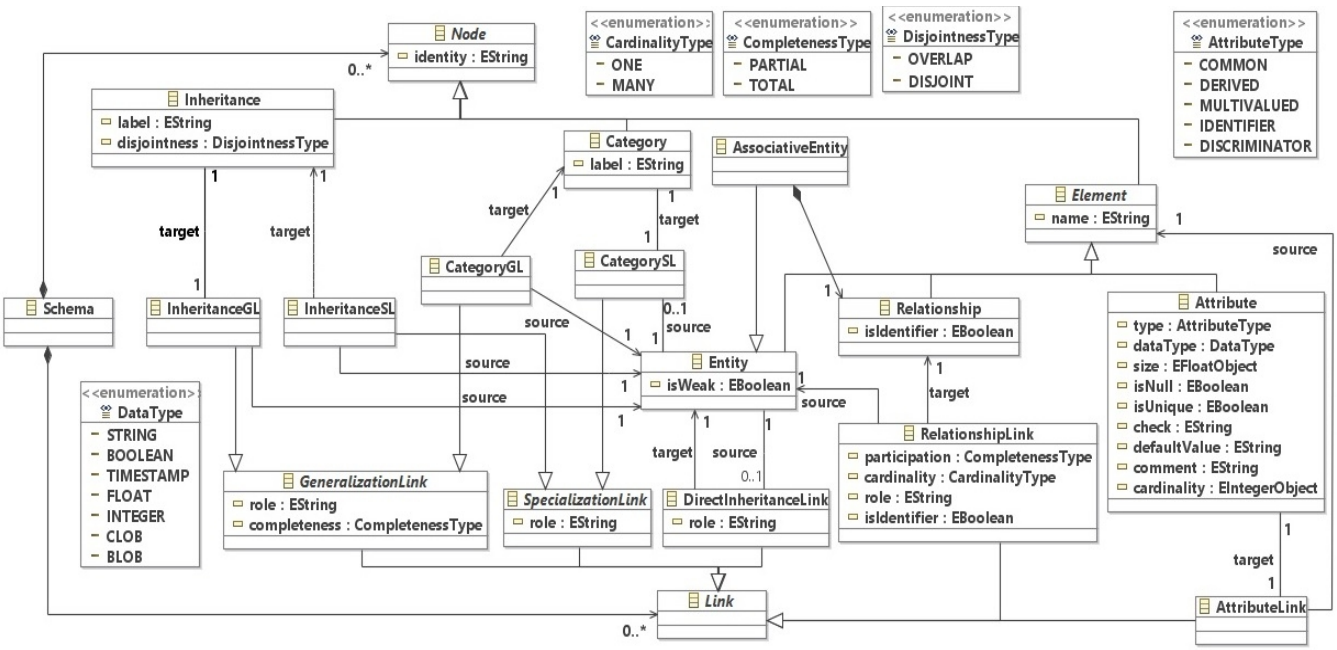
5. A APLICAÇÃO PRÁTICA

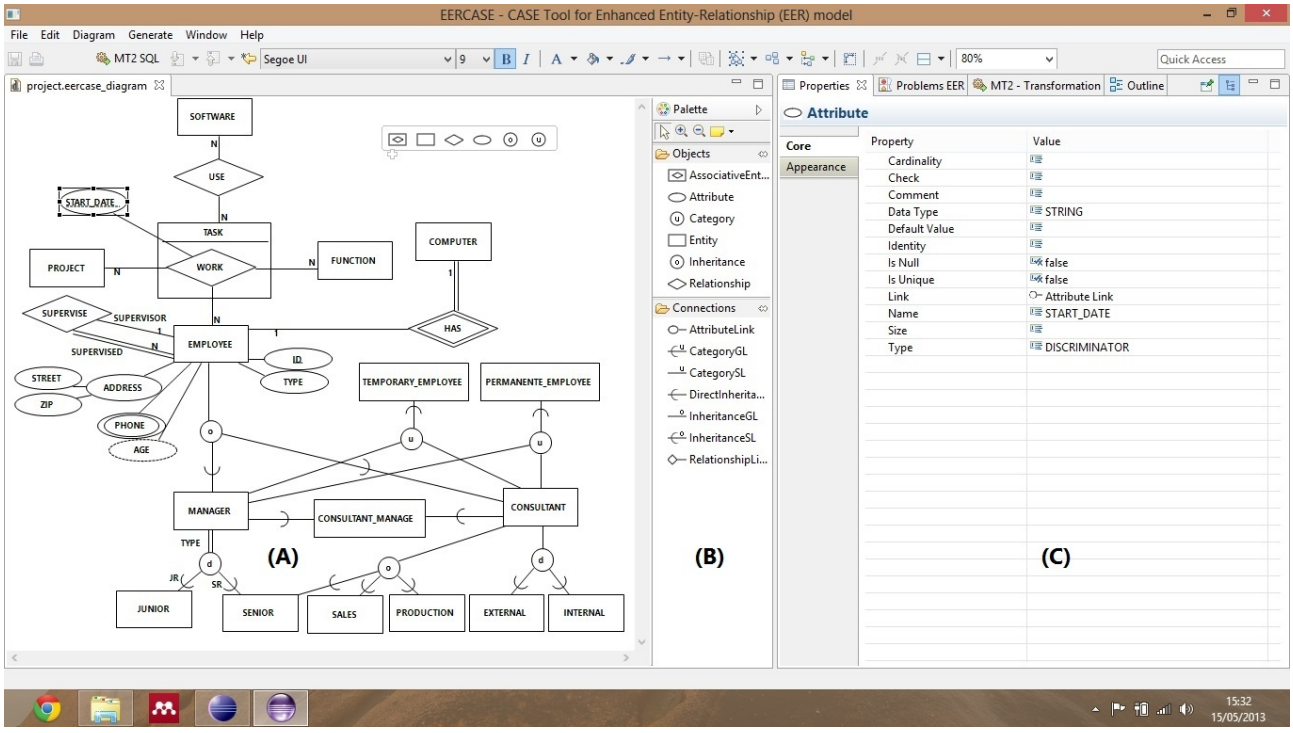
Um metamodelo é uma especificação para construtores de ferramentas CASE. Ou seja, uma ferramenta CASE que implementa um metamodelo impede a especificação de modelos inválidos, porque ele efetivamente assegura a coerência dos seus modelos. Neste contexto, a fim de demonstrar a viabilidade, expressividade e utilidade do nosso metamodelo, nós desenvolvemos uma ferramenta CASE, chamado EERCASE, e tê-lo usado para projetar

MAIN FEATURES OF CHEN'S EER MODEL		CWM	IMM	EERMM
1.	REGULAR ENTITY	YES	YES	YES
2.	ASSOCIATIVE ENTITY	INDIRECTLY	NO	YES
3.	WEAK ENTITY/RELATIONSHIP	NO	NO	YES
4.	SIMPLE ATTRIBUTE	YES	YES	YES
5.	COMPOSITE ATTRIBUTE	NO	NO	YES
6.	MULTI-VALUED ATTRIBUTE	INDIRECTLY	YES	YES
7.	DERIVED ATTRIBUTE	INDIRECTLY	YES	YES
8.	IDENTIFIER ATTRIBUTE	YES	YES	YES
9.	DISCRIMINATOR ATTRIBUTE	NO	NO	YES
10.	RELATIONSHIP WITH ATTRIBUTE	INDIRECTLY	NO	YES
11.	RECURSIVE RELATIONSHIP.	YES	YES	YES
12.	N-ARY RELATIONSHIP	YES	YES	YES
13.	RELATIONSHIP PARTICIPATION	YES	YES	YES
14.	RELATIONSHIP CARDINALITY	YES	YES	YES
15.	ROLE	YES	YES	YES
16.	SINGLE INHERITANCE	YES	YES	YES
17.	MULTIPLE INHERITANCE	YES	NO	YES
18.	TOTAL/PARTIAL INHERITANCE	YES	NO	YES
19.	DISJOINT/OVERLAP INHERITANCE	YES	NO	YES
20.	CATEGORY	NO	NO	YES
YES		60%	55%	100%
YES + INDIRECTLY		80%	55%	100%
NO		20%	45%	0%

nosso metamodelo (ver Fig. 4). Com a nossa ferramenta CASE, o designer pode interagir com um esquema EER, inserindo, excluindo, edição, visualização em diferentes níveis de zoom, e exportá-lo como uma figura ou como um arquivo XMI (XML Metadata Interchange). A ferramenta CASE é implementada utilizando tecnologias Java / Eclipse (ou seja, o Eclipse Modeling Framework (EMF) [Steinberg et al. 2009], Graphical Modeling Framework (GMF) [FOUNDATION de 2013] e Epsilon-Quadro [Kolovos et al. 2011], que são conformes à Meta Object Facility Essencial (EMOF) [OMG 2013] padrão. Além disso, a ferramenta gera código SQL para PostgreSQL e pode ser facilmente estendido para lidar com qualquer sistema de gerenciamento de banco de dados (novos compiladores só precisa considerar os tipos de dados e reservados palavras do novo alvo SQL / DDL dialetos). Na FIG. 9 nós mostramos uma visão de implementação do nosso metamodelo, que foi utilizado para desenvolver a nossa ferramenta CASE. Esta visão de implementação é especificado em Ecore [Steinberg et al. 2009] (o Java / tecnologia Eclipse usado para representar modelos em EMF / EMOF) e estende-se o ponto de vista conceptual apresentado na Fig. 4, a fim de (i) adicionar oito meta-atributos (tipo de dados, o tamanho, isnull, isUnique, cheque, defaultValue, comentar e cardinalidade) na metaclasse Atributo (porque estes meta-atributos são úteis para a geração de código ou para fins de anotação); (Ii) adicionar a meta-atributo isIdentifier na meta-RelationshipLink entidade (esta meta-atributo permite definir Entity.isWeak = True e Relationship.isIdentifier = True com apenas um clique para definir RelationshipLink.isIdentifier = True); (Iii) incorporar as enumerações que especificam os valores válidos para um atributo; e (iv) reformar o metaclass Associative_Entity como uma herança de uma entidade e uma composição de Relacionamento (porque Java não suporta herança múltipla).

Na Fig. 10 nós mostramos o Graphical User Interface (GUI) da nossa ferramenta CASE com um fragmento de um esquema hipotético para uma empresa de fabricação. É importante ressaltar que nosso objetivo é apresentar um exemplo didático e expressiva que é tão simples quanto possível e que abrange todos os construtores do modelo EER. Neste sentido, com o objetivo de não sobrecarregar o Fig. 10, só retratam um exemplo de cada tipo de atributo. Assim, nesta figura que ilustram uma entidade fraca (Computador), uma entidade associativa (Task), entidades regulares (todas as entidades, com exceção do computador), um atributo composto (endereço), um atributo de valores múltiplos (telefone), um atributo derivado (idade), um atributo-chave (id), um atributo discriminador (start_date), uma relação com um atributo (start_date),, de valor único e simples atributos armazenados (todos os atributos, exceto estes recentemente classificada), um relacionamento unário (Supervisionar), um relacionamento ternário (Trabalho), dois relacionamentos binários (usar e tem), papéis (supervisor e super-Vised), restrições de participação (total ou parcial - desenhado usando links de relacionamento com uma linha de casal ou solteiro, respectivamente), restrições de cardinalidade (um ou muitos -depicted usando o caracteres "1" ou "N" em links de relacionamento, respectivamente), heranças (disjuntos, sobreposição, totais e parciais - representados usando: "d", "o", linha dupla e única linha, respectivamente) e as categorias (representado usando "L").





Note que a GUI da nossa ferramenta CASE tem uma paleta (área "B" na Fig. 10), com todos os construtores do modelo EER. As tarefas de modelação começar com um clique no construtor desejada do Modelo EER seguido pela selecção onde irá ser colocado na área de desenho (região "A" na Fig. 10). Em seguida, o designer pode editar as propriedades do construtor (área "C" na Fig. 10), e adicionar novos construtores. Uma vez terminada a modelagem, o designer pode validar seu esquema usando nossa ferramenta CASE (Menu Editar -> Validar ou salvar o esquema). Por exemplo, pode verificar se: (i) há duas entidades ou dois atributos (na mesma entidade) com o mesmo nome; (Ii) a integridade e restrições disjunção de uma herança estão correctamente definidas (essas restrições só podem ser definidos em uma herança com, pelo menos, duas sub-entidades); (Iii) existem os nós isolados; e (iv) uma relação, bem como uma entidade fraca tem um atributo identificador (nestes casos, devemos usar um atributo discriminador quando uma identificação é necessária). Essas validações são implementados usando a Linguagem Epsilon Validation (EVL) [Kolovos et al. 2011]. Uma vez validado, o designer pode gerar automaticamente o código SQL /

DLL para o esquema modelado. O código de geração de scripts de SQL / DDL é implementado usando o Epsilon Generation Language (EGL) [Kolovos et al. 2011]. Devido a limitações de escopo e de espaço, uma especificação detalhada da nossa ferramenta CASE não está incluída neste artigo. No entanto, mais informações sobre a nossa ferramenta CASE (incluindo o seu link de download) podem ser encontradas em http://www.cin.ufpe.br/ eercase.

6. CONSIDERAÇÕES FINAIS

Metamodelos desempenhar um papel fundamental na linguagem de modelagem de construção ferramenta de definição e CASE, porque descrevem os construtores e efetivamente orientar o desenvolvimento de ferramentas CASE para linguagens de modelagem. Isto é, um metamodel especifica construtores de modelação, as suas relações, e as suas regras de boa formação, prevenindo a especificação de modelos inválidos. Portanto, um metamodelo oferece facilidades para a construção de ferramentas CASE. Por exemplo, uma ferramenta CASE baseada em um metamodelo pode detectar erros mais cedo ou até mesmo evitar que isso aconteça, orientar no sentido de padrões de design preferíveis, vá integralidade, informando sobre os elementos em falta, reduzir o trabalho de modelagem através da aplicação de convenções e valores padrão, apoiar a geração de código completo, e manter consistente especificações. Além disso, um metamodel também pode servir como uma opção simples para o intercâmbio de metadados com outras ferramentas. Portanto um metamodelo é tão útil para uma ferramenta de linguagem / CASE de modelagem como uma gramática é uma linguagem de programação / compilador [Kelly e Tolvanen de 2008].

Embora o modelo EER é uma das linguagens de modelagem mais utilizadas para o projeto conceitual de banco de dados e muito trabalho tem sido feito sobre este modelo, com o melhor de nosso conhecimento, existem apenas duas propostas de metamodelos TCE, e eles não abordam todos construtores de notação da Chen (ver pontos 3 e 4.2). Além disso, nem uma discussão sobre a engenharia utilizada para formalizar estes metamodelos foi apresentado nem uma comparação fundamentada entre eles pode ser encontrada na literatura.

Por estas razões, neste artigo temos mostrado uma visão detalhada e prática de como formalizar o modelo EER por meio de um metamodelo que (i) cobre todos os construtores desta linguagem de modelagem; (Ii) define regras de boa formação para prevenir a modelagem de esquemas TCE sintaticamente incorretos; (Iii) pode ser usado como um ponto de partida (um quadro teórico) por obras que visam construir ferramentas CASE TCE (por exemplo, extensão ou reutilizar parte do nosso metamodelo para abordar novos recursos ou metadados intercâmbio entre as ferramentas); e (iv) suporta SQL / DDL geração automática de código.

Com o objetivo de especificar um metamodelo independente de tecnologia, especificamos uma visão conceitual de nosso metamodelo (ver Fig. 4) por meio do Modelo EER (ou seja, um metamodelo reflexiva). Além disso, nós também apresentar uma visão de implementação do nosso metamodelo (ver Fig. 9). Esta visão de implementação foi modelado de acordo com tecnologias Ecore / EMOF e foi contratado para construir a nossa ferramenta CASE, que foi usado para modelar a visão conceitual do nosso metamodelo. A fim de mostrar a viabilidade, utilidade e expressividade do nosso metamodelo, nós usamos nossa ferramenta CASE para modelar um exemplo simples que abrange todos os construtores da TCE, confirmando que o nosso metamodelo é viável e mais expressiva do que aqueles relacionados (cf. secções 3 e 4.2). Nossa ferramenta CASE foi implementado em Java usando a estrutura EMF, GMF, e Epsilon. Em sua versão atual, ele gera o código SQL / DDL para o PostgreSQL. Em trabalhos futuros pretendemos cobrir outros sistemas de gerenciamento de banco de dados também. Nós também planejamos estender nossa metamodelo para apoiar a modelagem de dados espacial e temporal, realizar um teste de usabilidade de nossa ferramenta CASE, desenvolver módulos de engenharia reversa e especificar formalizações

matemáticas / provas para mostrar a justeza da nossa metamodelo e as suas regras de boa formação. Em conclusão, nosso trabalho contribui para o avanço do estado da arte de meta para o modelo EER, porque fornece uma nova perspectiva da investigação científica e aplicação industrial no campo de banco de dados de modelagem conceitual, uma vez que aponta as limitações atuais e mostra um mais expressivo maneira de definir um metamodelo EER. Em outras palavras, a metamodel é o primeiro que cobre todos os elementos de notação da Chen, o que é particularmente importante porque define a base para o modelo de TCE. Ou seja, nosso metamodelo define um conjunto de declarações que não deve ser falsa para qualquer modelo EER válido.