Metamodeling the Enhanced Entity-Relationship Model

Robson N. Fidalgo[1], Edson Alves[1], Sergio España[2], Jaelson Castro[1], Oscar Pastor[2]

[1] Center for Informatics, Federal University of Pernambuco, Recife(PE), Brazil
{rdnf, eas4, jbc}@cin.ufpe.br
[2] Centro de Investigación ProS, Universitat Politècnica de València, València, España
{sergio.espana,opastor}@pros.upv.es

Abstract. A metamodel provides an abstract syntax to distinguish between valid and invalid models. That is, a metamodel is as useful for a modeling language as a grammar is for a programming language. In this context, although the Enhanced Entity-Relationship (EER) Model is the "de facto" standard modeling language for database conceptual design, to the best of our knowledge, there are only two proposals of EER metamodels, which do not provide a full support to Chen's notation. Furthermore, neither a discussion about the engineering used for specifying these metamodels is presented nor a comparative analysis among them is made. With the aim at overcoming these drawbacks, we show a detailed and practical view of how to formalize the EER Model by means of a metamodel that (i) covers all elements of the Chen's notation, (ii) defines well-formedness rules needed for creating syntactically correct EER schemas, and (iii) can be used as a starting point to create Computer Aided Software Engineering (CASE) tools for EER modeling, interchange metadata among these tools, perform automatic SQL/DDL code generation, and/or extend (or reuse part of) the EER Model. In order to show the feasibility, expressiveness, and usefulness of our metamodel (named EERMM), we have developed a CASE tool (named EERCASE), which has been tested with a practical example that covers all EER constructors, confirming that our metamodel is feasible, useful, more expressive than related ones and correctly defined. Moreover, we analyze our work against the related ones and present our final remarks.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Data description languages

Keywords: enhanced entity-relationship model, metamodel

1. INTRODUCTION

Modeling languages are used to create models that aim to raise the level of abstraction and hide implementation details. In order to prevent invalid models, the syntax of a modeling language must define well-formedness rules that specify what a valid model is. In this context, it is important to highlight that the abstract syntax of a modeling language is formalized by means of a metamodel, which also serves as a basis to interchange models with other tools. For this reason, the word "meta" is used because the modeling language specification is one level higher than the usual models. In its simplest form, we can say that a metamodel is a conceptual model of a modeling language. That is, it describes the concepts of a modeling language, their properties and the legal connections between language elements. Therefore, metamodels are mandatory and important artifacts in the specification of modeling languages. In other words, metamodels are basic entities for model designers and tool developers, because they precisely define how tools and models should work together [Kelly and Tolvanen 2008].

The Enhanced Entity-Relationship (EER) Model [Chen 1976] is one of the most used modeling languages for the conceptual design of database, that is: (i) many research proposals use the EER notation [Bollati et al. 2012; Amalfi et al. 2011; Badia and Lemire 2011; Motta and Pignatelli 2011; Calì et al. 2010; Franceschet et al. 2009; Karanikolas and Vassilakopoulos 2009; Xu et al. 2010; Zhang

et al. 2008; Combi et al. 2008]; (ii) there are a numerous Computer Aided Software Engineering (CASE) tools that aim to support EER modeling (e.g., EER/Studio XE2, PowerDesigner, ERWin and SmartDraw); (iii) most computer science university curricula includes it [ACM/IEEE 2008], and (iv) most database textbooks [Elmasri and Navathe 2010; Silberschatz et al. 2010; Connolly and Begg 2009; Garcia-Molina et al. 2008] dedicate one or more chapters to present it. However, to the best of our knowledge, there are no metamodels that provide full support to Chen's notation (including its extensions – cf. Fig.1). In this context, since (i) a metamodel is like a grammar for a modeling language, (ii) the EER Model is the "de facto" standard for database conceptual modeling, and (iii) there is no EER metamodel that effectively covers all constructors of the Chen's notation (the most used EER notation), we argue that the specification of an EER metamodel is as useful for the Database community as the specification of the UML metamodel is for the Software Engineering community.

Aiming to overcome the previous shortcoming, in this article, we present a detailed and practical view of how to formalize the EER Model by means of a metamodel that (i) supports all constructors of Chen's notation, (ii) defines well-formedness rules that prevent the designing of invalid EER schemas, and (iii) can be used as a starting point to (a) create EER CASE tools, (b) interchange metadata among these tools, (c) allow automatic SQL/DDL code generation, and/or (d) extend (or reuse part of) the EER Model to address new features (e.g. spatial and/or temporal modeling).

The remainder of this article is organized as follows. In Section 2 we briefly describe the main modeling languages for the conceptual design of databases. Following, in Section 3 we discuss related work. After that, in Section 4, we present how our metamodel was defined, its well-formedness rules and a comparative analysis between our work and the most important ones. In turn, in Section 5 we show a general idea of how our metamodel has been used to build an EER CASE tool and, using this tool, we model a practical example covering all EER constructors. Finally, in Section 6 we provide some conclusions and indications of future works.

2. MODELING LANGUAGES FOR CONCEPTUAL DESIGN OF DATABASES

There are many notations that can be used for the conceptual design of database [Song et al. 1995]. Some of them allow binary relationships at most and for this reason are known as binary notations, while others allow n-ary relationships and are known as n-ary notations. In general, most CASE tools use binary notations. However, these notations (i) do not allow attributes in relationships – which does not seem natural, because relationships can have descriptive properties and (ii) do not allow relationships among three or more entities – which is also inconvenient, because some relationships are intrinsically n-ary and, although there are transformations from an n-ary relationship to n-binary relationships, the result often has different semantics unless accompanied by integrity constraints [Hartmann 2003; Jones and Song 1996; Song et al. 1995]. For these reasons, n-ary notations are semantic and conceptually more expressive than binary notations [Song et al. 1995], and are more often used in academia. Regardless of the notation used, there are two different conventions to refer the place where roles, cardinalities, and participations are specified on a relationship, namely "look-across" (i.e., opposite-side of entity) and "look-here" (i.e., same-side of entity) [Hartmann 2003] [Song et al. 1995].

Among the notations for the conceptual design of a database, the EER Model and the Class Diagram (CD) of the Unified Modeling Language (UML) [Rumbaugh et al. 2004] are the most used. Recently, three research studies [Lucia et al. 2008] [Lucia et al. 2010] [Bavota et al. 2011] perform comparisons between the EER Model and the CD and report interesting results. Lucia et al. [2008] and Lucia et al. [2010] suggest that the CD notation is generally more comprehensible than the EER notation. However, Bavota et al. [2011] conclude that the understanding of EER Model is significantly higher than CD if "composite attribute", "multi-value attribute", and "weak entity" are required in a given modeling. We point out that the authors have still not taken into account constructions such as "associative entities", "attributes in relationships", and the "inaccuracy of look-across notations (such

as CD)" that does not effectively represent the semantics of participation constraint on relationships with degree higher than two [Hartmann 2003; Dullea et al. 2003; Song et al. 1995]. Therefore some important EER constructors still need to be evaluated. We expect that, if an assessment is performed taking in account the full set of constructors, more advantages to the EER Model are revealed. To summarize this discussion, we think that any comparison between the EER Model and the CD can provide results to debate and arguing and the choice for a particular notation rather than other is a matter of preference or project contingency.

As our work is based on the EER Model, in Fig. 1, we show its constructors according to the Elmasri-Navathe's notation [Elmasri and Navathe 2010]. We choose this notation because it is very close to the original Chen, is widely used in database courses and is well accepted by the database community [Song and Chen 2009]. Fig. 1 presents all components of EER Model according to the Elmasri-Navathe's notation. Since Category is a useful concept, but it is not supported by any other notation, in order to be self-contained, we present an overview of this constructor and explain the difference between Category and Multiple Inheritance. Thus, a Category represents a collection of instances that is a subset of (partial Category) or the union of (total Category) distinct entities. Note that a category should have, at least, two super-entities. That is, unlike a Multiple Inheritance, an instance of a Category belongs to one, not all, of the super-entities. We point out that Elmasri-Navathe's notation is n-ary, uses look-here for specifying roles and participations and look-across for specifying cardinalities [Elmasri and Navathe 2010].

3. RELATED WORK

Despite the weaknesses of CD (cf. Section 2), many UML vendors have expressed a desire to use CD for data modeling and ended up defining their own tool-specific profiles for each. As a result, there is neither an accepted standard nor interoperability of models developed using such profiles/tools [OMG 2003]. Aiming at overcoming such limitation, the Object Management Group (OMG) is revising its specification named Common Warehouse Metamodel (CWM) [OMG 2001] in order to provide more normative metamodels (including a metamodel for the EER Model) called Information Management Metamodel (IMM) [OMG 2009]. We show in Fig. 2 (based on OMG [2001]) and Fig. 3 (based on OMG [2009]) the EER metamodels of CWM and IMM, respectively.

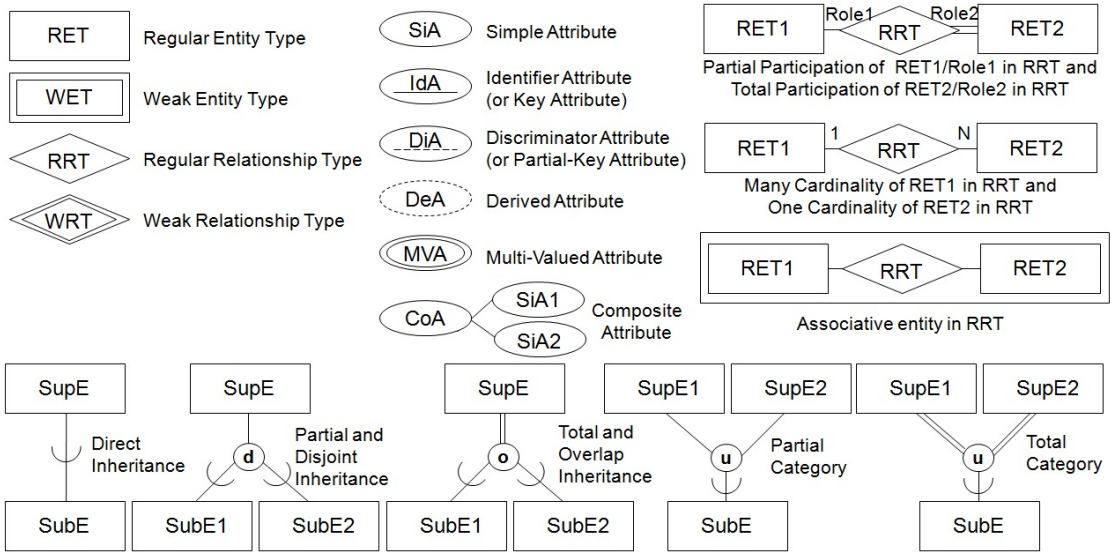


Fig. 1. Constructors of the EER Model according to the Elmasri-Navathe's notation

In Fig. 2, the CWM metamodel extends the UML metamodel and specifies the metaclasses for the main concepts of the EER Model (i.e., Entity, Relationship and Attribute). Considering the EER constructors shown in Fig. 1, it can be noted that the CWM metamodel neither includes metaclasses nor meta-attributes for Weak Entity/Relationship, Composite Attribute, Discriminator Attribute, and Category. Furthermore, it is worth emphasizing that the constructors Associative Entity, Multi-valued Attribute, Derived Attribute, and Relationship with Attribute, although not directly represented in Fig.2, can be mapped using the UML notation. That is: (i) an Associative Entity can be represented as an Association Class; (ii) a Multi-valued Attribute can be represented using brackets "[ ]"; (iii) a Derived Attribute can be represented using a leading slash "/", and (iv) an Attribute on Relationship can also be represented using an Association Class.

The other constructors/constructions of the EER Model, namely Unary Relationship, Relationship Cardinality, Relationship Participation, N-ary Relationship, Role, Single Inheritance, Multiple Inheritance, Total/Partial Inheritance, and Disjoint/Overlap Inheritance, even though not represented in Fig. 2, they can also be captured by CWM, since these constructors/constructions are defined in the UML metamodel. It is important to highlight that the CD is a general purpose language. Therefore an extension of the UML metamodel for database design requires the definition of some restrictions in Object Constraint Language (OCL) to suppress UML elements that are out of database context (i.e., unnecessary elements that can introduce errors) and assure well-formedness rules. However, no OCL constraint is specified in the EER metamodel of CWM, which is an important limitation. Furthermore, the CWM metamodel defines the ForeignKey metaclass, which is a constructor for relational database modeling. That is, this is another negative point, because this metaclass is not a constructor of the EER Model.

The EER metamodel of IMM (see Fig. 3), unlike the CWM metamodel, neither extends UML metaclasses for specifying its metamodel nor mixes EER constructors with relational database ones. However, also taking into account the EER constructors presented in Fig. 1, the IMM does not specify metaclasses, meta-attributes, or constructions for Associative Entity, Category, Discriminator Attribute, Composite Attribute, Multi-valued Attribute, Attribute on Relationship, Specialization

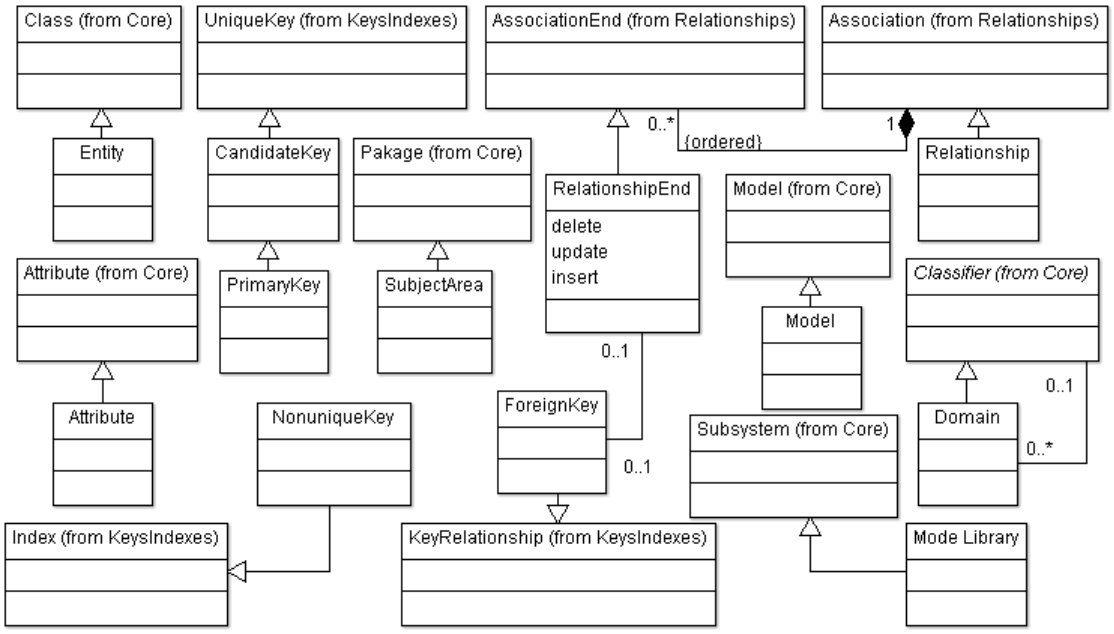


Fig. 2. The EER metamodel of CWM

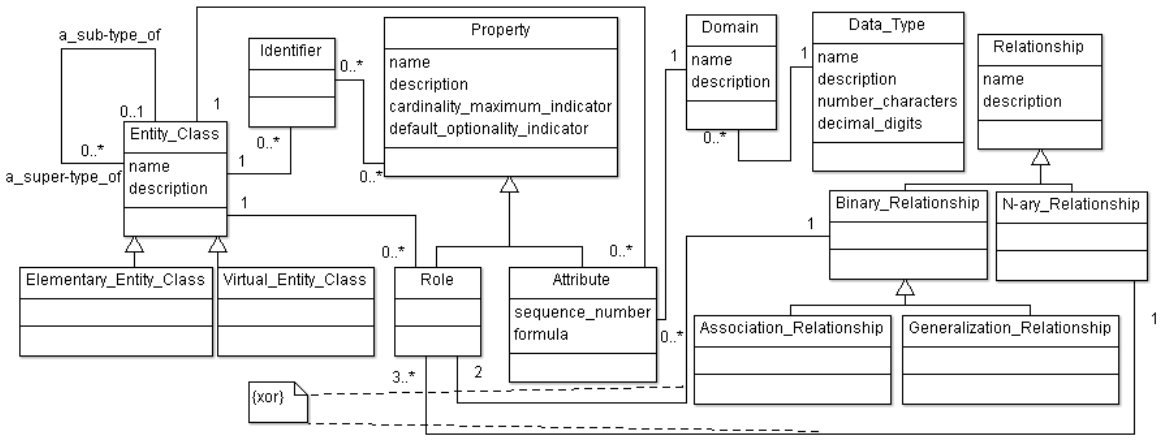


Fig. 3. The EER metamodel of IMM

Relationship, and Multiple Inheritance. Therefore, there is still no metamodel that provides a full support to Chen's notation.

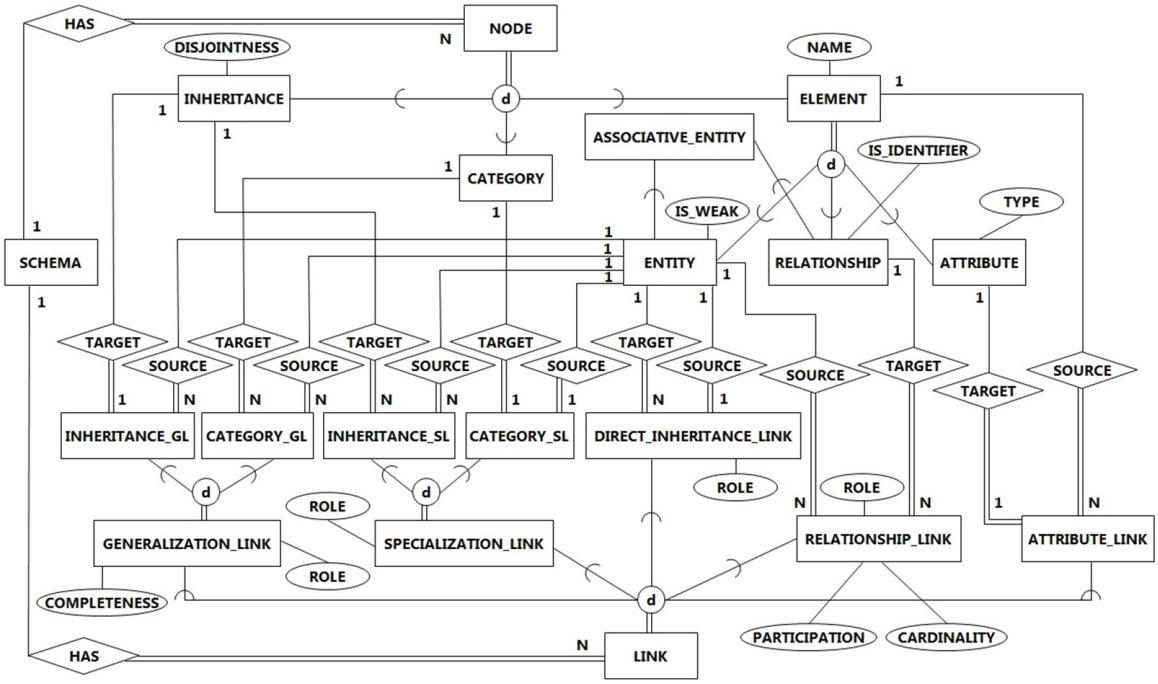
4. METAMODELING THE EER MODEL

The EER Model is a node-link diagram [Irani et al. 2001; Ware and Bobrow 2004], where nodes represent Entities, Associative Entities, Attributes, Relationships, Inheritances, and Categories, whereas links denote Attribute Link, Relationship Link, Generalization Link, and Specialization Link (see the bottom of Fig. 4). Then, with the node-link abstraction and the EER notation in mind, a conceptual view of our metamodel, named Enhanced Entity-Relationship MetaModel (EERMM), is presented in Fig 4 (see the top).

According to Fig.4, our metamodel has three main meta-entities: Schema, Node, and Link. Schema is the root meta-entity that corresponds to the drawing area of an EER schema. For this reason, Schema can have many instances of Node and many instances of Link, which cannot exist (total participation) without a Schema. Besides the main meta-entities, our metamodel has three specialized meta-entities for Node: Element, Inheritance, and Category, which cover the main constructors of the EER Model.

The Element meta-entity has a meta-attribute called name (i.e., a label) and it is specialized in three meta-entities: Entity, Relationship, and Attribute, which are used to capture concepts of the real world, how these concepts are related to each other, and the properties of these concepts/relations, respectively. Note that (i) the Entity meta-entity has a meta-attribute called is_Weak that is a Boolean value (i.e., is_Weak = "True" defines a weak entity and is_Weak = "False" defines a regular entity); (ii) the Relationship meta-entity has a meta-attribute called is_Identifier that is also a Boolean value (i.e., is_Identifier = "True" defines an identifying relationship and is_Identifier = "False" defines a regular relationship); (iii) the Attribute meta-entity has a meta-attribute called type that specifies whether an instance of Attribute is "common", "identifier", "discriminator", "derived" or "multivalued", exclusively; and (iv) the Associative_Entity meta-entity is both an Entity and a Relationship (i.e., it is a specialization of these meta-entities).

In the sequence, the Inheritance meta-entity captures the "inheritance" concept and has the disjointness meta-attribute, which defines whether an inheritance is mutually exclusive ("disjoint") or not ("overlap"). Finally, the Category meta-entity captures the "category" concept (also called "union type"). Note that, by definition [Elmasri and Navathe 2010], a category is disjoint. Then, in our metamodel we do not model the disjointness meta-attribute for a category.



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

Fig. 4. Conceptual view of our metamodel

Besides the Node meta-entities, our metamodel has five specialized meta-entities for Link, namely Attribute_Link, Relationship_Link, Specialization_Link, Generalization_Link, and Direct_Inheritance_Link, which address the links for an attribute, a relationship, a specialization, a generalization, and a direct inheritance, respectively. These meta-entities are presented as follow:

The Attribute_Link meta-entity has a source relationship with the Element meta-entity such that an instance of Attribute_Link has only one instance of Element as source, but an instance of Element can be source for many instances of Attribute_Link. Furthermore, the Attribute_Link meta-entity has a target relationship with the Attribute meta-entity such that an instance of Attribute_Link has only one instance of Attribute as target and an instance of Attribute can be target for only one instance of

Attribute_Link. That is, an instance of Element can be source for many instances of Attribute_Link (i.e., many attributes on entities, many attributes on relationships and many composite attributes), but an instance of Attribute can be target for only one instance of Attribute_Link (i.e., an Attribute cannot be linked with more than one Element). In order to clarify this discussion, in Fig.5 we focus on the Attribute_Link (depicted in thick line) and, in this figure, we present the part of our metamodel that addresses this type of link and a fragment of an EER schema with its occurrence diagram illustrating the cardinalities of source and target relationships.

The Relationship_Link meta-entity also has a source relationship with the Entity meta-entity. In this relationship an instance of Relationship_Link has only one instance of Entity as source, but an instance of Entity can be source for many instances of Relationship_Link. Moreover, the Relationship_Link meta-entity has a target relationship with the Relationship meta-entity so that an instance of Relationship_Link has only one instance of Relationship as target. However, an instance of Relationship can be target for many instances of Relationship_Link. That is, an instance of Relationship_Link connects one instance of Entity with one instance of Relationship, but an instance of Entity or an instance of Relationship can be connected to many instances of Relationship_Link (i.e., an entity can be linked to many relationships and a relationship can be n-ary). Furthermore, the Relationship_Link meta-entity has three meta-attributes: (i) participation – it determines whether all (i.e., "total") or only some (i.e., "partial") instances of an entity will participate in the relationship; (ii) cardinality – it specifies whether the maximum number of instances of an entity in a relationship is "one" or "many"; and (iii) role – it is a text that can be used to describe the function of an entity in a relationship. In Fig. 6 we focus on Relationship_Link (depicted in thick line). In this figure we show the part of our metamodel that cover the Relationship_Link meta-entity, as well as a piece of an EER schema and its occurrence diagram that exemplify the cardinalities of the source and target relationships.

Other meta-entity of our metamodel is Specialization_Link. This meta-entity has the role meta-attribute that corresponds to a text for describing the function of a sub-entity in a specialization. Specialization_Link is specialized in two meta-entities: Inheritance_SL (Inheritance Specialization Link) and Category_SL (Category Specialization Link), which are used to define a specialization link between a sub-entity and an inheritance or between a sub-entity and a category, respectively. Note that: (i) an instance of Inheritance_SL or Category_SL has only one instance of Entity as source. However, while an instance of Entity can be source for many instances of Inheritance_SL (i.e., a sub-entity can be a specialization of many inheritances – a multiple inheritance), an instance

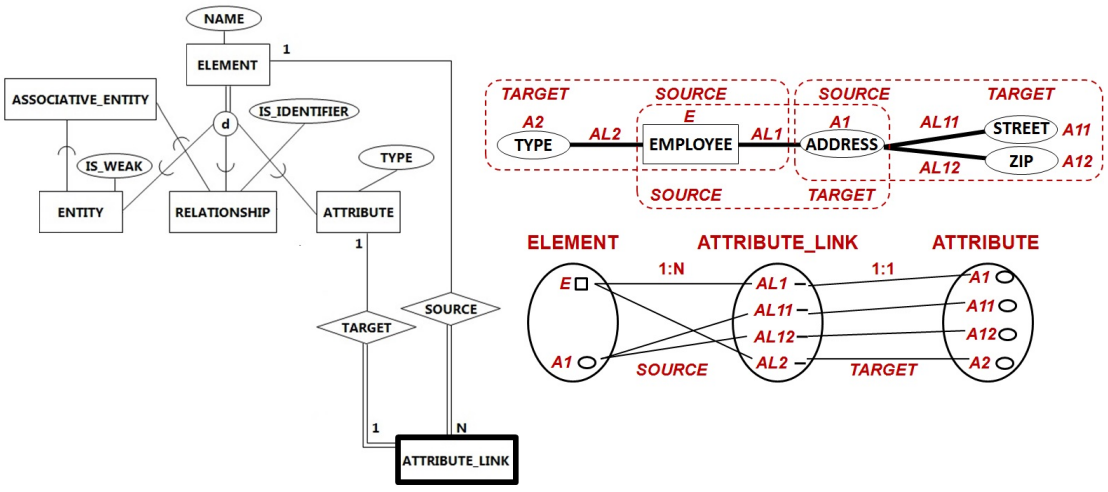


Fig. 5. An example of Attribute_Linkl

of Entity can be source for only one instance of Category_SL (i.e., a sub-entity can be a specialization of only one category) and (ii) an instance of Inheritance_SL or Category_SL has only one instance of Inheritance or Category as target, respectively. Nevertheless, although an instance of Inheritance can be target for many instances of Inheritance_SL (i.e., a simple inheritance with many sub-entities), an instance of Category, by definition [Elmasri and Navathe 2010], can be target for only one instance of Category_SL (i.e., a category can only have one sub-entity).

Similarly to the Specialization_Link, the Generalization_Link meta-entity has also the role meta-attribute corresponding to a text that can be used to describe the function of a super-entity in a generalization. Furthermore, Generalization_Link has the completeness meta-attribute, which specifies whether the generalization is completely defined ("total") or not ("partial"). Generalization_Link is specialized in two meta-entities: Inheritance_GL (Inheritance Generalization Link) and Category_GL (Category Generalization Link), which are used to define a generalization link between a super-entity and an inheritance, or between a super-entity and a category, respectively. Note that: (i) an instance of Inheritance_GL or Category_GL has only one instance of Entity as source, but an instance of Entity can be source for many instances of Inheritance_GL (i.e., a super-entity can be a generalization of many inheritances) or Category_GL (i.e., a super-entity can take part of many categories) and (ii) an instance of Inheritance_GL or Category_GL has only one instance of Inheritance or Category as target, respectively. However, while an instance of Inheritance can be target for only one instance of Inheritance_GL (i.e., an inheritance can have only one super-entity), an instance of Category can be target for many instance of Category_GL (i.e., a category can have many super-entities. Actually, it must have at least two super-entities). Similar to the last two figures, in Fig. 7 we exhibit a fraction of our metamodel and a piece of an EER schema with its four occurrence diagrams that illustrate the cardinalities of the source and target relationships. In Fig. 7 we are focusing on Inheritance_SL, Category_SL, Inheritance_GL and Category_GL (depicted in thick lines and using different colors).

Differently to the Inheritance_SL and the Inheritance_GL, the Direct_Inheritance_Link meta-entity connects two entities without using an inheritance node. A Direct_Inheritance_Link is important, because sometimes it is necessary to model a multiple inheritance such that a sub-entity directly inherits from two or more super-entities or model an inheritance with only one sub-entity. Note that, in both cases it is a mistake to define the disjointness and completeness properties of an inheritance with only one sub-entity. For this reason, a Direct_Inheritance_Link does not have these properties. Concerning the source and target relationships, the Direct_Inheritance_Link meta-entity has a source relationship with the Entity meta-entity such that an instance of Direct_Inheritance_Link has only one instance of Entity as source and an instance of Entity can be source for only one instance of Di-

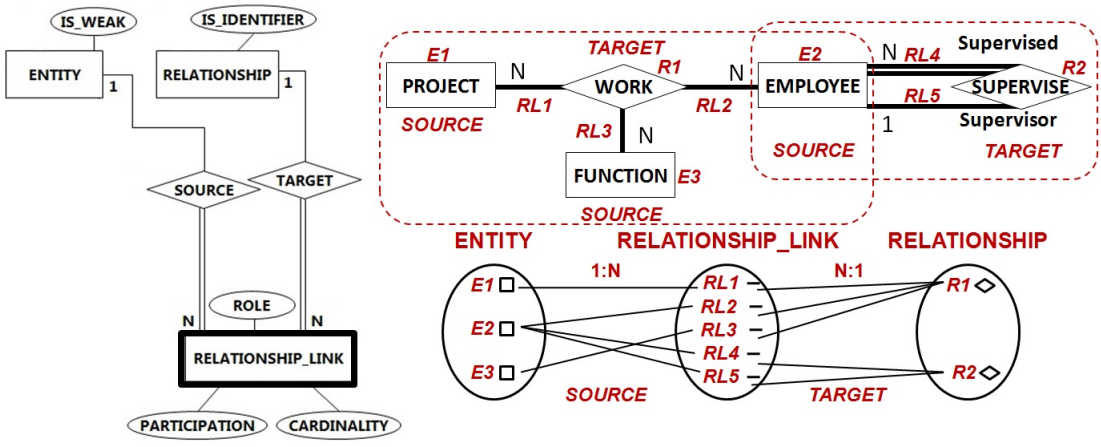


Fig. 6. A example of Relationship_Link

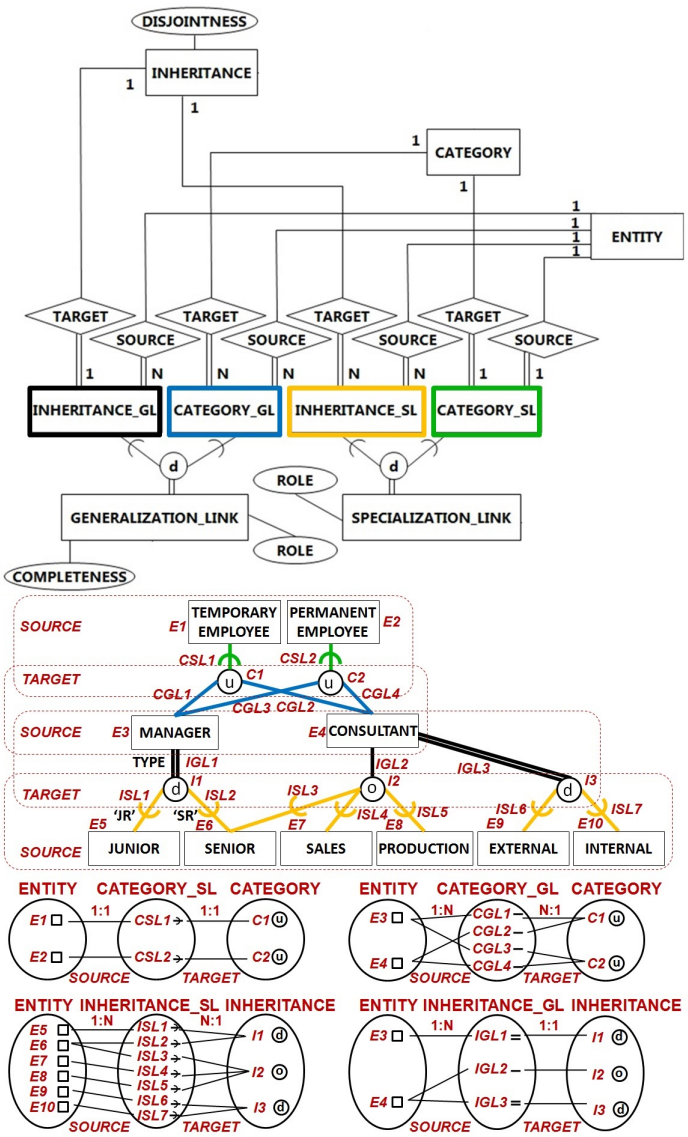


Fig. 7. A example of Inheritance_SL, Category_SL, Inheritance_GL and Category_GL

rect_Inheritance_Link. That is, an Entity (super-entity) can have only one Direct_Inheritance_Link, and a Direct_Inheritance_Link can be connected with only one super-entity. This Well-Formedness rule is important because two or more Direct_Inheritance_Link, at the same super-entity, can be modeled (without semantic loss) using a disjoint or overlap inheritance. Therefore in order to avoid this modeling redundancy, our metamodel only allows one Direct_Inheritance_Link by super-entity. Furthermore, the Direct_Inheritance_Link meta-entity also has a target relationship with the Entity meta-entity. In this relationship, an instance of Direct_Inheritance_Link has only one instance of Entity as target, but an instance of Entity can be target for many instance of Direct_Inheritance_Link (i.e., a multiple inheritance). At last, a Direct_Inheritance_Link also has the role meta-attribute. Similar to the last three figures, in Fig. 8 we show a fragment of our metamodel and a piece an EER schema and its occurrence diagram that illustrate the cardinalities of the source and target relationships. In the Fig. 8 we are focusing on Direct_Inheritance_Link (depicted in thick line).

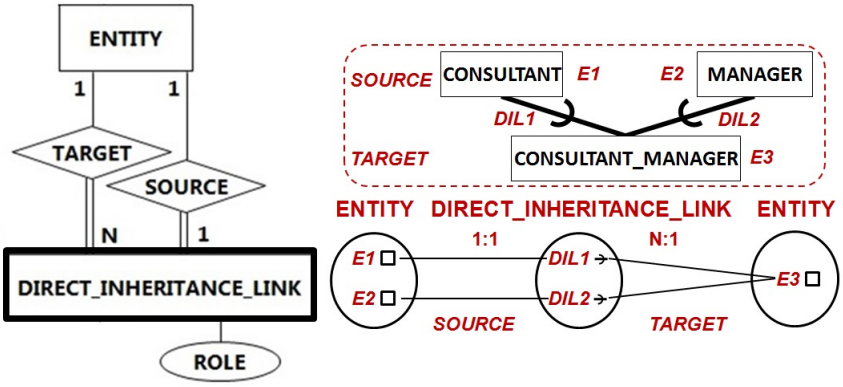


Fig. 8. A example of Direct_Inheritance_Link

Finally, note that a Link cannot exist without a Node (total participation), but a Node can be instantiated without a connection to a Link (partial participation), because we understand that a designer first models nodes and after models links to connected the nodes. Furthermore, all inheritances are disjoint (mutually exclusive) and total (completely defined), and all meta-attributes, except the role meta-attribute, are mandatory. In the next section, we present the main well-formedness rules that are automatically supported by our metamodel.

4.1 Well-Formedness Rules

A metamodel (or abstract syntax) of a modeling language describes the concepts, the relationships between them, and the structuring rules that constrain the combination of these concepts according to the domain rules of a modeling language. That is, a metamodel is useful to provide a precise definition of all modeling concepts and the well-formedness rules needed for creating syntactically correct models [Kelly and Tolvanen 2008]. In this context, we enumerate the main well-formedness rules that are intrinsically assured by our metamodel. That is, these well-formedness rules are directly derived from our metamodel and, for this reason, they can be checked against it (the same way as when it is necessary to check whether a model provides support to assure business rules).

(1) A Node cannot be connected to another Node without a Link, and vice-versa. That is, neither a node can be directly connected to other nodes nor a link can be directly connected to other links;
(2) A Link cannot be created without a Node as a source and another as a target. That is, a Link cannot be depicted isolated in the drawing area;
(3) A Relationship_Link neither can connect an Entity to others nor a Relationship to others. That is, Relationship_Link cannot exist among entities or among relationships;
(4) An Inheritance or a Category cannot have an Attribute and an Attribute cannot be linked to more than one Element. That is, an Attribute only belongs to an Entity, an Associative_Entity, a Relationship or another Attribute (composite attribute);
(5) An Inheritance can be disjoint or overlap, total or partial and have many Specialization_Links, but cannot have more than one Generalization_Link. In other words, an Inheritance can be specialized in many sub-entities. However, an Inheritance must be generalized to only one super-entity;
(6) A Category neither can have overlapping nor more than one Specialization_Link, but it can have many Generalization_Link. That is, a Category is disjoint, has only one sub-entity, but can have many super-entities;
(7) An Entity cannot be source of more than one Category_SL. That is, a sub-entity can have only one Category_SL.

(8) An Entity cannot be source of more than one Direct_Inheritance_Link. That is, a super-entity can have only one Direct_Inheritance_Link.

4.2 Comparative Analysis

In this Section, we present the results of the comparative analysis of our metamodel (EERMM) against the metamodels of CWM and IMM. Table 1 summarizes the main points discussed in Sections 3 and 4, and presents the results of our analysis. In this sense, we can see that (i) neither CWM nor IMM support Weak Entity/Relationship, composite attribute, discriminator attribute, and category; (ii) CWM indirectly supports: associative entity, multi-valued attribute, derived attribute, and relationship with attribute; (iii) IMM does not also support Associative Entity, relationship with attribute, multiple inheritance, total/partial inheritance, and disjoint/overlap inheritance; and (iv) Our metamodel covers all EER constructors. In summary, as highlighted throughout this article, the EER metamodels of OMG are clearly not capable of providing a full support to the EER Model according to Chen's notation, such that IMM (the most recent one) only covers 55% of the features shown in Table 1 and CWM (the precursor of IMM), although it can cover until 80% of the features (considering indirect constructions derived from the UML notation), it is a UML profile without OCL rules to assure valid models. That is, given these shortcomings, we have good reasons to propose a new metamodel rather than modify the existing proposals, mainly because we have no guarantees that the work required to define OCL rules or to add the missing features is less complex than the effort required to define EERMM. Furthermore, the modified metamodel can be more complex than EERMM.

5. A PRACTICAL APPLICATION

A metamodel is a specification for builders of CASE tools. That is, a CASE tool that implements a metamodel prevents the specification of invalid models, because it effectively ensures the consistency of its models. In this context, in order to demonstrate the feasibility, expressiveness, and usefulness of our metamodel, we have developed a CASE tool, named EERCASE, and have used it to design

Table I. Comparing CWM, IMM and Our Metamodel

	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%

our metamodel (see Fig. 4). With our CASE tool, the designer can interact with an EER schema by inserting, excluding, editing, visualizing at different zoom levels, and exporting it as a figure or as an XMI (XML Metadata Interchange) file. The CASE tool is implemented using Java/Eclipse technologies (i.e., Eclipse Modeling Framework (EMF) [Steinberg et al. 2009], Graphical Modeling Framework (GMF) [FOUNDATION 2013] and Epsilon Framework [Kolovos et al. 2011], which are conformed to the Essential Meta Object Facility (EMOF) [OMG 2013] standard. Moreover, the tool generates SQL code for PostgreSQL and it can be easily extended to deal with any database management system (new compilers just need to consider the data types and reserved words of the new target SQL/DDL dialects). In Fig. 9 we show an implementation view of our metamodel, which was used to develop our CASE tool. This implementation view is specified in Ecore [Steinberg et al. 2009] (the Java/Eclipse technology used to represent models in EMF/EMOF) and it extends the conceptual view presented in Fig. 4 in order to (i) add eight meta-attributes (datatype, size, isNull, isUnique, check, defaultValue, comment and cardinality) in the Attribute metaclass (because these meta-attributes are useful for code generation or annotation purposes); (ii) add the isIdentifier meta-attribute in the RelationshipLink meta-entity (this meta-attribute allows to set Entity.isWeak = True and Relationship.isIdentifier = True with only one click to define RelationshipLink.isIdentifier = True); (iii) incorporate the enumerations that specify the valid values for an attribute; and (iv) remodel the Associative_Entity metaclass as an inheritance of an Entity and a composition of Relationship (because Java does not support multiple inheritance).

In Fig. 10 we show the Graphical User Interface (GUI) of our CASE tool with a fragment of a hypothetical schema for a manufacturing Company. It is important to point out that our goal is to present a didactic and expressive example that is as simple as possible and that covers all the constructors of the EER Model. In this sense, aiming not to clutter up Fig. 10, we only depict one example of each attribute type. Hence, in this figure we illustrate a weak entity (Computer), an associative entity (Task), regular entities (all entities, except Computer), a composite attribute (address), a multi-valued attribute (phone), a derived attribute (age), a key attribute (id), a discriminator attribute (start_date), an relationship with an attribute (start_date), simple, single-value and stored attributes (all attributes, except these recently listed), a unary relationship (Supervise), a ternary relationship (Work), two binary relationships (Use and Has), roles (supervisor and supervised), participation constraints (total or partial – drawn using relationship links with a double or single line, respectively), cardinality constraints (one or many –depicted using the characters "1" or "N" on relationship links, respectively), inheritances (disjoint, overlap, total and partial – represented using: "d", "o", double line and single line, respectively) and categories (depicted using "u").

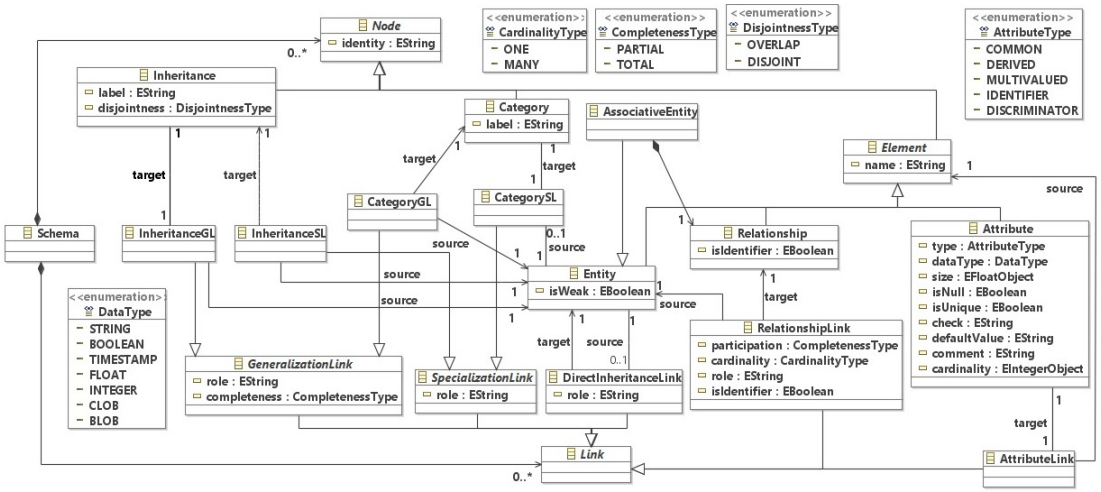


Fig. 9. Implementation view of our metamodel

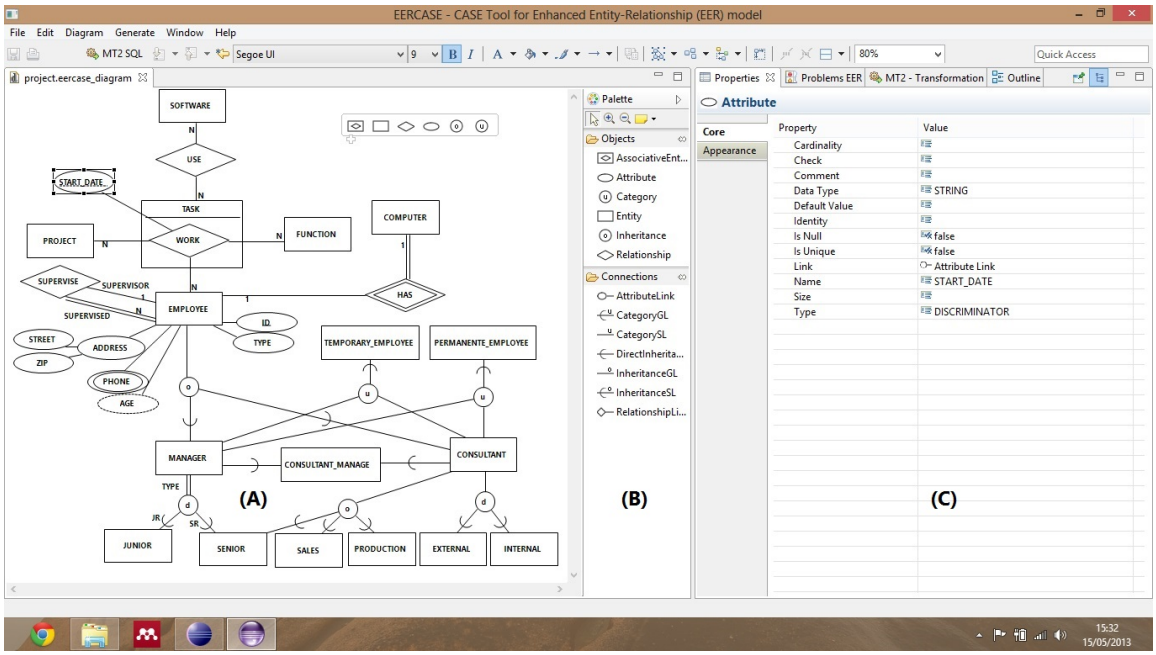


Fig. 10. GUI of our CASE tool with an EER schema using all constructors of the EER Model

Note that the GUI of our CASE tool has a palette (area "B" in Fig. 10) with all constructors of the EER Model. The modeling tasks start with a click on the desired constructor of the EER Model followed by the selection where it will be placed in the drawing area (area "A" in Fig. 10). Next, the designer may edit the properties of the constructor (area "C" in Fig. 10), and add new constructors. Once finished the modeling, the designer may validate its schema using our CASE tool (Menu Edit -> Validate or saving the schema). For example, it can check whether: (i) there are two entities or two attributes (in the same entity) with the same name; (ii) the completeness and disjointness constrains of an inheritance are correctly defined (these constraints can be only defined in an inheritance with at least two sub-entities); (iii) there are isolated nodes; and (iv) a relationship as well as a weak entity have an identifier attribute (in these cases, we must use a discriminator attribute when an identification is needed). These validations are implemented using the Epsilon Validation Language (EVL) [Kolovos et al. 2011]. Once validated, the designer may automatically generate the SQL/DLL code for the modeled schema. The generation code for SQL/DDL scripts is implemented using the Epsilon Generation Language (EGL) [Kolovos et al. 2011]. Due to scope and space limitations, a detailed specification of our CASE tool is not included in this article. However, more information about our CASE tool (including its download link) can be found at http://www.cin.ufpe.br/∽eercase.

6. FINAL REMARKS

Metamodels play a fundamental role in modeling language definition and CASE tool construction, because they describe the constructors and effectively guide the development of CASE tools for modeling languages. That is, a metamodel specifies modeling constructors, their relationships, and their well-formedness rules, preventing the specification of invalid models. Therefore, a metamodel provides facilities for building CASE tools. For example, a CASE tool based on a metamodel can detect errors earlier or even prevent them from happening, guide towards preferable design patterns, check completeness by informing about missing elements, reduce modeling work by applying conventions and default values, support full code generation, and keep specifications consistent. Moreover, a metamodel also can serve as a straightforward option for interchanging the metadata with other tools. Therefore

a metamodel is as useful for a modeling language/CASE tool as a grammar is for a programming language/compiler [Kelly and Tolvanen 2008].

Although the EER Model is one of the most used modeling languages for the conceptual design of database and much work has been done on this model, to the best of our knowledge, there are only two proposals of EER metamodels, and they do not address all constructors of the Chen's notation (see Sections 3 and 4.2). Moreover, neither a discussion about the engineering used for formalizing these metamodels was presented nor a reasoned comparison among them can be found in the literature. For these reasons, in this article we have shown a detailed and practical view of how to formalize the EER Model by means of a metamodel that (i) covers all constructors of this modeling language; (ii) defines well-formedness rules for preventing the modeling of syntactically incorrect EER schemas; (iii) can be used as a starting point (a theoretical framework) by works that aim to build EER CASE tools (e.g., extending or reusing part of our metamodel to address new features or interchanging metadata among tools); and (iv) supports automatic SQL/DDL code generation.

Aiming to specify a technology independent metamodel, we have specified a conceptual view of our metamodel (see Fig. 4) by means of the EER Model (i.e. a reflexive metamodel). Furthermore, we also present an implementation view of our metamodel (see Fig. 9). This implementation view was modeled according to Ecore/EMOF technologies and was employed to build our CASE tool, which was used to model the conceptual view of our metamodel.

In order to show the feasibility, usefulness, and expressiveness of our metamodel, we have used our CASE tool to model a simple example that covers all EER constructors, confirming that our metamodel is feasible and more expressive than related ones (cf. Sections 3 e 4.2). Our CASE tool was implemented in Java using the EMF, GMF, and Epsilon framework. In its current version, it generates SQL/DDL code for PostgreSQL. In future work we plan to cover other database management systems as well. We also plan to extend our metamodel to support spatial and temporal data modeling, perform a usability test of our CASE tool, develop reverse engineering modules and specify mathematical formalizations/proofs to show the correctness of our metamodel and its well-formedness rules.

In conclusion, our work contributes to advance the state of the art of metamodels for the EER Model, because it provides a novel perspective of scientific research and industrial application in the field of database conceptual modeling, since it pinpoints current limitations and shows a more expressive way to define an EER metamodel. In other words, our metamodel is the first one to covers all elements of the Chen's notation, which is particularly important because it defines the basis for the EER Model. That is, our metamodel defines a set of statements that must not be false for any valid EER Model.

REFERENCES

ACM/IEEE. Computer Science Curriculum 2008: an interim revision of CS 2001. http://www.acm.org/education/curricula/ComputerScience2008.pd, 2008.

Amalfi, M., Artale, A., Calì, A., and Provetti, A. Generating Preview Instances for the Face Validation of Entity-Relationship Schemata: the acyclic case. In *Proceedings of International Conference on Database Systems for Advanced Applications*. Hong Kong, China, pp. 225–234, 2011.

Badia, A. and Lemire, D. A Call to Arms: revisiting database design. *SIGMOD Record* 40 (3): 61–69, 2011.

Bavota, G., Gravino, C., Oliveto, R., De Lucia, A., Tortora, G., Genero, M., and Cruz-Lemus, J. A. Identifying the Weaknesses of UML Class Diagrams During Data Model Comprehension. In *Proceedings of International Conference on Model Driven Engineering Languages and Systems*. Wellington, New Zealand, pp. 168–182, 2011.

Bollati, V. A., Atzeni, P., Marcos, E., and Vara, J. M. Model Management Systems vs. Model Driven Engineering: a case study. In *Proceedings of Annual ACM Symposium on Applied Computing*. Trento, Italy, pp. 865–872, 2012.

Calì, A., Gottlob, G., and Pieris, A. Query Answering Under Expressive Entity-Relationship Schemata. In *Proceedings of International Conference on Conceptual Modeling*. Vancouver, BC, Canada, pp. 347–361, 2010.

Chen, P. P.-S. The Entity-Relationship Model - toward a unified view of data. *ACM Transactions on Database Systems* 1 (1): 9–36, 1976.

Combi, C., Degani, S., and Jensen, C. S. Capturing Temporal Constraints in Temporal ER Models. In *Proceedings of International Conference on Conceptual Modeling*. Barcelona, Spain, pp. 397–411, 2008.

Connolly, T. M. and Begg, C. E. *Database Systems: a practical approach to design, implementation and management*. Addison-Wesley Publishing Company, USA, 2009.

Dullea, J., Song, I.-Y., and Lamprou, I. An Analysis of Structural Validity in Entity-Relationship Modeling. *Data & Knowledge Engineering* 47 (2): 167–205, 2003.

Elmasri, R. and Navathe, S. *Fundamentals of Database Systems*. Addison-Wesley Publishing Company, USA, 2010.

FOUNDATION, E. Graphical Modeling Framework. http://www.eclipse.org/gmf/, 2013.

Franceschet, M., Gubiani, D., Montanari, A., and Piazza, C. From Entity Relationship to XML Schema: a graph-theoretic approach. In *Proceedings of International XML Database Symposium on Database and XML Technologies*. Lyon, France, pp. 165–179, 2009.

Garcia-Molina, H., Ullman, J. D., and Widom, J. *Database Systems: the complete book*. Prentice Hall Press, USA, 2008.

Hartmann, S. Reasoning About Participation Constraints and Chen's Constraints. In *Proceedings of Australasian Database Conference*. Adelaide, Australia, pp. 105–113, 2003.

Irani, P., Tingley, M., and Ware, C. Using Perceptual Syntax to Enhance Semantic Content in Diagrams. *IEEE Computer Graphics and Applications* 21 (5): 76–85, 2001.

Jones, T. H. and Song, I.-Y. Analysis of Binary/Ternary Cardinality Combinations in Entity-Relationship Modeling. *Data & Knowledge Engineering* 19 (1): 39–64, 1996.

Karanikolas, N. N. and Vassilakopoulos, M. G. Conceptual Universal Database Language: moving up the database design levels. In *Proceedings of East European Conference on Advances in Databases and Information Systems*. Riga, Latvia, pp. 330–346, 2009.

Kelly, S. and Tolvanen, J.-P. *Domain-Specific Modeling: enabling full code generation*. Wiley-IEEE Computer Society Pr, 2008.

Kolovos, A., Rose, L., García-Romínguez, A., and Paige, R. *The Epsilon Book*. Eclipse.org, 2011.

Lucia, A. D., Gravino, C., Oliveto, R., and Tortora, G. Data Model Comprehension: an empirical comparison of ER and UML class diagrams. In *Proceedings of IEEE International Conference on Program Comprehension*. Netherlands, pp. 93–102, 2008.

Lucia, A. D., Gravino, C., Oliveto, R., and Tortora, G. An Experimental Comparison of ER and UML Class Diagrams For Data Modelling. *Empirical Software Engineering* 15 (5): 455–492, 2010.

Motta, G. and Pignatelli, G. From Strategic to Conceptual Information Modelling: a method and a case study. In *Information Technology and Innovation Trends in Organizations*, A. D'Atri, M. Ferrara, J. F. George, and P. Spagnoletti (Eds.). Physica-Verlag HD, pp. 179–186, 2011.

OMG. Common Warehouse Metamodel (CWM) Specification Volume 2 - extensions, organization. http://www.omg.org/spec/CWM/, 2001.

OMG. Common Warehouse Metamodel (CWM) Specification. http://portals.omg.org/imm/, 2003.

OMG. Information Management Metamodel (IMM) Specification Volume2 - business modeling. http://www.omgwiki.org/imm/doku.php, 2009.

OMG. Essential Meta-Object Facility (EMOF). http://www.omg.org/spec/mof/2.4.1/pdf, 2013.

Rumbaugh, J., Jacobson, I., and Booch, G. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.

Silberschatz, A., Korth, H., and Sudarshan, S. *Database System Concepts*. McGraw-Hill Education, 2010.

Song, I.-Y. and Chen, P. Entity Relationship Model. In *Encyclopedia of Database Systems*, L. Liu and M. T. Ozsu (Eds.). Springer US, pp. 1003–1009, 2009.

Song, I.-Y., Evans, M., and Park, E. K. A Comparative Analysis of Entity-Relationship Diagrams. *Jornal of Computer and Software Engineering*, 1995.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. *EMF: eclipse modeling framework 2.0*. Addison-Wesley Professional, 2009.

Ware, C. and Bobrow, R. Motion to Support Rapid Interactive Queries on Node–Link Diagrams. *ACM Transactions on Applied Perception* 1 (1): 3–18, 2004.

Xu, C., Liang, P., Wang, T. G., Wang, Q., and Sheu, P. C.-Y. Semantic Web Services Annotation and Composition Based on ER Model. In *Proceedings of IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing*. USA, pp. 413–420, 2010.

Zhang, F., Ma, Z. M., Lv, Y., and Wang, X. Formal Semantics-Preserving Translation from Fuzzy ER Model to Fuzzy OWL DL Ontology. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. Australia, pp. 503–509, 2008.