An ontology-driven unifying metamodel of UML Class Diagrams, EER, and ORM2

C. Maria Keet[a,*], Pablo Rubén Fillottrani[b,c]

[a] *Department of Computer Science, University of Cape Town, Cape Town 7701, South Africa. Tel: +27 (0)21 650 2667*
[b] *Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina*
[c] *Comisión de Investigaciones Científicas, Provincia de Buenos Aires, Argentina*

Abstract

Software interoperability and application integration can be realized through using their respective conceptual data models, which may be represented in different conceptual data modeling languages. Such modeling languages seem similar, yet are known to be distinct. Several translations between subsets of the languages' features exist, but there is no unifying framework that respects most language features of the static structural components and constraints. We aim to fill this gap. To this end, we designed a common and unified ontology-driven metamodel of the static, structural components and constraints in such a way that it unifies ER, EER, UML Class Diagrams v2.4.1, and ORM and ORM2 such that each one is a proper fragment of the consistent metamodel. The paper also presents some notable insights into the relatively few common entities and constraints, an analysis on roles, relationships, and attributes, and other modeling motivations are discussed. We describe two practical use cases of the metamodel, being a quantitative assessment of the entities of 30 models in ER/EER, UML, and ORM/ORM2, and a qualitative evaluation of inter-model assertions.

Keywords: Conceptual Modeling, Ontologies, Database semantics, Metamodeling, EER, UML Class Diagrams, ORM

*Corresponding author
URL: mkeet@cs.uct.ac.za (C. Maria Keet), prf@cs.uns.edu.ar (Pablo Rubén Fillottrani)

1. Introduction

Recent upscaling scientific collaboration in the life sciences [1] and pharmacology[1], launching and monitoring of e-government initiatives [2, 3], company mergers [4], integration frameworks [5], and a broader adoption of Semantic Web technologies require complex software system development and information integration from heterogeneous sources. While at the implementation and data level, markup languages and standards help interoperability (e.g., SBML [6], BEL [7], GML [8], RDF [9]), the design of such systems typically involves a data analysis stage with the design and linking or integration of conceptual models for the implementation-independent representation of the data requirements. The various extant conceptual data modeling (CDM) languages, such as UML [10], EER [11], ORM [12], MADS [13], and Telos [14], each have their strengths and typically multiple models represented in different languages are used for one system. Therefore, establishing connections between these conceptual models has become an important task. A motivating use case to illustrate this is described in the following example, which describes one of the problems.

Example 1. Consider the interoperability scenario between an EER diagram and a UML class diagram where both are to be maintained as far as possible, which is depicted in Figure 1. The EER diagram depicts a rudimentary model for a database of a generic termbank, which assumes it to be valid for all 11 official languages in South Africa. The UML diagram focuses on specific aspects of isiZulu terms for an isiZulu termbank that also requires some application layer processing due to its agglutinative characteristics. These diagrams are stylized and simplified from the actual model that is being developed for the isiZulu termbank at the University of KwaZulu-Natal [15], so as to illustrate the case rather than present models that span a few pages. For instance, to record *umuntu* (person/human) in the termbank, linguists need to have separated out the stem (*ntu*), the root (*nt*), suffix (*u*), prefix (*umu*) and pre-prefix (*u*), and besides grammatical number (singular/plural/mass) also the noun class (there are 17—*umuntu* belongs to class 1). This enables posing queries such as "retrieve all terms with root 'nt' " or "retrieve all nouns of class 1a", and computing various combinations in the business layer, as normally isiZulu terms are ordered alphabetical by stem,

[1]See for example `http://www.tipharma.com` and `http://www.phuse.eu`.

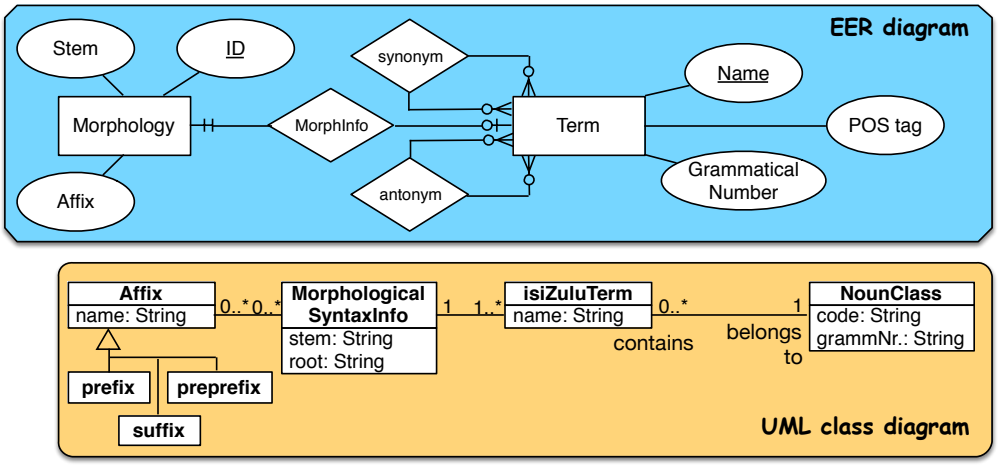


Figure 1: Example of an integration scenario with two diagrams in different languages (names for the two left-most associations omitted due to width limitations).

not full term, where the full term is computed using the affixes of the noun class.

Going by terms in the two models, there clearly are similarities, but it is unclear how to link them such that the overall model and their associated data remains consistent. We return to this problem in Section 4.2, where the linking is assessed and resolved. ♢

'Traditional' information systems development and management exhibit the capability of linking models represented in different languages only at the physical schema layer [16] or only for conceptual models represented in the same CDM language [17, 18]. Some results are available on transformations between CDM languages (e.g., [19, 20, 21]), but they cover only a subset of all the languages' features, as subtle representational and expressive differences in the languages make this task very difficult. Consequently, current tools offer only very limited functionality in linking or importing models represented in one language into one represented in another language.For example, mandatory and disjointness is catered for, but weak entity types, identification, or attributes are not [16, 20, 22].

Any differences between the main CDM languages—UML Class Diagrams, ER, EER, ORM, and ORM2—may seem merely terminological, but it is known not to be the case either from a metamodeling viewpoint [23] nor

even within the same family of languages [24]. Also, what may seem very different at first glance may actually be the same, or at least share part of their meaning. This concerns possible agreements or differences in ontological foundation or commitment among the CDM languages. However, the state of the art in this area has focused primarily on a single CDM language and only for UML and ORM (e.g., [24, 25]), and across languages only at the level of a few illustrations [26]. Put differently, it is unknown to what extent the languages agree on their underlying principles for modeling information. This is an obstacle for mapping and transformation algorithms in CASE tools to let one work in parallel on conceptual data models represented in different languages that otherwise could be highly useful in information integration and complex system development. In addition, from a cognitive viewpoint, a more precise insight in the commonalities and differences in the underlying modeling principles will contribute to the understanding of the extent to which the language features affects modeling information to tools, methods, and methodologies for CDM development and maintenance.

An approach toward solving these issues is to devise a comprehensive formalization of the languages so as to manage their interaction. To be able to do so, however, it first should be clear what entities and constraints exist in each language, how they can be used in the language, and how the differences can be reconciled without changing the languages. Put differently, not a comparison of the respective metamodels, but a *single integrated metamodel* inclusive of all language features is needed, so that one can unify the CDM languages and design transformation algorithms at the conceptual layer in software and database development. We designed such a unifying metamodel for the static, structural components and constraints of UML 2.4.1 class diagrams, EER, and ORM2/FBM, which is, to the best of our knowledge, the first of its kind. The metamodel is ontology-driven in the sense that our arguments and modeling decisions use motivations taken from the philosophical concept of Ontology, and ontologies in Computer and Information Sciences. At this stage, we are not concerned with the argument of convenience to fit with an *a priori* chosen logic language.

The unification makes clear the differences between and commonalities of the selected CDM languages. Notably regarding the entities in the languages, they agree only on Relationship (association) and Subsumption, Role (/association end/relationship component), and Object type (/class/entity type) and they all adhere to the positionalist ontological commitment for the meaning of relationship. They have different approaches or 'incomplete' coverage of

certain other features, however, such as attributes and weak entity types. Also for constraints there is a small intersection of shared language features out of the 49 constraints, being Mandatory, Cardinality, Single identification, Disjointness and Completeness, and Subset constraints. Two use cases illustrate concrete usefulness of the metamodel: one focuses on the classification of entities of 30 conceptual models into metamodel entities and the second one reconsiders the two conceptual models from Example 1 and examines their possible inter-model assertions, given the fine-grained semantics of the metamodel.

The main contributions of the paper can be summarized as follows:

- The unifying metamodel of the three main language 'families', being UML Class Diagrams, ER and EER, and ORM and ORM2;
- The insight in the language features thanks to the ontology-driven approach of the analysis;
- The demonstration that there is, in fact, little overlap in actual language features across the language families;
- Taken together, they constitute a solid theoretical foundation for CASE tool development that supports all three language families, notably facilitating inter-model assertions across models represented in different languages and for converting a model in one language into another.

We discuss related work in Section 2, describe the metamodel in Section 3, and demonstrate a selection of its use and usefulness in Section 4. We discuss it in Section 5 and conclude in Section 6.

2. Related Work

There are different strands of research in different subfields that focus on multiple CDM languages. We discuss systems-directed approaches, knowledge representation ones, and compare it with our approach.

2.1. Systems-directed Approaches

A useful step before unifying CDM languages is a comparison of the languages through their metamodels (in ORM) that highlights their differences [23]. In this section, however, we focus on the research on linking and unifying models.

A partial unification was designed by Venable and Grundy [21] and implemented in MViews [27] and Pounamu [28]. They developed the metamodel

in the CoCoA graphical language and it covers a fragment of ER and a fragment of NIAM (which is a precursor to ORM), as they omitted, mainly, value types, nested entity types, and composite attributes. In addition, NIAM is forced to have the attributes as in ER in the 'integrated' metamodel, even though attributes are treated differently in NIAM (for a discussion, see Section 3.3). Consequently, Venable and Grundy's "dynamic" ad hoc mappings are limited.

Bowers and Delcambre [16] introduce a framework for representing schema and data originating from several data models, being mainly relational, XML and RDF. Its principal feature is a flat representation of schema and data, and the possibility of establishing different levels of conformance between them. However, its representation language ULD does not cover all language features of UML, EER, and ORM, for it includes only ordinal, set and union class constructs, and cardinality constraints.

Boyd and McBrien [20] developed the Hypergraph Data Model to relate models represented in ER, relational, UML, and ORM, and they present transformation rules between them. The advantage of using graphs as intermediate representation is that it has a simple irreducible form for schemas that can be used to prove schema (model) equivalence. Assessing the features that are covered, we note that the representational language includes inclusion, exclusion and union class constructs, and mandatory, unique and reflexive constraints. The combination of these types of constraints gives a language to express basic cardinality constraints and arbitrary keys, but it omits roles, aggregation, and weak entity types and several constraints, such as value comparison constraints, general frequency constraints, and most ring constraints.

Atzeni et al. [17, 19] describe an automatic approach for translating a model from one language to another, considering ER, UML and physical schema languages. They achieve this by, first, specifying a "supermodel" that is a dictionary of constructs, and those "metaconstructs" are used to characterize different models, which are entities (called "abstracts"), attributes (called "lexicals"), relationships, generalization, foreign keys, and complex attributes, covering nine constructs overall. Subsequently, automatic translations between schemas are produced in the Datalog language, but translations from a rich representational language, if possible, may require a sequence of such basic translations.

There are other works that, based upon title and abstract, may seem relevant, but upon closer inspection addresses a different problem than the one

we are aiming to solve. For instance, consider the framework developed by Thalheim [29] for modeling layered databases, possibly integrating databases in different paradigms, such as OLAP systems and streaming databases: we focus only on standard databases and object-oriented applications described with CDM languages, hence, this type of database modeling is out of scope.

2.2. Knowledge Representation Approaches

There are several research projects to solve the issues from a knowledge representation viewpoint, which typically have as ultimate aim automated reasoning over conceptual data models. This can be divided into efforts in logic-based reconstructions of conceptual models and reasoning [30, 31, 32, 33, 34, 35], which can be considered as a prerequisite for unification, and comparisons and unifications [22, 36]. The typical approach is to choose a logic based on a broader research program or desired expressiveness and computational complexity, and then to show it captures 'sufficiently' one or more of the CDM languages. Due to these varying motivations, different logics have been used for different CDM languages and even for one CDM language, therewith still not providing the sought-after interoperability for either of the languages or among each other. For instance, the Description Logic (DL) language $\mathcal{ALUNI}$ was chosen as basis for a partial unification some time ago [22], but other languages are used for the formalization, such as *DL-Lite* and $\mathcal{DLR}_{ifd}$ [30, 31], which are also incomplete regarding the features they support. Complicating matters is that ORM has features that render the language undecidable [37], so various FOL [32, 33], DL [38, 39], and system-oriented logic-based reconstructions [40] are available. This being the case, it is not possible to simply link them and implement it—some of the languages used are not even implemented yet at all (e.g., $\mathcal{DLR}_{ifd}$ and $\mathcal{ALUNI}$). Once a comprehensive metamodel with all existing features is available, they could be either formalized in one logic, or possibly the different logics (if implemented) could be orchestrated by means of the Distributed Ontology Language [41] and system that is in the process of standardization (`http://ontoiop.org`).

2.3. Our approach

We take a different approach toward unification compared to the related works, notably regarding scope and methodology. First, we aim to capture *all* constructs of the languages under consideration and *generalize* in an ontology-driven way so that the integrated metamodel subsumes the structural, static

elements of EER, UML Class Diagrams v2.4.1, and ORM2. Thus, such an integrated metamodel has as *fragments* the respective metamodels of EER, UML Class Diagrams v2.4.1, and ORM2, therewith leaving the base languages intact. We include entities, objects, roles, relationships, attributes in their various forms, data types, value types, and several class constructs, and the constraints. This very broad variety of elements covers nearly all of the elements in UML, EER and ORM, as shown in the Appendix; it omits temporal aspects (a frozen attribute) and derived constraints. None of the related works includes roles, aggregation, and relationship constraints, thus only limited subsets of UML or ORM are covered. In underlying idea, it may seem the same as the supermodel dictionary of Atzeni et al. [19], so one could try to extend their MIDSTool. However, we not only include more constructs and ORM, but also model the relations and constraints between them—i.e., a real 'super'/metamodel, not a list of constructs. The main advantage is the richer, finer-grained semantics to determine real equivalence and *de facto* approximations. Other advantages are that part of what otherwise would have to be encoded in various algorithms is now accessible in one place, and that the metamodel can also be used to check whether the models that are linked or transformed were adhering to the right syntax.

Methodologically, the metamodel we propose in this paper is ontological rather than formal, which is in contrast to all other known works that present first a formal common language for translations that leave aside important particular aspects of each language. Concerning this ontology-driven approach, ontology-driven conceptual modeling can refer to i) logic-enhanced and reasoner-enhanced conceptual modeling, ii) using a (section of a) domain ontology in one's model, iii) adapting a domain ontology into a conceptual model for a specific application scenario, and iv) using insights from Ontology and ontologies to enhance the quality of a conceptual data model. For the metamodel we propose, we use principally the latter aspect of the ontology-driven approach. This comprises understanding better the modeling language and to improve the modeling of some particular aspect of a universe of discourse [42, 43, 44, 45]. For instance, using the Unifying Foundational Ontology (UFO) to analyze and extend the UML 2.0 metamodel Sortal Class in [25, 46], refining shared and composite aggregation in [47], and looking into the nature of relationships in [24, 48]. Furthermore, with respect to a specific domain, it can be used to model DNA [49] and catalytic reactions [36]. We use such insights to analyze the underlying commonalities and differences between the chosen CDMs as well as to enhance the quality of

the metamodel. Only after having developed a conceptual model of all possible entities, constraints, and their relations in the selected languages, we devised a comprehensive formalization in FOL [50] that is to be used for their translations. The main benefit of this methodological approach is that it allows one to have a comprehension of the scope and the meaning(s) of each entity in each language whilst coping with the larger amount of elements. Such an overview is an essential step towards achieving the full potential of information sharing through their respective conceptual models.

3. Metamodel

In the strict sense, what we present here is an integrated conceptual model about the entities and constraints in the selected modeling languages, in the literature referred to typically as a *metamodel*. This metamodel covers all their native features and is consistent. At this stage, it is not the purpose to examine whether a particular language feature is a good one or how one can make it better by using some ontological principles; we aim at representing in a unified way what is present in the language. In order to achieve this, we use several notions from Ontology (philosophy) and ontologies (artificial intelligence) so as to increase understanding of the language features, to reconcile or unify perceived differences, and to improve the quality of the metamodel. Note that this does not turn the metamodel into an ontology, for its scope is still just the selected modeling languages, not a logic-based representation about all CDM languages past, present, and future.

The entities (in the general sense) are shown hierarchically in Figure 2 and the constraints in Figure 3 in UML Class Diagram notation. A white fill of a class icon means that that entity is not present in either of the languages, a light grey fill means that it is present in one language, dark grey that it is present in two, and a black fill that it is present in all three families of languages (EER, UML v2.4.1, ORM2). The constraints included are only those that are explicitly available in the language as graphical or textual constraint in the diagram. Naming conventions and terminological differences and similarities of the entities are listed in the Appendix.

We describe several salient aspects of the metamodel and explain and motivate its contents for object types (Section 3.1), roles and relationships (Section 3.2), attributes and value types (Section 3.3), and constraints (Section 3.4).

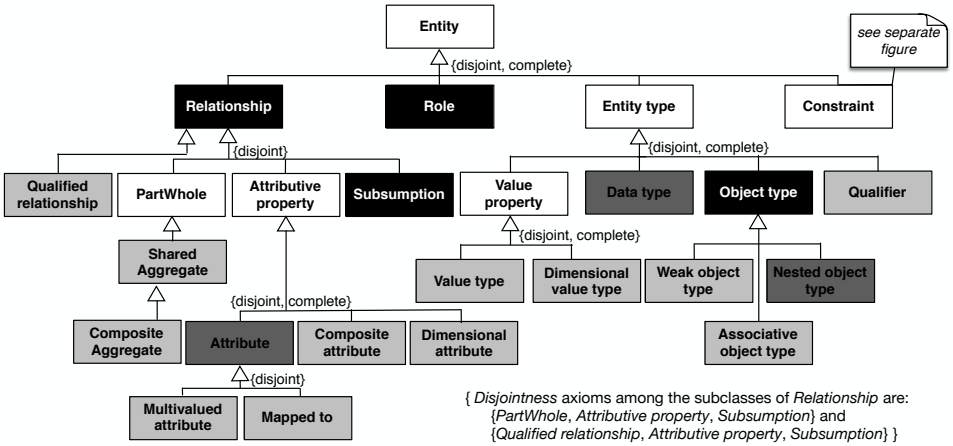


Figure 2: Principal static entities of the metamodel.

3.1. Object Types

One of the principal entities is what we call Object type. The CDMs refer to this as *entity type*, *object type*, and *class*, and, informally, also *concept*. Philosophically, they may be referring to the same kind of thing even though clear distinctions are made between *classes*, *concepts*, *properties*, and *universals* in Ontology, and *types* and *unary predicates* in the field of mathematics. *Class* concerns set theory and its extension (a set of actual objects as members), and two distinct classes must have different extensions. *Concept* generally refers to a mind-dependent entity that may, or may not, have individuals instantiating it [51]. *Properties* are "also called 'attributes,' 'qualities,' 'features,' " [52], which can be relational, e.g., the property of being married, or unary, e.g., a color or a shape. *Universal* generally refers to a mind-independent entity that has at least one instance and it also uses the instantiation relation [53]. *Entity* can mean whatever can be inferred from the context in which the term is used, though in any case, that thing it refers to is a discrete unit; hence, 'entity' can refer to an instance or object, or to a universal, concept, or class. For some refinements specifically tailored to CDM, see, e.g., [45]. Applying these notions to the relevant elements in UML, EER, and ORM, the following can be observed. *Class* with its set-orientation focuses on the extensional aspect and grouping similar things together, whereas *concepts* and *universals* focus on the intension. Informally, the latter are the descriptions, or a combination of properties, that those

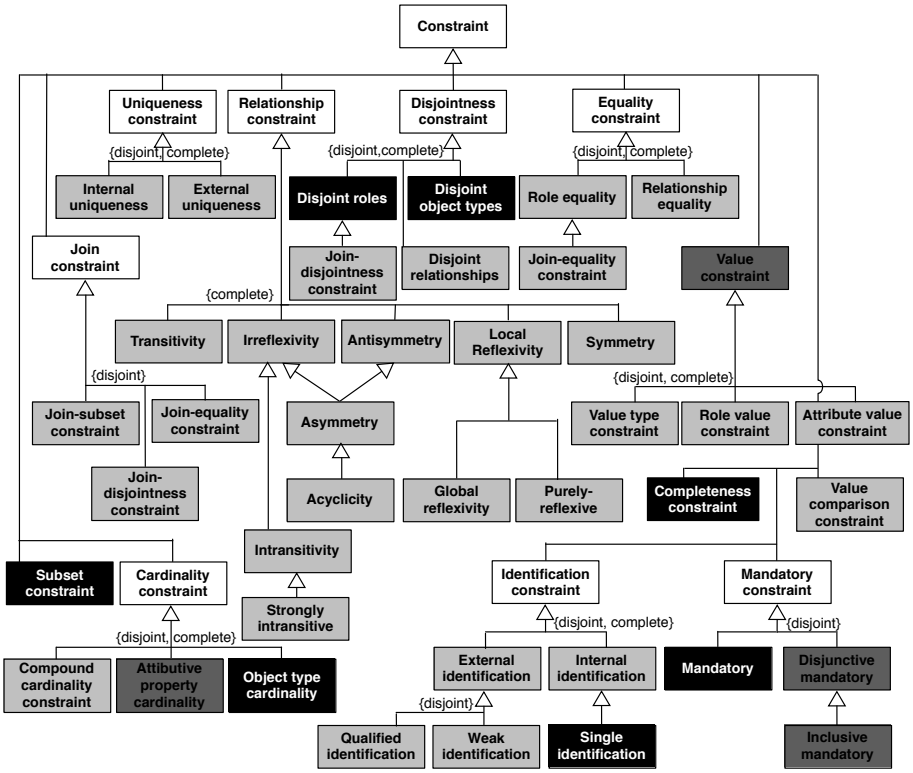


Figure 3: Unified hierarchy of constraints in the metamodel.

entities that instantiate them have; e.g., Orange is a fruit that has a Shape that is spherical and has a Color that is green or orange.

With respect to ORM, EER, and UML, it could be that because these CDM language use different terminology, they may have different ontological commitments. The descriptions in the literature [10, 11, 54, 55] are varied. For instance, the UML standard assumes the intensional notion, despite calling it *class*: "The purpose of a class is to specify a classification of objects and to *specify the features* that characterize the structure and behavior of those objects." (p. 49) (emphasis added) [10], whereas ORM has mixed descriptions. While the original ER model uses *entity sets*, "we know that it has *the properties common* to the other entities in the entity set" (p.11) [55] and, more clearly in [11]: "Entity types conceptualize structuring of things of reality through attributes.". Despite some differences in formulation, it does not make a real difference in the model: each one is used to denote a

kind of thing where the relevant features are described, i.e., the intension, and it will have an extension in the software as objects in object-oriented software or tuples in a database table. Each of those objects is assumed to represent an instance in reality, although one could design a database about mind-dependent deities in the Stone Age. From an Ontology viewpoint, we thus can postulate that the CDM's *entity type* or *class* is mostly a universal, and occasionally may be a concept, and the term we use henceforth for those entities in CDMs that describe the intension is Object type, which does not clash in intention with terminology in Ontology.

Object type has two subtypes: Weak object type and Nested object type. Weak object type represents ER and EER's *weak entity type* (see also Section 3.4). Its ontological status may not be fully clear [25, 56], but it is certainly an object type. This might seem less clear for Nested object type, or *association class*, *associative entity*, or *objectified fact type*. Literature on CDM languages discusses the idea of a "duality" of a nested object type as being both a relationship and an object type, and therewith suggesting a multiple inheritance of Nested object type to two supertypes, Relationship and Object type (e.g., [46]). This is not correct, for it is 'composed of' a relationship or at least the *outcome* of a reification of a relationship, which is distinct from *being* a relationship. Therefore, the metamodel relates them through a normal association (see Figure 4). The constraints for nested object types are very basic so as to reflect the flexibility concerning reification/objectification in UML and ORM: the UML standard does not mention any restrictions for objectifying an association into an association class [10], and earlier restrictions for objectification in ORM [12] have been lifted [57].

Example 2. Casting the elements of Figure 1 in terms of the metamodel introduced so far, Morphology and Term in the EER diagram, and Affix, Morphological SyntaxInfo, isiZuluTerm, and NounClass in the UML Class Diagram are all categorized as Object type. There are no Weak object types and Nested object types in the two models. ♢

3.2. Relationships

As Figure 2 shows, there are four main types of relationship: subsumption, part-whole relations, attributive properties, and qualified relationships, with their carefully crafted disjointness axioms. Much can be discussed about relationships, but we restrict ourselves to their nature and definition (not the

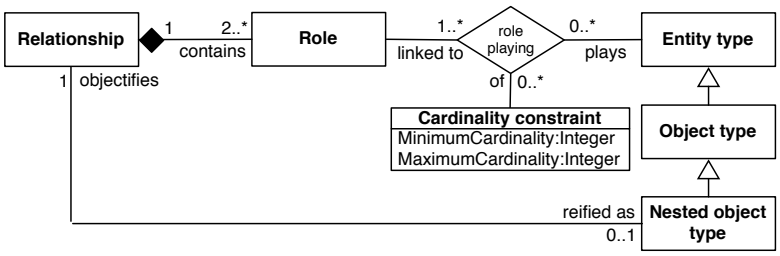


Figure 4: Relationships between Relationship, Role, and Entity type; see text for details.

possible types, as in [48]), and differences among them and with an object type. We discuss separately afterwards qualified relationships and attributes.

A relationship, also called a relational property in Ontology [52], is an entity that relates entities and thus it requires at least two entities to participate in it. This is in contrast to an entity type that is a thing on its own, be it independent or dependent on another entity. From this basic distinction, it follows that there are no unary relationships. The second difference between relationship and entity type is due to the existence of, and their relation to, *roles*, which are called *association ends* or *member ends* in UML [10], *roles* in ORM and fact-based modeling [12, 32, 54], and *components* of a relationship in EER [11, 55] (although on a quick reading, Chen's ER might seem to permit both). An object plays a role in a relationship and, thus, a relationship is composed of at least two roles. This forms part of the characterization of what Relationship is, and it assumes a commitment to the *positionalist* stance as to the nature of relations and relationships (see [58] for a good overview, which has been applied to ORM in [24]). All three CDM languages have roles, which, from an information modeling viewpoint at least, means that they form a part of the so-called 'fundamental furniture of the universe'. Thus, roles are ontologically distinct from both entity types and relationships. Therefore, they appear in the metamodel as separate entities (recall Figure 2), and Relationship, Role, and Entity Type are disjoint. The interaction between them is depicted in Figure 4. It follows also that roles will have to be first-class citizens in a formalization.

There are three points that deserve some attention still, of which we discuss two here. First, the relationship between Role and Relationship in the metamodel is a composite aggregation association, but ontologically this may be an underspecification, for one perhaps could argue that a relationship is even *defined* by its role-parts. Second, there is a ternary role playing between

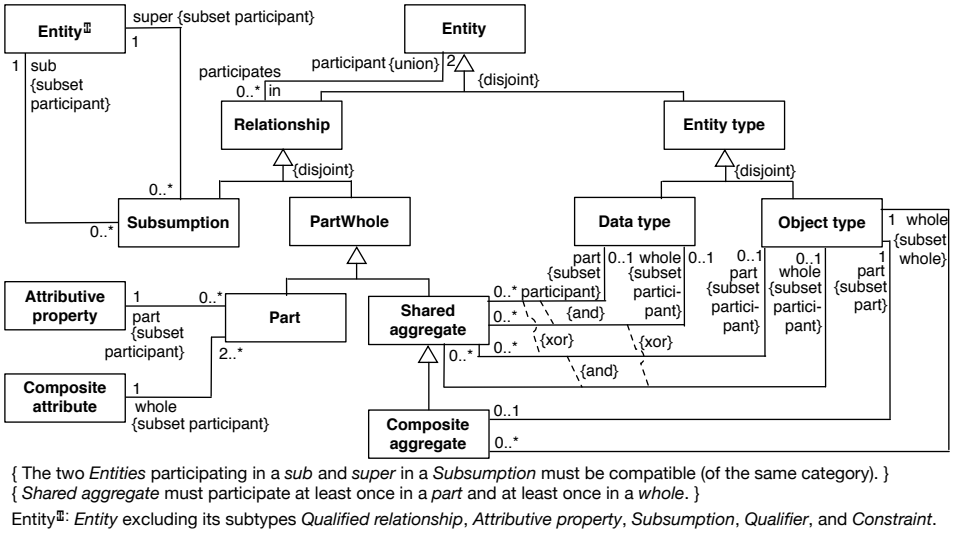


Figure 5: Subsumption and aggregation.

Role, Entity type, and Cardinality constraint, which captures that each role must have exactly one entity type with at most one cardinality constraint, and each entity type may play zero or more roles with or without declared cardinality constraint. The third issue has to do with predicates and is deferred to the discussion in Section 5.

Example 3. Casting the elements of Figure 1 in terms of the metamodel introduced so far, MorphInfo, Antonym, and Synonym in the EER diagram are categorized as Relationship. The association ends in the UML Class Diagram for the associations between Affix and Morphological SyntaxInfo and between Morphological SyntaxInfo and isiZuluTerm are omitted (due to width limitations in drawing), whereas the association ends contains and belongs to are categorized as Role in terms of the metamodel. ♢

3.2.1. Subsumption Relationships

Regarding types of relationships, we have to address subsetting and subtyping of relationships, and aggregation.

Unconstrained subsumption of object types is included in all CDM languages except ER. There are some arguments against multiple inheritance [59, 60], but this is driven by limitations of the OBO-language and/or that subtypes must always be disjoint (an Aristotelian left-over). Neither holds

necessarily for an arbitrary subject domain and, moreover, CDM languages permit it; therefore, the constraints in the metamodel reflect this. Furthermore, because, say, a role subsuming an entity type results in an inconsistency due to the disjointness, it makes sense to add a constraint "For each sub participating in a Subsumption, there must be a matching super and its participating Entity is of the same category as the Entity participating in the sub". However, this exclusion or type compatibility is not specified for the languages, and therefore is not in the metamodel.

There is more to say about subsumption for relationships and roles. The UML 2.4.1 standard distinguishes between *subsetting* where the association ends and/or participating classes are sub-ends/sub-classes of those participating in the super-association or indirectly through an association's attributes, and *specialization of associations* [10]. This specialization is not set-oriented, due to the differences in intension of the association [10], although the UML standard does not describe how that is supposed to work. The only option to change an association's intension is to restrict the relational properties of an association, in analogy to intensional subsumption in ontologies [61]. For instance, each relationship that is asymmetric is also irreflexive. There are only a few such subsumptions, which are depicted in Figure 3 based on the latest hierarchy [62], and little is known about its practicality other than the experiments reported in [61] for ontologies. Nevertheless, it may become relevant in the near future, and therefore this subsumption is covered with Subsumption as well. Finally, both UML and ORM include subsumption of roles, so that the participating entities for Subsumption are Entity.

3.2.2. Aggregation Relationships

A lot of literature is available on UML's *shared* and *composite aggregation* (among many, [25, 47, 63]), yet the UML standard still does not offer clarity on what they really are. Shared aggregation is generally matched to parthood and composite aggregation to proper parthood, but aggregation is also used for meronymic associations in UML Class Diagrams, such as *member-of* (see [47] for an overview). Therefore, there is no subsumption relation between the aggregations and parthood in the metamodel and there is no disjointness axiom on the subtypes of PartWhole. The UML standard's description of aggregation also includes behavioral characteristics or lifecycle semantics of the part and whole, which is not included in the metamodel because it is beyond the scope of the static components that we focus on here.

In addition to the *shared aggregation* and the stricter cardinality constraint for *composite aggregation*, we use a part-whole relation for ER/EER's Composite attribute, which has as part (but not proper part) Attributive property. The cardinality constraints are obvious from the whole-side—it is only composite if there is more than one attribute part of it—whereas from the part-side, we carry through the sharability of an attribute (see also next section), hence, the 0..*.

Example 4. Casting the elements of Figure 1 in terms of the metamodel introduced so far, the EER diagram does not have any subsumption, whereas the UML Class Diagram has three subclasses of Affix, being Subsumption relationships (in terms of the metamodel) with prefix, suffix, and preprefix. There is no Shared aggregate and no Composite aggregate. ♢

3.3. Attributes and Value types

From a logic-based perspective, the definition of an attribute is clear, as given in Definition 1; e.g., an attribute hasColor that relates an object type to a string: hasColor $\mapsto$ Flower $\times$ `String`.

Definition 1. *An* attribute *(A) is a binary relationship between a* Relationship *or* Object type *($R \cup E$) and a* Data type *(D), i.e., $A \mapsto R \cup E \times D$.*

However, this is not straightforward for CDM languages, and ontologists have various theoretical frameworks to deal with them (albeit only partially; see [64, 65]). We describe what Ontology and ontologies have to say about their counterpart to attributes, how it is included in UML, EER, and ORM, the notion of an optional dimension of an attribute value, and how attributes are represented in the metamodel and why.

3.3.1. Attributions in Ontology and ontologies

Object type, Value type, and Attributive property are disjoint either directly or through their parent (Figure 2), which not only reflects their meaning in the CDM languages but also can be motivated by Ontology and ontologies. Provided one accepts the existence of properties, which the CDM languages do, Ontology distinguishes between multiple types of properties [52]. Ontology's *sortal property*, or 'sortal' for short, refers to those things where one distinguishes, classifies, and identifies objects and they can exist on their own; e.g., Apple and Person. Ontology's *attribution* or attributive property, also called *quality*, such as the Color of an apple or its Shape, require a bearer

to exist in which they inhere. A characteristic that makes them distinct from other types of properties is that one cannot sort the objects by their attribution and know what those objects are (other than, e.g., 'red objects' or 'square objects'). An attribution is also formalized as a unary predicate instead of as binary attribute like in Definition 1. Thus, attributions differ from sortals, and philosophers agree on this matter. Philosophers do have several theories for certain details of attributions, however, with a notable distinction between whether they are universals or tropes or a combination thereof, where a 'trope' inheres in a single individual so that they are grouped by indistinguishability and make up an equivalence class rather than instantiating a universal [64, 66]. Such differences are relevant for linking the metamodel to a foundational ontology, but for the metamodel itself, there is no requirement to commit to either one of them and therefore we leave this option open.

That object types and attributive properties are distinct type of entities is also reflected in extant foundational ontologies. For instance, DOLCE distinguishes at the top-level between Quality (the attribution) and Endurant/Perdurant (the object) [64]. GFO not only includes this ontological distinction (differently from DOLCE, though), but also distinguishes between dimensional versus other attributes and atomic versus non-atomic attributes [65], where a non-atomic attribute is like EER's composite attribute.

3.3.2. Attributions in UML, EER, ORM

Let us first illustrate a sample of different notational variations in the CDM languages, as Figure 6 shows. UML class diagrams use the meaning of attribute as given in Definition 1 and it is typically represented 'inside' a class icon, like the color:string in Figure 6, but the attribute may be drawn also as an association with a class icon at the far end for the data type using the ≪datatype≫ stereotype [10]. ER and EER diagrams let one declare an attribute only partially with the ER/EER entity type and the attribute name but not the data type—assigning a datatype is carried out at the physical design stage of database development. This may give the impression that an ER/EER attribute is a unary entity, but the understanding reflected in the formal foundation of ER and EER [11, 55], is that the attribute is as in Definition 1. ER and EER also have composite and multivalued attributes, which have been included in the metamodel (see Figure 2) because they are in the languages, but, formally, there is no real addition, given that they can be remodeled as basic attributes.

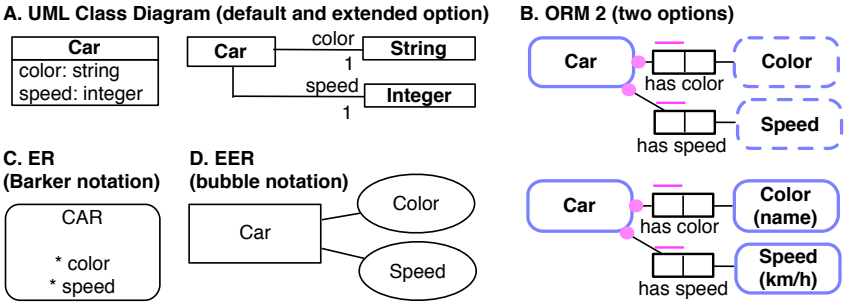


Figure 6: Examples of attributes in different CDM languages, with different notations.

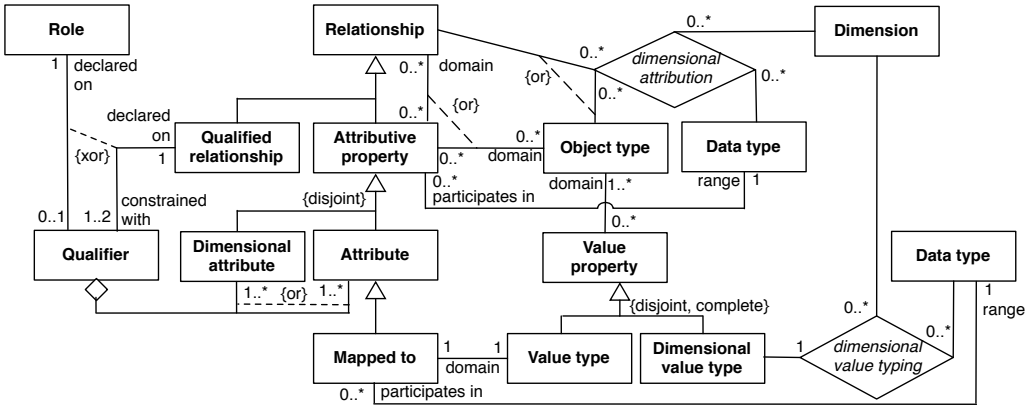


Figure 7: Metamodel fragment for value properties and simple attributes; Dimensional attribute is a reified version of the ternary relation dimensional attribution, and likewise for Dimensional value type and dimensional value typing.

ORM is said to be an "attribute-free" language [12], but in the strict sense of the meaning of *attribute*, they actually do have attributes in the language, albeit hidden. Indeed, ORM does not have attributes 'inside' the entity type like UML. UML's binary attribute, such as the color of the car, can be represented in ORM as a unary *value type* Color that can be related to more than one entity type through a user-defined relationship (ORM fact type). However, a data type has to be specified for each value type in an ORM diagram and these two are related through an attribute. More precisely, an ORM value type's unique feature that distinguishes it from an ORM entity type (in our metamodel, Object type), is that it has a mapped to relationship to the datatype [54], i.e., $\mathsf{mapped\_to} \mapsto \mathsf{ValueType} \times \mathtt{DataType}$, which is generated

by the CASE tool once a value type and its data type have been declared. In our example with the car's color, the assertion mapped_to $\mapsto$ Color $\times$ String is added to the model, which is in addition to a binary relationship called, say, hasColor that is asserted between Car (the object type) and Color (the value type). Thus, ORM does have attributes in the strict sense of the meaning. Practically from a modeling viewpoint, the principal difference between how UML and ORM deal with attributions, is that ORM uses three entities with two binary relationships where one of them is an attribute, whereas UML collapses it all into one binary relationship.

Having settled on the meaning and inclusion of attributive properties in the languages, we have to address one aspect of its inclusion in the metamodel. According to the constraints in Figure 7, an attribute can be of a relationship, of an object type, or both. That it may be of both may be contentious, because one could assume that an attribute drawn in one Object type or Relationship is different from an attribute with the same name and data type that occurs in another Object type or Relationship, due to the fact that the participating domain is different. Practically, UML and ER/EER tools request from a modeler to add again any subsequent occurrence of an attribute. However, theoretically as well as technically, there is no reason not to keep a separate list of attributes for the whole model, and upon reuse, select an earlier-defined one; e.g., declaring hasColor:Integer once, and using it for both Table and Chair. This is common practice for OWL's data properties [67] in ontology editors, which also still leaves open the option to add another color attribute, say, hasColor:String when one has to make that distinction for another object type (e.g., Paint colors of the Pantone system). Alternatively, one can allocate a unique identifier to each attribute and add a constraint that it must have exactly one domain declaration that is either an object type or a relationship, i.e, an {xor} instead of an {or} in Figure 7. Practically, such a stricter constraint still can be implemented in the tools and does not violate the metamodel. Notwithstanding, we keep the less constrained, more flexible way of permitting attribute reuse in the metamodel just in case in the near future such more efficient and consistent attribute management is implemented in the UML and EER CASE tools.

3.3.3. Attributive properties with a Dimension

It is possible to add a *dimension* of the value, such as cm, kg, or day to an attribute or value type; i.e., there is an implicit meaning in the values that has to do with measurements. For instance, when one needs to record the

speed limit of a car, the measured value is not a mere integer or real, but it refers to a value with respect to the measurement system (say, SI Units) and we measure it in km/h. An attribute—e.g., hasSpeed $\mapsto$ Car $\times$ `Integer` does not contain any of that information, yet it ought to be included in the specification of the attribute or value type, even if just to facilitate data integration. ORM's CASE tools allow for this specification in the graphical interface, even though there is no formal characterization of it yet. The ORM diagram serialization in the NORMA CASE tool (v. VS2010_2013-05CTP) actually does not store the dimension separately, but it is merged in the name of the value type.

There are two options to capture dimension of an attributive property's value more precisely and transparently: as a ternary, e.g.,

hasSpeed $\mapsto$ Car $\times$ `Integer` $\times$ km/h,

or with three relations:

hasSpeed $\mapsto$ Car $\times$ Speed

mapped_to $\mapsto$ Speed $\times$ `Integer`

hasDimension $\mapsto$ Integer $\times$ km/h.

Although there are many interesting aspects about measurements and dimensions, and also different models for how to represent a whole system of recording measurement data for a specific scenario (e.g., [68]), within the scope of unification representing the notion that there is a dimension suffices. Therefore, we chose for the more precise ternary relation dimensional value typing in our metamodel, as Figure 7 shows, with a mandatory data type and a dimension. The dimensional attribute is modeled in a similar way (see Figure 7), although the UML standard does not mention anything about dimensions explicitly. A ternary relationship does complicate the formal apparatus, but we prefer this option because unification is the aim, and therewith essentially permitting an extension of UML's metamodel.

3.3.4. Qualifiers

Having established that UML has attributes, and that role and relationship are distinct entities, we now can consider the Qualifier of a Qualified relationship. A qualifier has one or more attributes, partitioning the objects in the relationship. There are two main issues to consider. First, a qualifier itself is also a property, somehow, and thus a property of another property. It is not clear whether that other property is a UML association or association end [10]. In addition, the UML standard has statements such as "If the list is empty, then the Association is not qualified" ([10], p. 126) and "qualifier

rectangle is part of the association path" ([10], p. 130) versus "Designates the optional association end that owns a qualifier attribute" ([10], p. 126). This may be settled at a later date, so for the time being, our metamodel allows one or the other (see Figure 7, top-left). The second issue is the status as to what a qualifier is. The Superstructure Specification mentions that "A property may have other properties (attributes) that *serve* as qualifiers" ([10], p. 125, emphasis added)—as if an attribute plays a certain role in a particular context—yet elsewhere that qualifier is "An optional *list* of ordered qualifier attributes for the end" ([10], p. 126). UML Qualifier appears neither in its metamodel in the Superstructure Specification nor in the ODM to disambiguate its nature (*qualifier* does not even appear in the ODM). The latter statement combined with the graphical icon lets us lean toward a unary property like a UML class, so therefore Qualifier has been added as a subclass of Entity Type; Figure 7 shows additional relations and constraints.

Example 5. Casting the elements of Figure 1 in terms of the metamodel introduced so far, the EER diagram has six attributes that are classified as Attribute in the metamodel: ID, Stem, Affix, Name, Grammatical Number, and POS tag. The UML Class Diagram also has six attributes of type Attribute, being: name (of the affix), stem, root, name (of the isiZulu term), code, and grammNr. There is no Qualifier. ♢

3.4. Constraints

In this section, we consider constraints explicitly available in the language as graphical or textual constraints in the diagram. Observe that, strictly speaking, this does not include the Object Constraint Language (OCL) because OCL is an OMG standard separate from the UML standard. Figure 3 depicts the unified hierarchy of constraints, and individual details are shown in Figures 8-17, which we motivate and discuss in the remainder of this section. Note also that for this part of the metamodel, UML is not expressive enough to represent the full details in the graphical representation, and, where applicable, more precise/additional constraints holding for a particular constraint are added to the figures as structured textual constraints.

3.4.1. Basic Constraints: Mandatory and Uniqueness

The most basic constraints common to all CDM languages are mandatory and functional/uniqueness constraints (see Figures 8 and 9). Besides the basic Mandatory constraint—i.e., 'at least one', 1.., a solid blob or line,

existentially quantified—there are two advanced mandatory constraints in EER and ORM. ORM has also an icon for Disjunctive mandatory (*inclusive-or*) on roles, where all instances of the object type must play at least one of the (≥ 2) roles it participates in. This is also implicitly present in UML as an {or} with 1..*, as the more common {xor} has a flexible description that can be interpreted as permitting {or} in addition to {xor} (see pp. 58-59 of [10]). EER does not have this feature, because EER's arc is always XOR, whereas ORM's disjunctive mandatory that can be inclusive. The Inclusive mandatory is included to cater for EER's AND for relationships. Also this constraint is implicitly present in UML as {and} (see also pp. 58-59 of [10]). It is absent in ORM as such, although it could be approximated with a *join-equality* (see below).

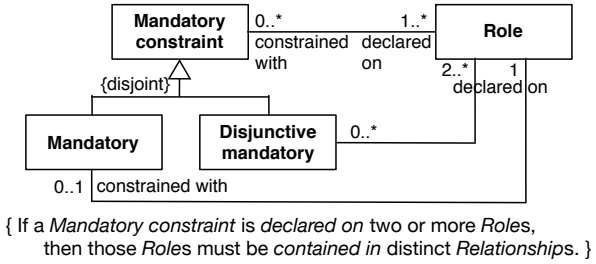


Figure 8: Mandatory constraints.

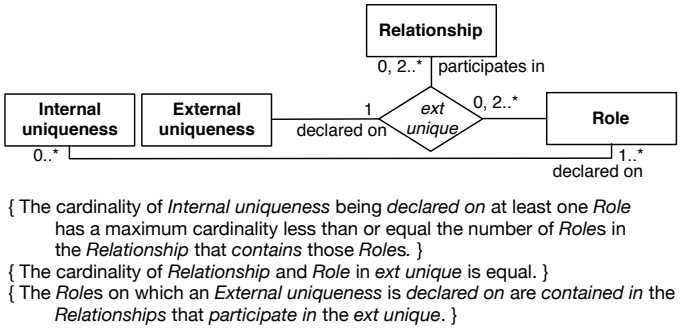


Figure 9: Uniqueness constraints.

An Internal uniqueness constraint on one role amounts to an 'at most one' constraint, or functional relationship, in ORM, EER, and UML, which can

be extended to a uniqueness constraint over more than one role in an n-ary relationship in ORM. An External uniqueness constraint is declared over at least two roles that are part of different relationships, but a role and a relationship need not participate in an external uniqueness constraint. External uniqueness exists explicitly in ORM and in part in UML as qualified association (Qualified relationship), and in part in EER for the weak entity type (see below). UML's qualified association can be seen in the light of Halpin's reconstruction of it in ORM, for which an external uniqueness constraint suffices [12]. Figure 10 shows an example of such a reconstruction. UML allows for more flexibility on qualified relationships, including 0 so that it is not an identifier anymore [10]. Hence, it is not an equivalence between any UML qualified association and ORM external uniqueness, but may depend on the case. Last, a role may participate in more than one uniqueness constraint.

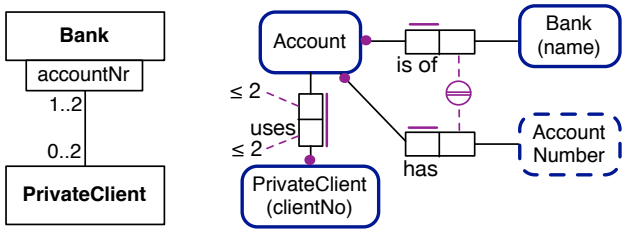


Figure 10: Example of UML's qualified association and a semantic equivalence in ORM2 notation with an external uniqueness that is also chosen as primary reference scheme (the "=" in the circle, not a "–") (Adapted from [12], p. 369).

Example 6. Casting the elements of Figure 1 in terms of the metamodel introduced so far, there are both Mandatory constraints and Uniqueness constraints. In the EER diagram, among others, there is both a Mandatory and a Uniqueness constraint of Morphology in MorphInfo, making an 'exactly 1' (two short lines). In the UML Class Diagram, Morphological SyntaxInfo has a mandatory (1..*) participation in the association with isiZuluTerm, and isiZuluTerm has an object type cardinality of exactly 1 in belongs to, hence, also a Uniqueness constraint. There are no External Uniqueness, Disjunctive mandatory, and Inclusive mandatory constraints in the models. ♢

3.4.2. Identifiers

Identity and identification in CDM languages has been investigated from an ontological perspective recently [56], which goes beyond the mere case of

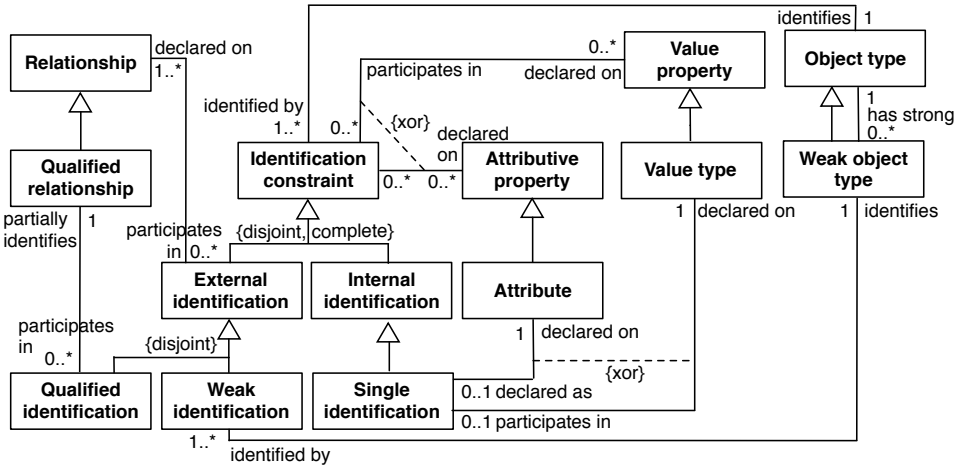
combining a mandatory and a uniqueness constraint for non-simple identifiers.

There are several identifiers, which differ to some extent across EER and ORM even when attribute vs. value type is accounted for (recall Section 3.3). The UML 2.4.1 standard has an {id} option for a single-attribute identifier ([10], p. 130) and a proposed extension in OMG's Ontology Definition Metamodel to flexibly approximate the idea of ER's weak entity type with an «identifier» ([69], pp. 291-292). Figure 11 depicts the unified view of the main entities and their relations. The main distinction is between 'internal' and 'external' identification constraints is that the constraint involves either the relevant object type only or it needs a relation external to the object type for identification. The internal identification serves the single and multi-attribute/compound identifiers within an entity type in ER and EER[2], of which Single identification is a specialization. Single identification corresponds to ER/EER's single attribute identifier, to ORM/ORM2's simple reference mode with a single value type, and UML's {id} construct, and it has a mandatory 1:1 constraint. The more general Internal identification also covers composite identification for a single object type, which is, strictly speaking, absent in ORM. It can be modeled in ORM in a roundabout way, though: an n-ary fact type with at most n-1 lexical or otherwise unconnected nonlexical object types participating, and one nonlexical object type, with a uniqueness constraint on n or n-1 roles and those roles that participate in the uniqueness constraint are also mandatory.

Logically, the {xor} between Attribute and Value type is redundant, because it is already declared for their respective parents Attributive property and Value property, but we have nevertheless included it in the diagram for purposes of clarity. Further, as a result of the associations drawn and their applicable constraints on it, the model allows multi-attribute identifiers that involve simple attributes, as well as composite, dimensional, and multi-valued attributes, as is allowed in ER/EER.

External identification is relevant in all three families of languages: for ER and EER's *weak entity type*, for ORM's *compound reference scheme* that involves more than one fact type, and for UML's *qualified association* that can have its own identifier using one or more attributes of the qualifier.

[2]Sometimes also 'key' is used, although strictly speaking, keys are a feature of the relational model, not the conceptual model.



{ A *Weak identification* is a combination of one or more *Attributive property* of the *Weak object type* it *identifies* together with the *Identification constraint* of the *Object type* it has a *Relationship* with and this *Object type* is disjoint with the *Weak object type*. }
{ The *Single identification* has a *Mandatory* constraint on the participating *Role* and the *Relationship* that *Role* is *contained in* has a 1:1 *Cardinality constraint* declared on it. }
{ *Qualified identification* and *External identification* are *declared on* only *Attributive property*. }
{ A *Qualified relationship participates in* a *Qualified identification* only if the *Cardinality constraint* is 1. }

Figure 11: Partial representation of identification constraints (see text for details).

The principal ontological issues and a formalization are discussed in [56]. A Weak object type has an identifying external uniqueness in which at least one relationship participates and one or more attributes of the object type itself. ORM has the same notion with the compound reference scheme in which object and/or value types from at least two relationships participate, but, thus far, this has not resulted in the object type having been given a specific name. This is principally because the compound reference scheme is a straightforward generalization of ORM's simple reference scheme (Single identification), hence, no particular or clear, crisp, notion of 'weakness' or 'dependency' is considered. Regarding ER's weak entity types [55, 70], this is almost the same as ORM's compound reference scheme, including the mandatory participation, but with the difference primarily caused by the difference between data types and value types. That is, instead of being agnostic about the range of the identifying relationship, as in ORM, it has to be specified that at least one of them must be a relationship.

Example 7. Casting the elements of Figure 1 in terms of the metamodel introduced so far, there are two instances of Single Identifier in the EER

Diagram, being ID and Name. The UML Class Diagram has no identifiers. There is no External identification. ♢

3.4.3. Multiplicity, Cardinality, and Frequency

Cardinality constraints on attributive properties can be declared for both relationships and object types (see Figure 12). Figure 4 captures the cardinality for value types with the cardinality on the role and a participating Entity type, which is one option of ORM's more generic frequency constraint that can apply to any role regardless whether it is a role played by a value type (Value property) or an entity type (Object type). The latter is captured in the metamodel with the Object type cardinality constraint.

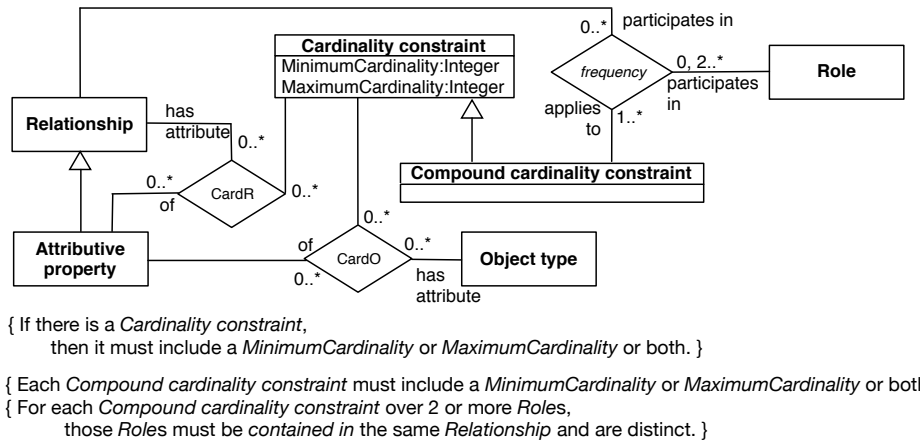


Figure 12: Cardinality constraints on attributes of object types and relationships, and compound cardinality over more than one role in a relationship.

The third type of cardinality is the Compound cardinality constraint, which is a constraint on more than one role in a single relationship, which exists in ORM also as a *frequency constraint* (see Figure 12). For each Compound cardinality constraint, at least two roles of one relationship have to participate in it, but a role and relationship do not have to participate in a compound cardinality constraint.

There are finer details regarding the differences between cardinality in EER and ORM and multiplicity in UML. The former separate optionality from the amount of times an entity participates in a relationship. This can be observed in the diagrams with, among others, black vs. white blobs, orthogonal line vs. white blob, solid vs. dashed line, or purple blob vs no icon to

distinguish between mandatory vs optional, respectively, and with so-called single line vs 'craws' feet' for 1 and many. Multiplicity takes them together into one notion through the specification of the multiplicity constraint with its minimumCardinality set to 0 if it is optional and set to 1 (or 1..* etc.) if it is mandatory. As such, multiplicity has the Mandatory constraint (see Figure 8) and the cardinality constraints from Figure 12 merged.

Example 8. Casting the elements of Figure 1 in terms of the metamodel introduced so far, all relationships have Object type cardinality constraints, extending those mentioned in Example 6. In the EER diagram, among others, there is an 'exactly one' cardinality of Morphology in MorphInfo, and an optional m..n for the relationship antonym. In the UML Class Diagram, isiZuluTerm an has an object type cardinality of exactly 1 in belongs to, and 0..* for Affix's participation. There is no Compound cardinality constraint. ♢

3.4.4. Value Constraints

Role value constraint is redundant in the sense that it can be modeled differently by using subtyping: the relationship is instantiated only when the object has a certain value, which is equivalent to constraining a property, which means it is a subtype. However, the language has a dedicated feature icon for it, and therefore it is included in our metamodel; see Figure 13.

ORM and ORM2 have additional features regarding constraints on value types, which are absent from EER and available in UML for its attributes. The values of a value type can be constrained by means of minimum and maximum values and by means of an enumeration of values; e.g. $\geq$18, $<$65, and {'M', 'F'}, respectively. Figure 13 summarizes this. UML has the notion of a "value specification" (see Figure 7.5 in [10]) with only optional upper and lower values, yet its Figure 7.13 also has enumeration of values. Hence, one can argue to extend the Attributive property with a similar constraint, which is included in Figure 13 on the right-hand side of the figure. ORM and ORM2 also have a Value comparison constraint by using a value comparison operator; see Figure 14.

Example 9. Casting the elements of Figure 1 in terms of the metamodel introduced so far, there are no Value comparison constraints and no Role value constraints in the EER diagram and UML Class Diagram. ♢

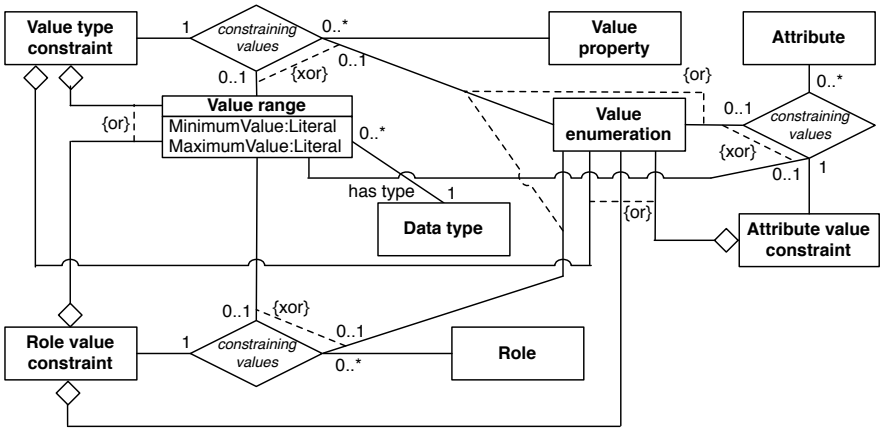


Figure 13: Value type, role, and attribute value constraints.

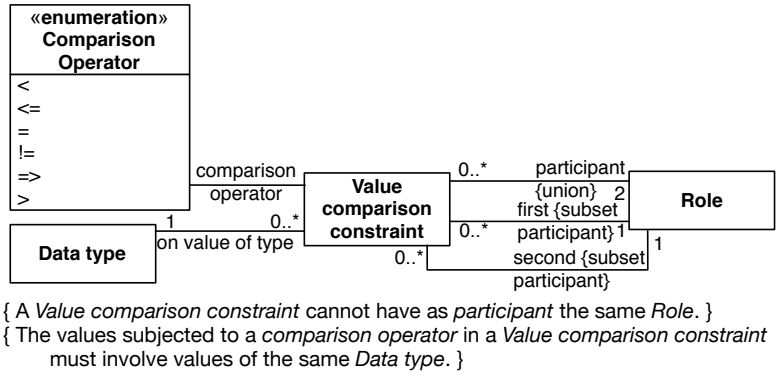
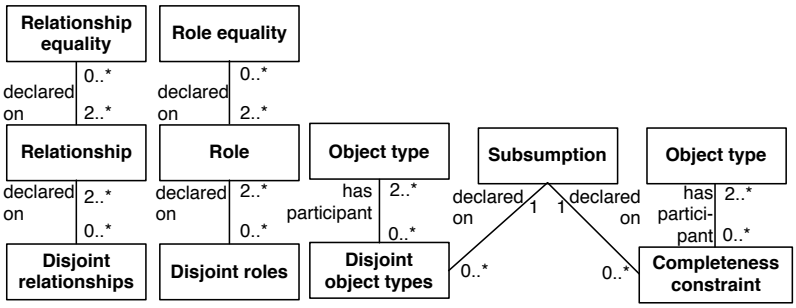


Figure 14: Value comparison constraint; the roles have to be compatible.

3.4.5. Disjointness and Completeness Constraints

The role and relationship constraints of UML, EER, and ORM2 differ mainly on the relational properties and the derived constraints of joins, and are similar regarding equality constraints (see Figure 15, top) and disjointness constraints (Figure 15). Both parts of the metamodel also include completeness for object types, but note that disjointness is represented over two or more subtype relations, not among the object types themselves. In addition, disjointness for relationships and entity types entails that its subtypes may also be involved, i.e., disjoint value properties and disjoint attributive properties, even though there is no graphical support for it at present. The

reason that it is permitted according to the metamodel is because they are by default disjoint; hence, this addition makes implicit constraints explicit. Finally, disjointness is not just asserted between 'compatibles', but they have to be the same kind of entity for it to make sense with respect to the information that is represented. That is, asserting a disjointness constraint between some relationship R and some attribute A or between value type V_1 and dimensional value type V_2 is uninteresting in the former case and already entailed in the latter.



{ For each *Disjoint object types* that is *declared on* two or more *Object type*s, these *Object type*s share a direct common subsumer. }
{ For each *Completeness constraint* that is *declared on* two or more *Object type*s, these *Object type*s share a direct common subsumer. }
{ Compatible *Object type*s are *Object type*s that are of the same category. }
{ Compatible *Role*s are *Role*s that are *linked to* compatible *Entity type*s and those *Entity type*s are either the same or one is a *sub* of the other. }
{ Compatible *Relationship*s are *Relationship*s that have the same number of *Role*s participating in the *sub* and *super* position and those *Role*s are compatible. }
{ Each *Disjoint object types* is *declared on* compatible *Object type*s. }
{ Each *Disjoint relationships* is *declared on* compatible *Relationship*s. }
{ Each *Disjoint roles* is *declared on* compatible *Roles* and those *Role*s are *contained in* distinct *Relationship*s. }
{ Each *Role equality* is *declared on* compatible *Roles* and those *Role*s are *contained in* distinct *Relationship*s. }
{ Each *Relationship equality* is *declared on* compatible *Relationship*s. }

Figure 15: Disjointness and completeness constraints, and role and relationship equality.

Example 10. Casting the elements of Figure 1 in terms of the metamodel introduced so far, there are no Disjointness constraints and no Completeness constraints in the EER diagram and UML Class Diagram (while prefix and suffix are conceptually distinct, their instances may not be). ♢

3.4.6. Relationship Constraints

EER does not have specific relational properties and they are ambiguous for UML Class Diagrams due to the underspecified semantics of the aggre-

gation association. There are multiple properties for ORM and even more in ORM2. In this family, relational properties are asserted over two roles at a time, the arity of the relationship may be ≥ 2, the roles have to be compatible, for some properties they have to be in a particular order, and the roles that are involved in the relational property constraint have to be contained in the same relationship. Overall, ORM2 has 11 different relational properties, which have been structured in a small taxonomy (recall Figure 3). For purposes of clarity, we include the definitions below, where the *purely reflexive* and *strong intransitive* [71] are relatively new (definitions taken from [71], unless mentioned otherwise):

- transitive: $\forall x, y, z(R(x, y) \wedge R(y, z) \rightarrow R(x, z))$;
- intransitive: $\forall x, y, z(R(x, y) \wedge R(y, z) \rightarrow \neg R(x, z))$;
- strongly intransitive: $\forall x, y, z((R(x, y) \wedge P(y, z) \rightarrow \neg R(x, z)$ and "P is recursively defined to give the transitive closure of R, i.e., $\forall x, y(P(x, y) \leftarrow (R(x, y) \vee \exists z(R(x, z) \wedge P(z, y))))$" [71] so as to prohibit 'jumping' over one or more classes in the hierarchy;
- locally reflexive: $\forall x, y(R(x, y) \rightarrow R(x, x))$, which requires a y to be there before x relates to itself through R;
- globally reflexive: $\forall x R(x, x)$;
- purely reflexive: $\forall x, y(R(x, y) \rightarrow x = y)$
- irreflexive: $\forall x \neg R(x, x)$;
- asymmetric: $\forall x, y(R(x, y) \rightarrow \neg R(y, x))$;
- symmetric: $\forall x, y(R(x, y) \rightarrow R(y, x))$;
- antisymmetric: $\forall x, y(R(x, y) \wedge R(y, x) \rightarrow x = y)$, or, as in [12]: $\forall x, y(\neg(x = y) \wedge R(x, y) \rightarrow \neg R(y, x))$;
- acyclic: R is acyclic iff $\forall x \neg(x$ has path to $x)$ [12].

Including all these constraints likely also covers those that may be perceived to be relevant for UML's aggregation association. Further, as already mentioned and shown in [12, 71], one constraint can imply another (an acyclic relationship is also asymmetric), some can be used together (symmetric and transitive), and yet others are excluded (e.g. declaring a relationship to be both symmetric and asymmetric). These are usage combinations, which do not need to be included in the metamodel.

Example 11. Casting the elements of Figure 1 in terms of the metamodel introduced so far, there are no Relationship constraints in the EER diagram and UML Class Diagram, as neither has them in the language. $\diamondsuit$

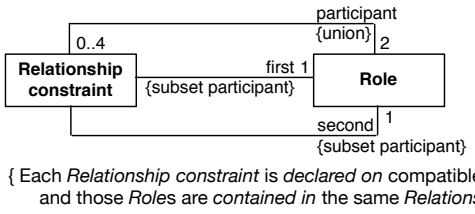


Figure 16: Relationship constraints (see Figure 3 for the hierarchy of relational properties).

3.4.7. Join Constraints

Finally, there are join constraints over roles in different relationships, as depicted in Figure 17, which can be reduced to query containment. Nevertheless, they have dedicated icons in ORM, and therefore they are included in the metamodel.

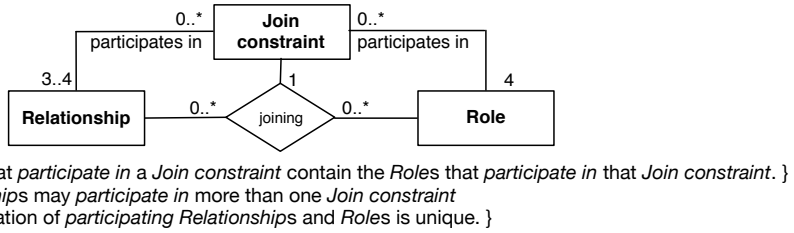


Figure 17: Join constraints.

Example 12. Casting the elements of Figure 1 in terms of the metamodel introduced so far, there are no Join constraints in the EER diagram and UML Class Diagram, as neither has them in the language. ♢

4. Use Cases of the Metamodel

To demonstrate usefulness of the metamodel beyond just having it for purposes of insight into the features of the three language families, we present here two of the many possible usages, being a quantitative case and the other a qualitative one. The first example focuses on classifying entities of 30 conceptual models into metamodel entities, which is a necessary step for automated inter-model assertions. The second case presents a detailed analysis of candidate inter-model assertions between entities from Example 1's

EER model and UML Class Diagram that were used in Section 3 as running example for classifying entities.

4.1. Entities in Terms of the Metamodel

For any implementation supporting inter-model assertions, one needs to be able to classify the entities of a conceptual model into the entities of the metamodel, which presupposes that a given model also adheres to the syntax and semantics of that modeling language. The latter prerequisite is a separate issue and not addressed here, though it can be facilitated by the formalized metamodel. Further, it is useful to gain insight into the prevalence of entities, which can inform which mapping rules should be designed and implemented first so as to cover most of the possible inter-model assertions or transformations. To this end, we collected and analyzed a set of models, aiming to answer the following questions: (1) what is in a set of random models now? (2) does the metamodel miss elements, and if so, which? and (3) how uncommon are the uncommon features?

4.1.1. Materials and methods

For this exploratory evaluation, we collected 30 models: 10 UML models from the GenMyModel repository[3], 10 ORM/ORM2 models, including three from the authors, several from ORM papers, and several diagrams that were sourced online via a Google image search, and 10 ER/EER models, including one from one of the authors and three or more from textbooks and study material from online sources. The models cover different universes of discourse, such as sports, library, video rental, airline reservation, ontology repository, a BPMN metamodel, and so on. Each model was analyzed and the entities present were manually enumerated and tabulated in a spreadsheet (MS Excel). To lessen the burden of counting disaggregated cardinality constraints (such as basic uniqueness and mandatory in ORM), we counted them together, and likewise an aggregation of ORM's ring constraints. The data analysis consisted mainly of the calculation of the average and median for the subtotals per model family, the aggregated for the total of 30 models, the percentage of presence of that feature among the models in the family and aggregated, and against the overlap in model entities (the color-coded classes in Figures 2 and 3).

[3]It is located at `https://repository.genmymodel.com/public/0`.

4.1.2. Results and discussion

All models and the spreadsheet with the raw data and analysis are available as supplementary material online at `http://www.meteck.org/files/DKE14data.zip`. 29 figures were syntactically correct, although possibly incomplete. The model named `fig8.7` in the dataset has a syntactic error: there is a multivalued attribute that also is a compound attribute, which ought not be the permissible.

One salient observation is that all UML diagrams had only names for association ends (Role) but not named associations (Relationship). In contrast, most ER/EER diagrams had only names for the relationships, whereas the variation in the ORM diagrams can be traced back to modeler and tool, and the decision that fact type (Relationship) readings are not taken also as role names unless the tool takes this stance. More than half of the UML models had a PartWhole relationship (aggregation) constituting 23% of the total amount of Relationships in UML models. More than half of the full set of models had one or more Subsumption on Object types, although the aggregated median is only 1.5 per model. Single identification is used most in ER/EER and ORM, then Weak identification and External identification (40% of the models), and only 2 of the 10 ER/EER models had an Internal identification consisting of more than one attribute; no UML diagram had an Identification constraint. There was a relatively high presence of Value type constraints (60% of the ORM models, 1.8 per model), Nested object types (50% and median of 0.5 per model), and relationship attributes in EER (90% with a median of 3 per model). Attributes and Value types are present in abundance, with 2.4 attributes per class in the UML diagrams and 3.4 attributes per entity type in ER/EER; this was only 0.7 for ORM (mainly due to two outlier models).

Regarding uncommon entities, it is noteworthy that there was only one Compound cardinality constraint (frequency over more than one role in ORM), and one Relationship equality, Disjoint role, Disjoint relationship, Join-subset constraint, and Join-equality constraint in any ORM diagram. Relationship constraints were also seldom observed in the ORM diagrams (10 in 4 diagrams). Noteworthy is that there was no Subsumption of relationships in EER and only two in ORM, there were no Disjoint object type or a Completeness constraint in UML, and few Attributive property cardinality assertions in UML (9) and EER (1) compared to Object type cardinality on roles (741), and no Qualified relationships in the set of UML diagrams.

It cannot be excluded that some models 'miss' interesting features be-

cause the modeling tool to draw the diagram does not allow it; for instance the tool with which the model `ORMmultiVerbRunnigExamomeDOGMAall` was made (DOGMA) allows only binary ORM fact types and no value types. Similarly, some textbooks feature associative entity types [72] whereas others [73] do not (at least the model with file name `fig7.2` is from [73] and `modernDBmgmt10thedPVC` and `modernDBmgmt10thedSWvendor` are from [72]). Such in-depth knowledge is not available for all models, however. Also open to speculation and further investigation is the observation that of the 2908 entities classified, 1613 come from the set of ORM diagrams, 559 from the UML diagrams, and 778 from the ER/EER diagrams.

Five models had (partially) derived attributes and one had a dependency association, which are not covered in the metamodel. Two stereotypes have been used, which are beyond the scope of the metamodel, and likewise a realization association. Another aspect is textual constraints, which possibly could be used in some form with inter-model assertions if the other model is expressive enough, but this requires NLP for various natural languages, which is outside our scope for the time being. One UML model had structured several classes in two packages, which do not affect any inter-model assertions either, nor do the explanatory model notes.

That said, one may wonder how feasible it is to achieve full model transformation or possibly linking each element. Of the 2908 entities (including constraints) of the 30 models, 1891 were classified into metamodel entities that appear in all three language families (the black shaded classes in Figures 2 and 3), 515 were classified into entities that appear in two of the three families (dark grey-filled classes), and 502 were classified in entities that appear in only one of the three language families (light grey). Put differently, only about 65% will have a relatively straightforward mapping rule, and the rest will have to be achieved by transformation or approximation rules, where possible, or is unmappable.

4.1.3. Final remarks

Although the sample size is quite small for drawing hard conclusions, it does show that any automated classification of entities into the metamodel based on an arbitrary set of models will be a non-trivial exercise. This is due to the lack of available serialized diagrams (unlike the undisclosed ORM set of [74]), an extant model repository, and that it is difficult to figure out with which tool a model was made and which features it supported at

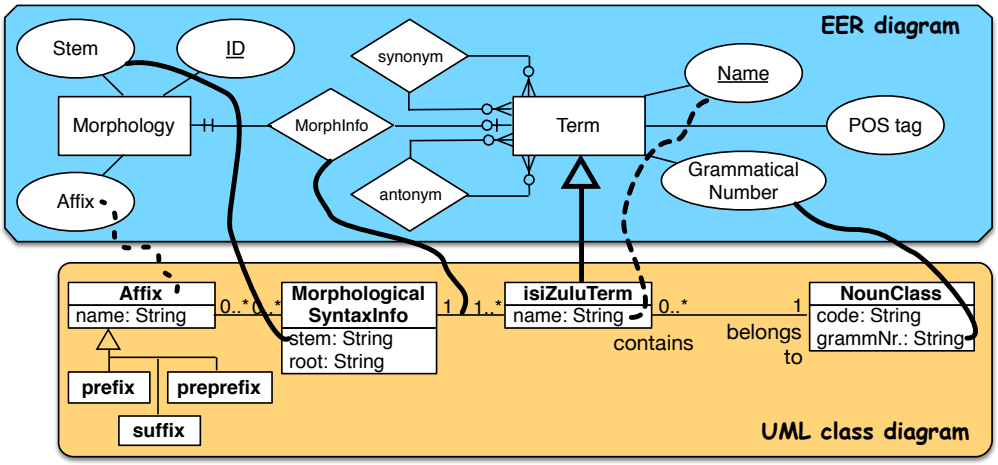


Figure 18: Example of an attempt to integrate the two diagrams of the running example; see text for details.

the time of creation of the model. The results show that our metamodel does have sufficient coverage and can help give insight into characteristics of extant models. Thus, they are promising results and likely to be worth the effort when scaled-up, both regarding interoperability and gaining a better understanding of which features are being used and why.

4.2. Linking Models

Let us revisit the problem described in Example 1 from Section 1, concerning two conceptual models in EER and UML for an isiZulu termbank. The isiZulu termbank needs a similar structure to the generic termbank for interoperability, but also store data specific to the isiZulu language and have the capability to compute terms in the application layer [15]. Figure 18 shows six seemingly obvious links between the entities in the two models, which will be assessed in the remainder of this section.

The asserted subsumption between Term and isiZuluTerm is rather obvious, but the mapping between UML's name:String and EER's Name identifier complicates this: UML's name:String does not guarantee uniqueness of the term, thus one cannot simply add the subsumption. It would have been possible to do so without any chance of having dirty data if the UML diagram were to have had name:String{id}.

The EER-Grammatical Number and UML-grammNr:String agree on both

being a real attribute, assuming that the EER-Grammatical Number is assigned a data type of `String` in the physical design stage, and likewise for the EER-Stem and UML-stem:String.

The MorphInfo relationship and hasMorphInfo association (name not shown in diagram) mean the same as well, but their cardinalities differ. If this were an interface to an automated reasoner-enabled modeling tool, such as Icom [18], then instead of equivalence it would infer that the hasMorphInfo association is a sub-relationship of the MorphInfo relationship, and reduce the 1..* to 1..1. As with name:String, any data that was already in the system would have to be double-checked on whether it is actually 1..1. Another option is to discard the deduction and change the cardinalities in the EER diagram.

Information about affixes in both models cannot be found automatically based on syntactic comparison and the metamodel alone, as the EER-Affix is an attribute and the UML-Affix is a class (object type in the metamodel), which are disjoint. It would be possible to find the correspondence based on string matching—a quite common approach in ontology alignment [75]—and it may useful to devise 'kind conversions', mainly between Attributive property and Object type and between Object type and Relationship.

5. Discussion

Opposite assumptions exist about the various CDM languages, ranging from being practically 'pretty much interchangeable' to 'fundamentally very different'. To in vestige this in detail, we took the approach of aiming to include all static structural elements of the three main CDM languages, rather than a quicker to implement subset, and taking an ontological approach to metamodel development rather than a logic-based one trying to fit it all into one *a priori* chosen logic (see also Section 2.3). What the resultant unifying metamodel clearly demonstrates is that, in the fine details, little is the same across the languages. Yet, at the more general level of principal ontological distinctions, they are quite alike in ontological commitment regarding entities. The integrated overview of the entities and constraints in Figures 2 and 3 provides a basic overview of overlaps, which is refined in the Appendix as to where exactly the overlap is for the classes filled with a dark grey. The amount of 'glue' entities that are in the hierarchy of entities but not in the languages in the two figures (the white-filled classes) are non-negligible compared to the entities in the languages. Six out of the 23 classes for the entities and 10 out of 49 for the constraints are principally used for grouping

similar constraints together and thus will serve well for rules for checking inter-model assertions and model transformations.

The reader may have noted that there are a few redundancies in the metamodel, such as multivalued attributes that can be represented by means of plain attributes, but the aim was to be complete with respect to the graphical features in the CDM languages and not to judge whether it can be represented more elegantly (this is addressed in the formalization). The fact is that the features are available in the graphical languages, hence, a unifying model and, moreover, any tools for linking and transforming CDMs, will have to be capable of handling such entities and constraints.

5.1. Metamodel Design Considerations

We made several design decisions during the development of the metamodel, which have to do with the approach and representation language and with Ontology and ontologies. We reflect on these now.

5.1.1. Its Representation Language

There are limitations on what can be represented in UML Class Diagrams, and we had to include some textual constraints that are not possible to express in plain UML Class Diagram notation. In addition, UML does not have a standardized formalization. However, we preferred a more widely understood graphical notation for reasons of broader communication and human readability and understandability compared to a richer language, such as ORM. In addition, we have formalized the metamodel in a suitable logic anyway [50]. This FOL version of the metamodel also provides the basis for a systematic analysis of transformation rules from/to the metamodel and within-metamodel conversions to generate and check the validity of inter-model assertions. For instance, relating a relationship induces the checking of its roles, of the object types that participate in it, and their identifiers, due to the chain of mandatory participations in the metamodel. The metamodel includes even more structural constraints than some formalizations of the CDM languages, for example OMG's UML infrastructure. This would allow one to realize more comprehensive checks on both intra-model and inter-model assertions.

5.1.2. Ontology-driven Metamodeling

Concerning the modeling of the metamodel itself, the quality has improved both by using the automated reasoner (data not included) and by

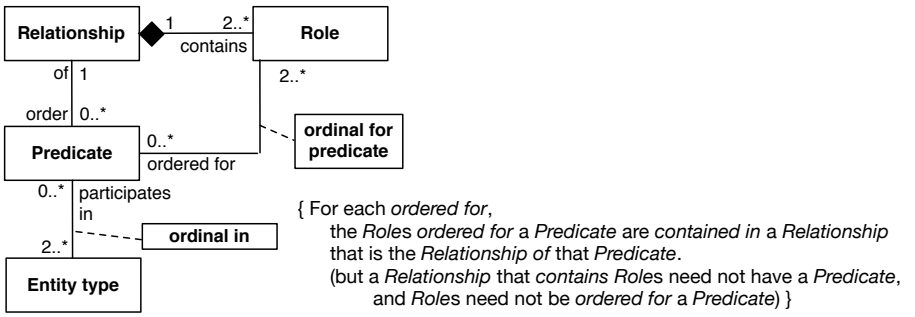


Figure 19: Relationships between these entities and Predicate, for explanatory purpose; see text for details.

being guided by insights obtained from Ontology and ontologies, most notably regarding the nature of Object type, Relationship and Role with the positionalist commitment, and Attributive property regarding the more precise characterization and constraints. One aspect deferred from Section 3.2, is the interaction of predicates with relationships and roles. Some of the ORM references [12, 54] add predicates as another way to handle its fact types, and many logic languages use the predicate-based approach, not roles. However, predicates do not appear in ORM's graphical icons, and therefore it is not included in the metamodel. Nevertheless, it is useful to address this briefly, which also contributes to understanding the positionalist commitment of the CDM languages. A relationship is composed of an unordered set of roles that participating objects play, compared to predicates that have the participating objects ordered in a fixed sequence. Predicate in the context of CDM Relationship, would have to be at least binary, whereas in logic, predicates can be unary. If unary, then they are, ontologically, a different kind of thing and order in a unary predicate does not make sense. In addition, it is not clear how to relate Relationship and Predicate. The draft ORM/FBM ISO standard depicts Predicate with a composite aggregation to Fact type (relationship) [54], but this is incorrect, because relationship and predicate each exemplify a different ontological commitment. Predicate exemplifies the so-called "standard view" and CDM's relationship the "positionalism" that requires the existence of roles [58]. Moreover, each permutation of an ordering among participating objects without the use of roles (i.e., in a predicate) is therewith not 'part' of a single unordered composition of roles, and the

predicates do not 'constitute' the relationship (roles do). It cannot be subsumption either, as some thing without roles cannot be always a thing with roles. Therefore, it may be modeled as a plain association, where a predicate is one of the possible orderings of the entities that play the roles in that relationship; see Figure 19 for indicative purpose. Note that Role is not *ordered in* a, but *for* a, Predicate, because the roles are ordered and then 'removed' to obtain a predicate. Alternatively, and ontologically more accurately, one can decide that roles have nothing to do with predicates so that it is only Entity type that participates in zero or more Predicates. While the idea is intuitively clear, ontologically, the interaction deserves refinement, but this is outside the scope of the metamodel.

5.2. Discussion of the Use Cases

Automated classification of entities has be shown to work when the model is made with a single toolset [74], but tool heterogeneity in their serialization as well as making such serializations available to the public is a hurdle to be overcome. The results of the manual effort described in Section 4.1 do demonstrate, however, that such efforts can be worthwhile in increasing understanding of publicly available conceptual models.

The qualitative use case also showed usefulness of the metamodel in analyzing and validating inter-model assertions across CDM languages. Currently, there is no tool support for it, but foundations have been laid both at the implementation level and now also at the theoretical level. On the one hand, ICom already allows for inter-model assertions and reasoning over them (albeit models in one language only, not two, as in Section 4.2) [18]. On the other hand, the formalization of this metamodel may be used for computing the mappings and transformation, as described in [76]. That is, the prospects are realistic now.

6. Conclusions

The main contribution of this paper is the new *ontology-driven unifying metamodel of all static, structural entities, their relationships, and the constraints* of the three most used families of CDM languages, being ORM/FBM, EER, and UML Class Diagrams (v2.4.1). From the feature analysis it follows that there is only a relatively small intersection of features. For entities, they are: relationship, role, object type, and the subsumption relationship. The languages agree also on the ontological commitments regarding object types

and, notably, the positionalist commitment as to the nature of relationships with roles. Further, while the languages agree on the inclusion of attributive properties, the actual language features to represent them differ, and the largely implicit notion of the dimension of an attribute or value type has been made explicit. For constraints, the common features include: disjoint roles, disjoint entity types, completeness constraint, mandatory, object-type cardinality, single identification, and the subset constraint. The metamodel fragment for constraints is fairly straightforward, except for identification constraints and qualified associations with qualifiers, the latter principally because it is ambiguous in the UML standard.

The use-cases showed where and how the metamodel can be deployed, thereby gaining insight in the use of model entities as well as ensuring precision in mapping, based on 30 conceptual models and the assertion of links between an EER and a UML diagram as examples.

We have completed the formalization of the metamodel [50] and have commenced with the characterization of transformation and mapping rules [76]. The unifying metamodel and its formalization will help in the comprehension of differences between heterogeneous conceptual models and in the development of tools that will aid in information integration and software interoperability that uses conceptual data models.

Acknowledgments

This work is based upon research supported by the National Research Foundation of South Africa and the Argentinian Ministry of Science and Technology (MINCyT). Any opinion, findings and conclusions or recommendations expressed in this material are those of the author and therefore the NRF does not accept any liability in regard thereto.

References

[1] A. Rosenthal, P. Mork, M. H. Li, J. Stanford, D. Koester, P. Reynolds, Cloud computing: a new business paradigm for biomedical information sharing, Journal of Biomedical Informatics 43 (2010) 342–353.

[2] K. Mendes Calo, K. M. Cenci, P. R. Fillottrani, E. C. Estevez, Information sharing – benefits, Journal of Computer Science & Technology 12 (2012) 49–55.

[3] United Nations Department of Economic and Social Affairs, United Nations E-Government Survey 2010 – Leveraging e-government at a time of financial and economic crisis, Technical Report ST/ESA/PAD/SER.E/131, United Nations, 2010.

[4] A. Banal-Estanol, Information-sharing implications of horizontal mergers, International Journal of Industrial Organization 25 (2007) 31–49.

[5] D. Chen, G. Doumeingts, F. Vernadat, Architectures for enterprise integration and interoperability: Past, present and future, Computers in Industry 59 (2008) 647–659.

[6] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J. H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang, The systems biology markup language (SBML): a medium for representation and exchange of biochemical network models, Bioinformatics 19 (2003) 524–531.

[7] Selventa, OpenBEL Consortium, Biological Expression Language v1.0, 2011. `http://www.openbel.org/`.

[8] C. Portele, OGC Geography Markup Language (GML) – Extended schemas and encoding rules, Technical Report OGC 10-129r1, Open Geospatial Consortium, 2012.

[9] P. J. Hayes, P. F. Patel-Schneider, RDF 1.1 Semantics, W3C recommendation, World Wide Web Consortium, 2014. `http://www.w3.org/TR/rdf11-mt/`.

[10] Object Management Group, Superstructure Specification, Standard 2.4.1, Object Management Group, 2012. Http://www.omg.org/spec/UML/2.4.1/.

[11] B. Thalheim, Extended entity relationship model, in: L. Liu, M. T. Özsu (Eds.), Encyclopedia of Database Systems, volume 1, Springer, 2009, pp. 1083–1091.

[12] T. Halpin, Information Modeling and Relational Databases, San Francisco: Morgan Kaufmann Publishers, 2001.

[13] C. Parent, S. Spaccapietra, E. Zimányi, Conceptual modeling for traditional and spatio-temporal applications—the MADS approach, Berlin Heidelberg: Springer Verlag, 2006.

[14] J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos: Representing knowledge about information systems, ACM Transactions on Information Systems 8 (1990) 325–362.

[15] S. Dlamini, C. M. Keet, L. Khumalo, K. Mngadi, E. A. N. Ongoma, ULPDO isiZulu Termbank Development, in: 6th Workshop on African Language Technology (AfLaT'14), p. 1. 27-28 Nov, 2014, Cape Town, South Africa. [software demo abstract].

[16] S. Bowers, L. M. L. Delcambre, Using the uni-level description (ULD) to support data-model interoperability, Data & Knowledge Engineering 59 (2006) 511–533.

[17] P. Atzeni, P. Cappellari, R. Torlone, P. A. Bernstein, G. Gianforme, Model-independent schema translation, VLDB Journal 17 (2008) 1347–1370.

[18] P. R. Fillottrani, E. Franconi, S. Tessaris, The ICOM 3.0 intelligent conceptual modelling tool and methodology, Semantic Web Journal 3 (2012) 293–306.

[19] P. Atzeni, G. Gianforme, P. Cappellari, Data model descriptions and translation signatures in a multi-model framework, AMAI Intelligence 63 (2012) 1–29.

[20] M. Boyd, P. McBrien, Comparing and transforming between data models via an intermediate hypergraph data model, Journal on Data Semantics IV (2005) 69–109.

[21] J. Venable, J. Grundy, Integrating and supporting Entity Relationship and Object Role Models, in: M. P. Papazoglou (Ed.), Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Modelling (ER'95), volume 1021 of *LNCS*, Springer, 1995, pp. 318–328. Gold Coast, Australia, December 12-15, 1995.

[22] D. Calvanese, M. Lenzerini, D. Nardi, Unifying class-based representation formalisms, Journal of Artificial Intelligence Research 11 (1999) 199–240.

[23] T. A. Halpin, Advanced Topics in Database Research, volume 3, Idea Publishing Group, Hershey PA, USA, pp. 23–44.

[24] C. M. Keet, Positionalism of relations and its consequences for fact-oriented modelling, in: R. Meersman, P. Herrero, D. T. (Eds.), OTM Workshops, International Workshop on Fact-Oriented Modeling (ORM'09), volume 5872 of *LNCS*, Springer, 2009, pp. 735–744. Vilamoura, Portugal, November 4-6, 2009.

[25] G. Guizzardi, Ontological Foundations for Structural Conceptual Models, Phd thesis, University of Twente, The Netherlands. Telematica Instituut Fundamental Research Series No. 15, 2005.

[26] Z. Khan, C. M. Keet, The foundational ontology library ROMULUS, in: A. Cuzzocrea, S. Maabout (Eds.), Proceedings of the 3rd International Conference on Model & Data Engineering (MEDI'13), volume 8216 of *LNCS*, Springer, 2013, pp. 200–211. September 25-27, 2013, Amantea, Calabria, Italy.

[27] J. Grundy, J. Venable, Towards an integrated environment for method engineering, in: Proceedings of the IFIP TC8, WG8.1/8.2 Method Engineering 1996 (ME'96), volume 1, pp. 45–62.

[28] N. Zhu, J. Grundy, J. Hosking, Pounamu: a metatool for multi-view visual language environment construction, in: IEEE Conf. on Visual Languages and Human-Centric Computing 2004.

[29] B. Thalheim, Model suites for multi-layered database modelling, in: Proceeding of the XXI Conference on Information Modelling and Knowledge Bases 2010, Frontiers in Artificial Intelligence and Applications, IOS Press, 2010, pp. 116–134.

[30] A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, M. Zakharyaschev, Reasoning over extended ER models, in: C. Parent, K.-D. Schewe, V. C. Storey, B. Thalheim (Eds.), Proceedings of the 26th International Conference on Conceptual Modeling (ER'07), volume 4801 of *LNCS*, Springer, 2007, pp. 277–292. Auckland, New Zealand, November 5-9, 2007.

[31] D. Berardi, D. Calvanese, G. De Giacomo, Reasoning on UML class diagrams, Artificial Intelligence 168 (2005) 70–118.

[32] T. Halpin, A logical analysis of information systems: static aspects of the data-oriented perspective, Ph.D. thesis, University of Queensland, Australia, 1989.

[33] A. H. M. t. Hofstede, H. A. Proper, How to formalize it? formalization principles for information systems development methods, Information and Software Technology 40 (1998) 519–540.

[34] K. Kaneiwa, K. Satoh, Consistency checking algorithms for restricted UML class diagrams., in: Proceedings of the 4th International Symposium on Foundations of Information and Knowledge Systems (FoIKS'06), Springer Verlag, 2006.

[35] A. Queralt, E. Teniente, Decidable reasoning in UML schemas with constraints, in: Z. Bellahsene, M. Léonard (Eds.), Proceedings of the 20th International Conference on Advanced Information Systems Engineering (CAiSE'08), volume 5074 of *LNCS*, Springer, 2008, pp. 281–295. Montpellier, France, June 16-20, 2008.

[36] C. M. Keet, Ontology-driven formal conceptual data modeling for biological data analysis, in: M. Elloumi, A. Y. Zomaya (Eds.), Biological Knowledge Discovery Handbook: Preprocessing, Mining and Postprocessing of Biological Data, Wiley, 2013, pp. 129–154.

[37] C. M. Keet, Prospects for and issues with mapping the Object-Role Modeling language into $\mathcal{DLR}_{ifd}$, in: 20th International Workshop on Description Logics (DL'07), volume 250 of *CEUR-WS*, pp. 331–338. 8-10 June 2007, Bressanone, Italy.

[38] C. M. Keet, Mapping the Object-Role Modeling language ORM2 into Description Logic language $\mathcal{DLR}_{ifd}$, Technical Report

arXiv:cs.LO/0702089v2, KRDB Research Centre, Free University of Bozen-Bolzano, Italy, 2009. ArXiv:cs.LO/0702089v2.

[39] E. Franconi, A. Mosca, D. Solomakhin, The formalisation of ORM2 and its encoding in OWL2, KRDB Research Centre Technical Report KRDB12-2, Faculty of Computer Science, Free University of Bozen-Bolzano, Italy, 2012.

[40] A. Jahangard Rafsanjani, S.-H. Mirian-Hosseinabadi, A Z Approach to Formalization and Validation of ORM Models, in: E. Ariwa, E. El-Qawasmeh (Eds.), Digital Enterprise and Information Systems, volume 194 of *CCIS*, Springer, 2011, pp. 513–526.

[41] T. Mossakowski, C. Lange, O. Kutz, Three semantics for the core of the Distributed Ontology Language, in: M. Grüninger (Ed.), Proceedings of the 7th International Conference on Formal Ontology in Information Systems (FOIS'12), IOS Press, 2012. 24-27 July, Graz, Austria.

[42] N. Guarino, G. Guizzardi, In the defense of ontological foundations for conceptual modeling, Scandinavian Journal of Information Systems 18 (2006) (debate forum, 9p).

[43] S. R. Fiorini, M. Abel, C. M. Scherer, An approach for grounding ontologies in raw data using foundational ontology, Information Systems 38 (2013) 784 – 799.

[44] G. Guizzardi, G. Wagner, R. de Almeida Falbo, R. S. S. Guizzardi, J. P. A. Almeida, Towards ontological foundations for the conceptual modeling of events, in: W. Ng, V. C. Storey, J. Trujillo (Eds.), Proceedings of the 32nd International Conference on Conceptual Modeling (ER'13), volume 8217 of *LNCS*, Springer, 2013, pp. 327–341. 11-13 November, 2013, Hong Kong.

[45] C. Partridge, C. Gonzalez-Perez, B. Henderson-Sellers, Are conceptual models concept models?, in: W. Ng, V. C. Storey, J. Trujillo (Eds.), 32nd International Conference on Conceptual Modeling (ER'13), volume 8217 of *LNCS*, Springer, 2013, pp. 96–105. 11-13 November, 2013, Hong Kong.

[46] G. Guizzardi, G. Wagner, Using the unified foundational ontology (UFO) as a foundation for general conceptual modeling languages,

in: Theory and Applications of Ontology: Computer Applications, Springer, 2010, pp. 175–196.

[47] C. M. Keet, A. Artale, Representing and reasoning over a taxonomy of part-whole relations, Applied Ontology 3 (2008) 91–110.

[48] G. Guizzardi, G. Wagner, What's in a relationship: An ontological analysis, in: Q. Li, S. Spaccapietra, E. S. K. Yu, A. Olivé (Eds.), Proceedings of the 27th International Conference on Conceptual Modeling (ER'08), volume 5231 of *LNCS*, Springer, 2008, pp. 83–97. Barcelona, Spain, October 20-24, 2008.

[49] A. M. Martínez Ferrandis, O. Pastor López, G. Guizzardi, Applying the principles of an ontology-based approach to a conceptual schema of human genome, in: 32nd International Conference on Conceptual Modeling (ER'13), volume 8217 of *LNCS*, Springer, 2013, pp. 471–478. 11-13 November, Hong Kong.

[50] P. R. Fillottrani, C. M. Keet, KF metamodel formalization, Technical Report 1078634, 2014. Arxiv.org, 21p.

[51] D. Earl, The classical theory of concepts, in: Internet Encyclopedia of Philosophy, 2005. Http://www.iep.utm.edu/c/concepts.htm.

[52] C. Swoyer, Properties, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, Stanford, winter 2000 edition, 2000. Http://plato.stanford.edu/archives/win2000/ entries/properties/.

[53] M. C. MacLeod, E. M. Rubenstein, Universals, in: The Internet Encyclopedia of Philosophy, 2005. Http://www.iep.utm.edu/u/universa.htm.

[54] Committee Members, Information technology – metamodel framework for interoperability (MFI) – Part xx: Metamodel for Fact Based Information Model Registration, 2012. ISO/IEC WD 19763-xx.02.

[55] P. P. Chen, The entity-relationship model—toward a unified view of data, ACM Transactions on Database Systems 1 (1976) 9–36.

[56] C. M. Keet, Enhancing identification mechanisms in UML class diagrams with meaningful keys, in: Proceeding of the SAICSIT Annual Research Conference 2011 (SAICSIT'11), ACM Conference Proceedings, 2011, pp. 283–286. Cape Town, South Africa, October 3-5, 2011.

[57] T. Halpin, Objectification of relationships, in: Proceedings of the 10th International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'05), CEUR-WS. 13-14 June, 2005, Porto, Portugal.

[58] J. Leo, Modeling relations, Journal of Philosophical Logic 37 (2008) 353–385.

[59] A. Rector, Modularisation of domain ontologies implemented in description logics and related formalisms including owl, in: J. Genari (Ed.), Proceedings of Knowledge Capture 2003 (K-CAP03), pp. 121–129.

[60] B. Smith, Beyond concepts, or: Ontology as reality representation, in: A. Varzi, L. Vieu (Eds.), Formal Ontology and Information Systems. Proceedings of the Third International Conference (FOIS'04), Amsterdam: IOS Press, 2004, pp. 73–84.

[61] C. M. Keet, Detecting and revising flaws in OWL object property expressions, in: A. ten Teije, et al. (Eds.), 18th International Conference on Knowledge Engineering and Knowledge Management (EKAW'12), volume 7603 of *LNAI*, Springer, 2012, pp. 252–266. Oct 8-12, Galway, Ireland.

[62] C. M. Keet, Preventing, detecting, and revising flaws in object property expressions, Journal on Data Semantics 3 (2014) 189–206.

[63] J. Odell, Advanced Object-Oriented Analysis & Design using UML, Cambridge: Cambridge University Press, 1998.

[64] S. Borgo, C. Masolo, Foundational choices in DOLCE, in: Handbook on Ontologies, Springer, 2 edition, 2009, pp. 361–381.

[65] H. Herre, General Formal Ontology (GFO): A foundational ontology for conceptual modelling, in: Theory and Applications of Ontology: Computer Applications, Springer, 2010, pp. 297–345.

[66] G. Guizzardi, C. Masolo, S. Borgo, In defense of a trope-based ontology for conceptual modeling: An example with the foundations of attributes, weak entities an datatypes., in: D. W. Embley, A. Olivé, S. Ram (Eds.),

Proceedings of the 25th International Conference on Conceptual Modeling (ER'06), volume 4215 of *LNCS*, Springer, 2006, pp. 112–125. Tucson, AZ, USA, November 6-9, 2006.

[67] B. Motik, P. F. Patel-Schneider, B. Parsia, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, W3C Recommendation, W3C, 2009. Http://www.w3.org/TR/owl2-syntax/.

[68] S. Bowers, J. S. Madin, M. P. Schildhauer, A conceptual modeling framework for expressing observational data semantics, in: Q. Li, et al. (Eds.), Proceedings of the 27th International Conference on Conceptual Modeling (ER'06), volume 5231 of *LNCS*, Springer, 2008, pp. 41–54.

[69] Object Management Group, Ontology Definition Metamodel v1.0, Technical Report formal/2009-05-01, Object Management Group, 2009. `http://www.omg.org/spec/ODM/1.0`.

[70] I.-Y. Song, P. P. Chen, Entity relationship model, in: L. Liu, M. T. Özsu (Eds.), Encyclopedia of Database Systems, volume 1, Springer, 2009, pp. 1003–1009.

[71] T. A. Halpin, M. Curland, Enriched support for ring constraints, in: R. Meersman, T. S. Dillon, P. Herrero (Eds.), OTM Workshops 2011, volume 7046 of *LNCS*, Springer, 2011, pp. 309–318. Hersonissos, Crete, Greece, October 17-21, 2011.

[72] J. A. Hoffer, V. Ramesh, H. Topi, Modern Database Management, Pearson Education, 10th edition, 2011.

[73] R. Elmasri, S. B. Navathe, Database systems: models, languages, design, and applicationon programming, Pearson Education, 6th edition, 2011.

[74] Y. Smaragdakis, C. Csallner, R. Subramanian, Scalable automatic test data generation from modeling diagrams, in: Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE'07), pp. 4–13. Nov. 5-9, Atlanta, Georgia, USA.

[75] J. Euzenat, P. Shvaiko, Ontology Matching, Springer, 2007.

[76] P. Fillottrani, C. Keet, Conceptual model interoperability: a metamodel-driven approach, in: A. Bikakis, et al. (Eds.), Proceedings of

the 8th International Web Rule Symposium (RuleML'14), volume 8620 of *LNCS*, Springer, 2014, pp. 52–66. August 18-20, 2014, Prague, Czech Republic.

Appendix

Terminology comparison and conventions of the entities in UML Class Diagrams, EER, and ORM/FBM (for indicative purposes).

Table 1: Terminology comparison between CDM languages.

metamodel term	**UML Class Diagram v.2.4.1**	**EER**	**ORM/FBM**
		Core entities	
Relationship	association, can be 2-ary according to the MOF 2.4.1, but also >2-ary according to the Superstructure Spec 2.4.1	relationship, ≥2-ary	atomic/compound fact type, ≥1-ary
Role	association end / member end	component of a relationship	role
Entity type	classifier	ABSENT	object type
Object type	class	entity type	nonlexical object type/entity type
		Attributive properties	
Attribute	attribute	attribute, but without including a data type in the diagram	ABSENT(represented differently)
Dimensional attribute	ABSENT(no recording of dimension)	ABSENT	ABSENT(represented differently)
Composite attribute	more general: a property can be a composite of another property (and an attribute is a property)	composite attribute	implicitly present by adding new roles

Continued on next page

Table 1 – continued from previous page			
metamodel term	**UML Class Diagram v.2.4.1**	**EER**	**ORM/FBM**
Multivalued attribute	ABSENT(represented differently)	multivalued attribute	ABSENT(represented differently)
Value type	ABSENT	ABSENT	lexical object type/value type, without dimension
Dimensional value type	ABSENT	ABSENT	lexical object type/value type, with dimension
Data type	LiteralSpecification (there are only six, see Fig 7.6 in the UML Superstructure v2.4.1)	ABSENT	data type
Special object types			
Nested object type	association class	ABSENT	objectified fact type
Weak object type	ABSENT	weak entity type	ABSENT
Associative object type	ABSENT	associative entity type	ABSENT
Special relationships			
Qualified relationship	qualified association	ABSENT	ABSENT
Composite aggregate	composite aggregation	ABSENT	ABSENT
Shared aggregate	shared aggregation	ABSENT	ABSENT
Subsumptions			
Object subtype Sub-relationship	subclass subsetting or subtyping of association	subtype subtyping the relationship (not present in all EER variants)	subtype subset constraint on fact type
Mandatory constraints			
Mandatory Disjunctive mandatory	mandatory role implicitly present (e.g., as an {or}, see pp. 58-59 of [10])	mandatory ABSENT	mandatory disjunctive mandatory/inclusive-or on roles
Continued on next page			

Table 1 – continued from previous page			
metamodel term	**UML Class Diagram v.2.4.1**	**EER**	**ORM/FBM**
Inclusive mandatory	implicitly present (e.g., as an {and}, see pp. 58-59 of [10])	AND for relationships	ABSENT as such (but can be approximated with a join-equality)
Identification and uniqueness constraints			
Internal identification	alt key mechanism, for one class	compound identifier, for attributes within entity type	ABSENT
Single identification	alt key mechanism (in [69], not in v2.4.1) made mandatory 1:1	unary identifier (mandatory 1:1)	(simple) reference mode (mandatory 1:1)
External identification	an (extended) alt key mechanism [69], see also a refinement in [56]	needed for weak entity type	compound reference scheme
Qualified identification	qualified association	ABSENT	(limited version of) external uniqueness
Weak identification	ABSENT	weak identifier	(limited version of) external uniqueness
Uniqueness constraint	ABSENT	ABSENT	uniqueness constraint
Internal uniqueness	ABSENT	exists in part, where the participants are all attributes (see also EER's compound identifier)	internal uniqueness constraint where the participating roles can be roles of either Object types or Value types
External uniqueness	exists in part, with the qualified association	exists in part, where the participants are all attributes	external uniqueness constraint where the participating roles can be roles of either Object types or Value types

Continued on next page

Table 1 – continued from previous page			
metamodel term	**UML Class Diagram v.2.4.1**	**EER**	**ORM/FBM**
Cardinality and frequency constraints			
Attributive property cardinality constraint	multiplicity on attribute	cardinality on attribute	ABSENT(add cardinality to the fact type with a lexical object type)
Object type cardinality constraint	multiplicity constraint, with minimum and/or maximum	cardinality constraint, with minimum cardinality and/or maximum cardinality	frequency constraint (object cardinality) constraint on one role, with minimum cardinality and/or maximum cardinality
Compound cardinality constraint (in one relationship, involving $>$ 1 role)	ABSENT	ABSENT	frequency constraints in one fact type over more than one role
Value constraints			
Value comparison constraint	ABSENT	ABSENT	value comparison constraint
Value type constraint	value specification	ABSENT	value constraint
Role value constraint	ABSENT	ABSENT	role value constraint
Attribute value constraint	attribute value constraint	ABSENT	ABSENT
Subsumption constraints			
Disjoint entity types	disjoint	disjoint	exclusive subtypes
Completeness constraint	complete	total	total (exhaustive) covering subtypes
Set-comparison constraints			
Role subset (inherited from the subsumption on Entity)	subsetting of association ends	ABSENT	subset constraint on role
Continued on next page			

Table 1 – concluded from previous page			
metamodel term term	**UML Class Diagram v.2.4.1 Diagram v.2.4.1**	**EER**	**ORM/FBM**
Disjoint roles	{xor}	XOR	exclusion (on roles)
Role equality	ABSENT	ABSENT	equality (on roles)
Disjoint relationships	ABSENT	ABSENT	exclusion (on fact types)
Relationship equality	ABSENT	ABSENT	equality (on fact types)
Join-subset constraint	ABSENT	ABSENT	join-subset constraint among four roles in three or four relationships
Join-disjointness constraint	ABSENT	ABSENT	join-exclusion among four roles in three or four relationships
Join-equality constraint	ABSENT	ABSENT	join-equality among four roles in three or four relationships
Relationship constraint	ABSENT	ABSENT	for the full list of supported relationship constraints, see [71] and Section 3.4.5 with Figure 3, above.