Structural entities of an ontology-driven unifying metamodel for UML, EER, and ORM2

C. Maria Keet[1] and Pablo Rubén Fillottrani[2,3]

[1]School of Mathematics, Statistics, and Computer Science, University of KwaZulu-Natal and UKZN/CSIR-Meraka Centre for Artificial Intelligence Research, South Africa, keet@ukzn.ac.za
[2] Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, Bahía Blanca, Argentina, prf@cs.uns.edu.ar
[3]Comisión de Investigaciones Científicas, Provincia de Buenos Aires, Argentina

Abstract. Software interoperability may be achieved by using their respective conceptual data models. However, each model may be represented in a different conceptual data modelling language for the tool's purpose or due to legacy issues. Several translations between small subsets of language features are known, but no unified model exists that includes all their language features. Aiming toward filling this gap, we designed a common and unified, ontology-driven, metamodel covering and unifying EER, UML Class Diagrams v2.4.1, and ORM2. This paper presents the static, structural, components of the metamodel, highlighting the common entities and summarizing some modelling motivations.

1 Introduction

The need for, and reality of, complex software system design and integration of information from heterogeneous sources is motivated by upscaling of scientific collaboration in the life sciences [36], e-government initiatives [31], company mergers [4], and the Semantic Web. Therefore, establishing connections between multiple conceptual models has become an important task, as the system's conceptual data models may be available in, mainly, UML, EER and ORM. However, this capability is not common in traditional information systems development and management other than at the physical schema layer [6] and for conceptual models represented in the same language [2,12]. Subtle representational and expressive differences (e.g., [17]) in the languages are primarily due to their different origins and purposes, and makes this task very difficult, and even within one language family there are differences in meaning of an element [14,24].

The state of the art in this area has only incidentally gone beyond a single Conceptual Data Modelling (CDM) language and only for UML and ORM (e.g., [14,24,25]). It is unclear to what extent the languages agree on their underlying ontological foundations to model information. This limits mapping and transformation algorithms for 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. Moreover, a more detailed insight in the overlap and underlying modelling principles will contribute to investigating the effect of language features on modelling information.

To solve these issues, it first should be clear what entities and constraints exist in each language and how the differences can be reconciled without changing the languages. We achieve this by developing a single integrated metamodel inclusive of all language features; in particular, we describe in this work such a unifying metamodel for the static, structural components of UML 2.4.1 class diagrams, EER, and ORM2/FBM, which, to the best of our knowledge, is the first of its kind. We use insights from Ontology and ontologies during the metamodel development rather than the argument of convenience to fit with an *a priori* chosen logic language. The unification brings afore the differences and commonalities: 1) they agree on Relationship, Role (/association end), and Object type (/class/entity type), and 2) they differ in coverage of, among others, attributes. We provide an overview of related works in Section 2, describe the metamodel in Section 3, and conclude in Section 4.

2 State of the Art

Within CDM and software engineering, CDMs are compared through their metamodels (in ORM) highlighting their differences in [17], which is useful before unifying them. Most other works take a formal approach only. This can be by means of a single formalisation in a chosen logic by first formalizing a language (among others, [1,5,18,21,22,34]), and optionally in the scope of partial unification [8,27]. Perhaps due to this approach, different logics have been used for different CDM languages, therewith still not providing the sought-after interoperability for either of the languages or among each other; e.g., the Description Logic $\mathcal{ALUNI}$ is used for a partial unification [8] but *DL-Lite* and $\mathcal{DLR}_{\mathsf{ifd}}$ are used for partial formalisations to stay within the chosen decidable fragment [1,5,23], omitting features that render the language undecidable [23], so they cannot simply be linked up and implemented.

Other approaches include [41,6,7,2,3], which are more advanced in the notion of a common language. Venable and Grundy designed [41] and implemented a unification [13,42] for a part of ER and a part of NIAM (a precursor to ORM), omitting, mainly, value types, nested entity types, and composite attributes, and NIAM is forced to have the attributes as in ER. Bowers and Delcambre [6] present a framework for representing schema and data information coming from several data models, mainly relational, XML and RDF. Its main characteristic is a flat representation of schema and data, and the possibility of establishing different levels of conformance between them. However, its representational language ULD only includes ordinal, set and union class constructs, and cardinality constraints. Boyd and McBrien [7] use a Hypergraph Data Model to relate schemas represented in ER, relational, UML, and ORM, and includes transformation rules between them. It has the features inclusion, exclusion and union class constructs, and mandatory, unique and reflexive constraints, and various notions of cardinality constraints and keys, but roles, aggregation, and weak entity types are missing. Atzeni et al [2,3] describe an automatic approach that translates a schema from one model to another by means of a small set of "metaconstructs"—entities (called "abstracts"), attributes (called "lexicals"), relationships, generalization, foreign keys, and complex attributes—that can be used to characterize different models. Automatic translations between schemas are produced in Datalog, but translations from a rich representational language may require a sequence of such basic translations, if possible. Guizzardi [14] proposes a Unifying Foundational Ontology (UFO) which is used to redefine UML metamodel for structural conceptual modelling concepts.

Our approach is different regarding scope and methodology. We aim to capture *all* the languages' constructs and *generalise* in an ontology-driven way so that the integrated metamodel subsumes the static elements of EER, UML Class Diagrams v2.4.1, and ORM2 without changing the base languages. Methodologically, our metamodel is ontological rather than formal, compared to all other known works that present first a formal common language for translations that leave aside important particular aspects of each language. We first develop a conceptual model of all possible entities and their relations in the selected languages, and will devise a formalization for their translations afterward. The main benefit is that it allows one to have a clear comprehension of the meaning(s) of an entity in each language whilst coping with the broader scope. This is an essential step towards achieving the full potential of information sharing.

3 Ontology-driven metamodel

The metamodel is a conceptual model about the selected CDM languages that covers all their native features and is still consistent. Whether a particular feature is a good feature is beyond the scope, because we aim at representing in a unified way what is already present in the language. We do, however, use Ontology (philosophy) and ontologies (artifacts in IT and computing) to enhance understanding of the features and to unify perceived differences through generalization, and to improve the quality of the metamodel.

The principal entities (cf. constraints) are depicted in Fig. 1 in UML Class Diagram notation, where a white fill of a class icon indicates that that entity is not present in either of the three languages, a single diagonal fill that it is present in one language, a double diagonal that it is present in two, and a dark fill that it is present in all three groups of languages (EER, UML v2.4.1, ORM2); naming conventions and terminological differences and similarities of the entities are listed in the appendix at the end of the paper. Although UML Class Diagrams have limited expressiveness, we prefer a more widely known graphical notation for the purpose of communication (it will be formalised in a suitable logic anyway). The remainder contains explanations of the metamodel fragments, being roles, relationship, and attributes, class/entity type, nested and weak entity type, subsumption, and aggregation.

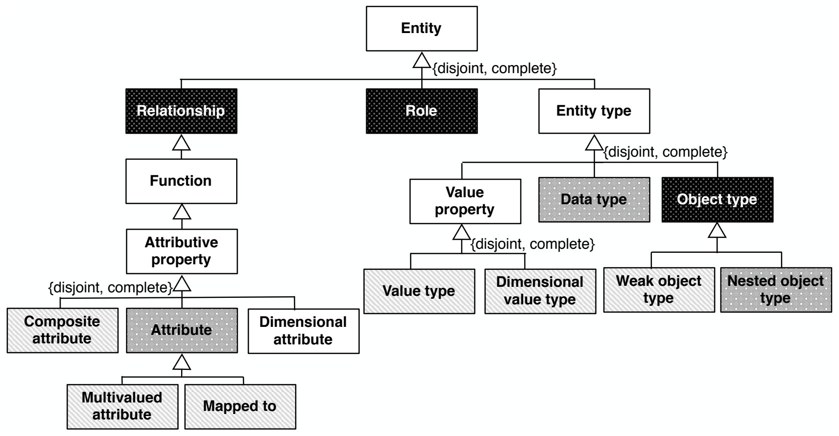

Fig. 1. Main static, structural, entities of the metamodel; see text for details.

3.1 Classes, concepts, types

CDMs use terms such as entity type, object type, and class. In philosophy, distinctions are made between classes, concepts, types, and universals. The meaning of these terms do not coincide, so in order to be more precise in the characterisation of these of core entities, we shall first briefly summarise a selection of the general, mostly agreed-upon, meanings, principally examined in Ontology (its aim is to clarify some possible terminological confusion and we take them at face-value, but will not debate which one is 'better' or how to formulate their description more precisely).

- *Class* is associated with set theory and the extension—a set of actual objects as members. Two distinct classes must have different extensions.
- *Concept* generally refers to a mind-dependent entity that may, or may not, have instances, and where there is not a membership relation between the individual object and its concept, but an instantiation relation between the object and the concept it instantiates [11].
- *Universals* are mind-independent entities, which do have at least one instance, hence, also uses the instantiation relation, not membership [30].
- *Properties* are "(also called 'attributes,' 'qualities,' 'features,' 'characteristics,' 'types') are those entities that can be predicated of things or, in other words, attributed to them." [39], which can be relational or unary.
- *Type* denotes a 'kind' of thing, and is closely associated with mathematics.
- *Unary predicate* is used in logic, and may have instances associated to it through the interpretation function (assuming a model-theoretic semantics).
- *Entity* means whatever can be deduced from the context in which the term is used, but such that that thing is a discrete unit, therewith it can refer to an instance or object, but also to a universal or concept or class.

In contrast to the extensional view and grouping similar things together into a class or set, concepts and universals concern the intension; informally: they are the descriptions, or a set of properties, that those entities that instantiate them, have. For instance, that each Computer has a CPU and memory, and has the function to compute, or that Apple is a kind of fruit that has a certain shape and colour. The main philosophical difference lies in the mind (in-)dependence and, with that, whether there is such thing as reality.

ORM, EER, and UML use different terminology, which might indicate different ontological commitments regarding the subject domain we wish to represent in the conceptual data model. In UML, "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." (p49) (emphasis added) [32]. For ORM, the draft ISO standardisation [10] declares that object type is a "concept used to classify the objects in the universe of discourse into different kinds" (p5), which is divided into non-lexical object type—an "object type, each of whose instances are non-lexical objects" (p5)—and lexical object type, which "is a metaclass each instance of which represents a concept that is used to represent values" (p12). The original ER diagram uses "entity sets" [9], known as entity types since soon thereafter [38], into which entities are classified and "we know that it has the properties common to the other entities in the entity set" (p11) [9] and "An *entity type E* defines a collection of entities that have the same attributes." (p1004) [38], hence, both extension and intension are considered. EER generally follows the same line, although it is formulated differently by [40]: "Entity types conceptualize structuring of things of reality through attributes." and "$E \doteq (attr(E), \Sigma_E)$ where E is the entity type defined as a pair – the set $attr(E)$ of attribute types and the set Σ_E of integrity constraints that apply to E." (p1084), i.e., definitely the intension, and it comprises both attributes and constraints explicitly. Despite the differences in formulation, practically, it does not make a real difference: in the conceptual data model, each one is used to denote a kind of thing where the relevant aspects of its intension is described, and will have an extension in the software (e.g., as object in OO software, or tuples in a database table) and each of those objects represents an instance in reality. However, one just as well can design a database about [mind-dependent] deities in the Stone Age. From the ontological viewpoint, we thus can postulate that the conceptual data models' entity type or class, is, mostly, a universal, but sometimes may be a concept and one can let them be subsumed by Entity. The terminology we use henceforth for those entities in conceptual data models that describe the intension, is Object type. Although UML is more widely used than either EER or ORM, hence, the term 'class' more familiar, what actually is being modelled/represented, is the intension with respect to the application domain, not a placeholder for the extension, and that term does not clash in intention with terminology in ontology.

Finally, Weak object type is included thanks to ER and EER's weak entity type that has an identification that is a combination of one or more of its attributes and the identifier of its related strong entity type. Depending on the ontological status of weak entity type [14,25], it is possible to approximate its meaning in UML with the Identifier profile or use ORM's compound reference scheme, but there is no icon for it in UML and ORM.

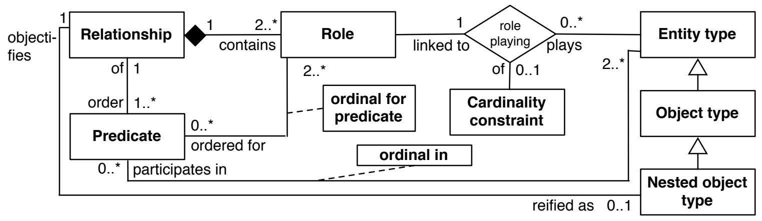

Fig. 2. Relationships between Relationship, Role, and Entity type; see text for details.

3.2 Roles and relationships

There are many points that can be discussed about roles and relationship, but we shall restrict ourselves to their definition, and differences among them and with a object type, how to deal with the aggregation associations, and subsumption.

Distinguishing roles, relationships, and predicates A relationship, or relational property in ontology [39], is an entity that relates entities, hence, it requires at least two entities to participate in it. Thus, there are no unary relationships, whereas there can be unary predicates and object types are unary. Second, a relationship is composed of at least two *roles*, or: "association ends" or "member ends" in UML [32], ORM/FBM use "roles" [18,19,10], and EER's roles may be called components of a relationship [9,40]. A role is something that an object plays in a relationship, therewith characterising relationship and committing to the *positionalist* ontological commitment of relations and relationships as to what they are (see also [29,24]). The three CDM languages agree on this and therefore they appear in the metamodel (recall Fig. 1), including the disjointness between Relationship, Role, and Entity Type and their interaction is depicted in Fig. 2. The ternary role playing enforces that each role must have exactly one entity type with no or one cardinality constraint (the minimum and maximum cardinality are part of the Cardinality constraint), and each entity type may play zero or more roles.

ORM's predicates are an addition to roles and relationships and adheres to the so-called "standard view" of relations [29]. They have an intricate relation with roles and relationships: 1) an ordered for between roles, 2) a predicate can exist only if there is a relation between those roles that compose the relationships, and the relationship that that predicate is an ordering of (i.e., it is a join-subset), and 3) entities that participate in the predicate must play those roles that compose the relationship of which that predicate is an ordered version of.

Nested object types A Nested object type (see Fig. 1) is also called association class, associative entity, or objectified fact type. Although documentation of CDM languages discuss the notion of a "duality" of nested object types as both being a relationship and an object type, and therewith indicating that one

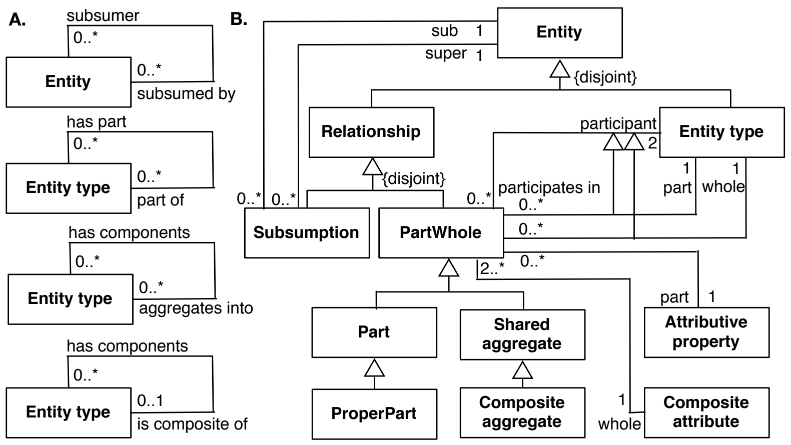

Fig. 3. Subsumption and aggregation. A: short-hand notation; B: more detailed representation, where there is an additional constraint on the two associations of subsumption such that the participating entities must be of the same type, and an attributive property participates only in a part-whole relation if it is part of a composite attribute.

could have multiple inheritance of Nested object type to two supertypes, Relationship and Object type (e.g., [15]), this is not correct. A nested object type is *composed of* a relationship or the outcome of a *transformation of* or *reification of* a relationship, each one of which is certainly distinct from *being* a relationship. Therefore, the metamodel relates them through a normal association.

The constraints regarding nested object types are rather basic, which reflects the flexibility concerning reification/objectification in UML and ORM. No restrictions are mentioned in the UML standard regarding objectifying an association into an association class [32], and although ORM initially had some restrictions on when one would be allowed to objectify a fact type [19], these restrictions have been lifted more recently [16].

Subsumption and aggregation relationships Considering relationships in more detail, we have to address subsetting relationships versus subtyping, and aggregation. The two relationships in Fig. 3 are represented twice: once as recursive relationship, and once as being a kind of relationship that has two participating entities, each providing a different perspective on the relationship.

Subsumption. Subsumption of entities is included in all languages except ER. Some argue against allowing multiple inheritance [35,37], but there it is assumed that there are no real relationships (an OBO-language limitation) and/or that subtypes must always be disjoint (an Aristotelean left-over), which need not be the case for some subject domain and, moreover, CDM languages permit it. Thus, the constraints in the metamodel reflects this.

Subsumption for relationships and roles is more interesting. UML 2.4.1 distinguishes between subsetting—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 [32]. Specialization is not set-oriented, but because of the differences in intension of the association [32], although the UML standard does not describe how that is supposed to work. The only way to change an association's intension, is to restrict the relational properties of an association, as recognised in ontology [26]; e.g., each relationship that is asymmetric is also irreflexive. Only few such subsumptions exists [19], however, and little is known about its practicality other than the few experiments reported in [26] for ontologies. Nevertheless, it may become more relevant in the near future, and therefore we keep this option available. Both ways of relationship subsumption are captured in the metamodel with the more general Subsumption, and both UML and ORM include subsumption of roles, therefore, the participating entities for Subsumption are Entity.

Aggregation. Much has been written about UML's shared and composite aggregation (among many, [14,28,33]), yet the UML standard still does not offer any more clarity on what they really are. Shared aggregation tends to be mapped loosely onto parthood and composite aggregation to proper parthood, yet aggregation is also used for meronymic associations in UML Class Diagrams, such as *member-of* (see [28] for an overview); hence, one can neither draw a subsumption relation between the aggregations and parthood nor adorn the subsumption from PartWhole with a disjointness axiom. The UML standard's description of shared and composite aggregation also indicates behavioural characteristics or lifecycle semantics of the part and whole, but its inclusion in the metamodel is beyond the scope of the static components.

Finally, observe the possible participation of attributes in a PartWhole relationship, which is added to accommodate ER/EER's Composite attribute. Given that the entity that plays the part-role is Attributive property, it entails that it is still possible to include uncommon representations, such as nested composite attributes or a composite attribute with a multivalued attribute.

Attributes and value types Ontological aspects of attributes deserve a separate paper; here we only summarise the outcome. Formally, an *attribute* (A) is a binary relationship between a *relationship* or *object type* ($R \cup E$) and a *data type* (D, sometimes called 'concrete domains'), i.e., $A \mapsto R \cup E \times D$; e.g., an attribute $\mathsf{hasColour} \mapsto \mathsf{Flower} \times \mathtt{String}$.

The different attributes in Fig. 1 reflect their differences in meaning in the CDM languages and it can be motivated by Ontology and ontologies. An attribution, or quality (in ORM called Value types), such as the Colour of an apple or its Shape, 'inheres in' (needs a) bearer to exist and is formalised as a unary predicate, not a binary. They differ from what we call here Object Type, and philosophers agree on this. This distinction is also reflected in the foundational ontologies; e.g., GFO has Property (the attribution) and Presential/Occurrent (the object or relation), which is refined into atomic and non-atomic attributes [20] that resemble EER's simple and composite attributes.

UML uses the standard definition and meaning of attribute and it is normally modelled 'inside' a class- or association-icon as, e.g., "hasColour:String",

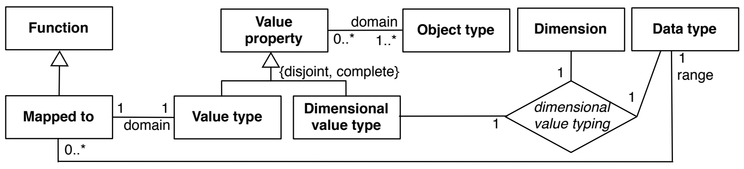

Fig. 4. Value properties, distinguishing between plain and dimensional value types.

although hasColour may be drawn also as an association with at the far end a class-icon for the data type [32]. ER and EER have partial attributes: no ER or EER diagram notation lets the modeller specify the data type of an attribute, because declaring datatypes is carried out at the physical design stage of database development. Although the 'bubble notation' of EER could give an impression that an attribute is an unary entity of itself, the understanding from its formal foundation [9,40] is that the attribute is alike UML's attribute. ER and EER contain two additional types of attributes: composite and multivalued attributes even though both can be remodelled as basic attributes.

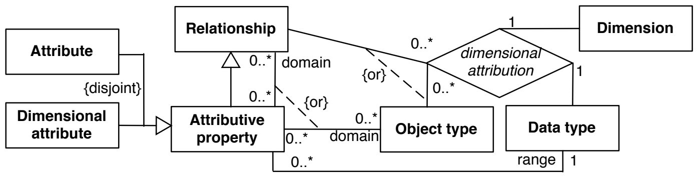

Fig. 5. Simple and dimensional attributes; Dimensional attribute is reified version of the ternary relation dimensional attribution.

ORM is said to be "attribute-free" [19], but it does have attributes in the strict sense of the meaning. ORM's value type differs from object type (ORM entity type) in that there is a behind-the-scenes software-generated "mapped to" relationship [10], being mapped_to $\mapsto$ ValueType $\times$ `DataType`; e.g., hasColour is asserted between Flower and Colour, and then a mapped_to $\mapsto$ Colour $\times$ `String`. The crucial differences between UML's and ORM's attributes, are that ORM uses three entities with two binary relationships where the unary attribution (value type) can be reused, whereas UML collapses it all into one binary relationship and keeps the attribute 'inside' the class.

Finally, ORM's CASE tools lets one specify not only the data type when declaring a value type, but also, if desired, the dimension of the measurement, such as cm or day; thus, adding meaning to the values. This has not been specified in a metamodel or formalised yet. One can model this as a ternary—e.g., as hasHeight $\mapsto$ Flower $\times$ `Integer` $\times$ cm—or as three relations between object type, and value type, value type and data type, and data type and dimension. For our metamodel, we chose for the more precise ternary relation dimensional value typing, as shown in Fig. 4. For symmetry, we also added its analogue for attributes (Fig. 5), although the UML standard does not mention dimensions explicitly.

4 Conclusions

We presented a unifying metamodel capturing ORM, EER, and UML v2.4.1 with respect to their static, structural, entities and their relationships. Strictly, the only intersection of features among all these CDM languages are role, relationship (including subsumption), and object type. Attributions are represented differently, but, ontologically, they aim to represent the same. Several implicit aspects, such as dimensional attribute and its reusability and relationship versus predicate, have been made explicit.

The complete metamodel with the section for the constraints has been developed, but not described here due to space limitations, and likewise for more detailed justifications for the modeling decisions taken. We will use the metamodel for the formalisation, where it will aid in the comprehension of differences between CDM languages and in the development of tools.

Acknowledgements This work is based upon research supported by the National Research Foundation of South Africa (Project UID: 80584) and the Argentinian Ministry of Science and Technology.

References

1. Artale, A., et al.: Reasoning over extended ER models. In: Proc. of ER'07. LNCS, vol. 4801, pp. 277–292. Springer (2007)
2. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. VLDB Journal 17(6), 1347–1370 (2008)
3. Atzeni, P., Gianforme, G., Cappellari, P.: Data model descriptions and translation signatures in a multi-model framework. AMAI 63, 1–29 (2012)
4. Banal-Estanol, A.: Information-sharing implications of horizontal mergers. International Journal of Industrial Organization 25(1), 31–49 (2007)
5. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artificial Intelligence 168(1-2), 70–118 (2005)
6. Bowers, S., Delcambre, L.M.L.: Using the uni-level description (ULD) to support data-model interoperability. Data & Knowledge Engineering 59(3), 511–533 (2006)
7. Boyd, M., McBrien, P.: Comparing and transforming between data models via an intermediate hypergraph data model. J. on Data Semantics IV, 69–109 (2005)
8. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. Journal of Artificial Intelligence Research 11, 199–240 (1999)
9. Chen, P.P.: The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
10. Committee Members: Information technology – metamodel framework for interoperability (MFI) – Part xx: Metamodel for Fact Based Information Model Registration (Draft release date: 2012-04-18 2012), iSO/IEC WD 19763-xx.02
11. Earl, D.: The classical theory of concepts. In: Internet Encyclopedia of Philosophy (2005), http://www.iep.utm.edu/c/concepts.htm

12. Fillottrani, P.R., Franconi, E., Tessaris, S.: The ICOM 3.0 intelligent conceptual modelling tool and methodology. Semantic Web Journal 3(3), 293–306 (2012)
13. Grundy, J., Venable, J.: Towards an integrated environment for method engineering. In: Proceedings of the IFIP TC8, WG8.1/8.2 Method Engineering 1996 (ME'96). vol. 1, pp. 45–62 (1996)
14. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Phd thesis, University of Twente, The Netherlands. Telematica Instituut Fundamental Research Series No. 15 (2005)
15. Guizzardi, G., Wagner, G.: Using the unified foundational ontology (UFO) as a foundation for general conceptual modeling languages. In: Theory and Applications of Ontology: Computer Applications, pp. 175–196. Springer (2010)
16. Halpin, T.: Objectification of relationships. In: Proc. of EMMSAD'05. CEUR-WS (2005), 13-14 June, 2005, Porto, Portugal
17. Halpin, T.A.: Advanced Topics in Database Research, vol. 3, chap. Comparing Metamodels for ER, ORM and UML Data Models, pp. 23–44. Idea Publishing Group, Hershey PA, USA (2004)
18. Halpin, T.: A logical analysis of information systems: static aspects of the data-oriented perspective. Ph.D. thesis, University of Queensland, Australia (1989)
19. Halpin, T.: Information Modeling and Relational Databases. San Francisco: Morgan Kaufmann Publishers (2001)
20. Herre, H.: General Formal Ontology (GFO): A foundational ontology for conceptual modelling. In: Theory and Applications of Ontology: Computer Applications, chap. 14, pp. 297–345. Springer (2010)
21. Hofstede, A.H.M.t., Proper, H.A.: How to formalize it? formalization principles for information systems development methods. Information and Software Technology 40(10), 519–540 (1998)
22. Kaneiwa, K., Satoh, K.: Consistency checking algorithms for restricted UML class diagrams. In: Proc. of FoIKS'06. Springer (2006)
23. Keet, C.M.: Prospects for and issues with mapping the Object-Role Modeling language into $\mathcal{DLR}_{ifd}$. In: Proc. of DL'07. CEUR-WS, vol. 250, pp. 331–338 (2007)
24. Keet, C.M.: Positionalism of relations and its consequences for fact-oriented modelling. In: Proc. of ORM'09. LNCS, vol. 5872, pp. 735–744. Springer (2009)
25. Keet, C.M.: Enhancing identification mechanisms in UML class diagrams with meaningful keys. In: Proc. of SAICSIT'11. pp. 283–286. ACM Conference Proceedings (2011), cape Town, South Africa, October 3-5, 2011
26. Keet, C.M.: Detecting and revising flaws in OWL object property expressions. In: Proc. of EKAW'12. LNAI, vol. 7603, pp. 252–266. Springer (2012)
27. Keet, C.M.: Ontology-driven formal conceptual data modeling for biological data analysis. In: Biological Knowledge Discovery Handbook: Preprocessing, Mining and Postprocessing of Biological Data, chap. 6, p. in press. Wiley (2013)
28. Keet, C.M., Artale, A.: Representing and reasoning over a taxonomy of part-whole relations. Applied Ontology 3(1-2), 91–110 (2008)
29. Leo, J.: Modeling relations. Journal of Philosophical Logic 37, 353–385 (2008)
30. MacLeod, M.C., Rubenstein, E.M.: Universals. In: The Internet Encyclopedia of Philosophy (2005), http://www.iep.utm.edu/u/universa.htm
31. Mendes Calo, K., Cenci, K.M., Fillottrani, P.R., Estevez, E.C.: Information sharing-benefits. Journal of Computer Science & Technology 12(2) (2012)
32. Object Management Group: Superstructure specification. Standard 2.4.1, Object Management Group (2012), http://www.omg.org/spec/UML/2.4.1/
33. Odell, J.: Advanced Object-Oriented Analysis & Design using UML. Cambridge: Cambridge University Press (1998)

34. Queralt, A., Teniente, E.: Decidable reasoning in UML schemas with constraints. In: Proc. of CAiSE'08. LNCS, vol. 5074, pp. 281–295. Springer (2008)
35. Rector, A.: Modularisation of domain ontologies implemented in description logics and related formalisms including OWL. In: Proc. of K-CAP'03. pp. 121–129 (2003)
36. Rosenthal, A., Mork, P., Li, M.H., Stanford, J., Koester, D., Reynolds, P.: Cloud computing: a new business paradigm for biomedical information sharing. Journal of Biomedical Informatics 43(2), 342–353 (2010)
37. Smith, B.: Beyond concepts, or: Ontology as reality representation. In: Varzi, A., Vieu, L. (eds.) Proc. of FOIS'04. pp. 73–84. Amsterdam: IOS Press (2004)
38. Song, I.Y., Chen, P.P.: Entity relationship model. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, vol. 1, pp. 1003–1009. Springer (2009)
39. Swoyer, C.: Properties. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Stanford, winter 2000 edn. (2000), `http://plato.stanford.edu/archives/win2000/entries/properties/`
40. Thalheim, B.: Extended entity relationship model. In: Liu, L., Özsu, M.T. (eds.) Encyclopedia of Database Systems, vol. 1, pp. 1083–1091. Springer (2009)
41. Venable, J., Grundy, J.: Integrating and supporting Entity Relationship and Object Role Models. In: Proc. of ER'95. LNCS, vol. 1021, pp. 318–328. Springer (1995)
42. Zhu, N., Grundy, J., Hosking, J.: Pounamu: a metatool for multi-view visual language environment construction. In: IEEE Conf. on Visual Languages and Human-Centric Computing 2004 (2004)

Appendix: naming conventions. The following naming conventions are used, in the order of Metamodel, UML v2.4.1, EER, ORM/FBM. Where no value is given, that element is absent.

- Relationship; association, can be 2-ary according to the MOF 2.4.1, but also >2-ary [32]; relationship, ≥2-ary; atomic/compound fact type, ≥1-ary.
- Predicate; ; ; predicate.
- Role; association end / member end; component of a relationship; role.
- Entity type; classifier; ; object type.
- Object type; class; entity type; non-lexical object type / entity type.
- Attribute; attribute; attribute, but without a data type in the diagram; (represented differently).
- Dimensional attribute; (no recording of dimension); ; (represented differently).
- Composite attribute; a property can be a composite of another property ; composite attribute; implicitly present by adding new roles.
- Multivalued attribute; ; multivalued attribute; .
- Value type; ; ; lexical object type / value type, without dimension.
- Dimensional value type; ; ; lexical object type / value type, with dimension.
- Data type; Data type, LiteralSpecification; ; data type.
- Object subtype; subclass; subtype; subtype.
- Sub-relationship; subsetting or subtyping of association; subtyping the relationship ; subset constraint on fact type.
- Nested object type; association class; associative entity type; objectified fact type.
- Weak object type; ; weak entity type; .
- Composite aggregate; composite aggregation; ; .
- Shared aggregate; shared aggregation; ; .