Formalization of ORM Revisited

Terry Halpin

INTI International University, Malaysia and LogicBlox, Australia
terry.halpin@logicblox.com

Abstract. Fact-oriented modeling approaches such as Object-Role Modeling (ORM) and Natural Language Information Analysis Method (NIAM) enable conceptual information models to be expressed using graphical diagrams that may be assigned formal semantics by mapping them onto sets of logical formulae. Various formalizations for such mappings exist. This paper extends such previous work by providing a new approach to formalizing second generation ORM (ORM 2). We show that the metalevel association between semantic value type and data type must be a mapping relationship rather than a subtyping relationship, and we axiomatize a special representation relationship to support this mapping at the instance level. Our new formalization includes coverage of preferred reference schemes and additional constraints introduced in ORM 2. Other issues examined briefly include the use of finite model theory, sorted logic, and practical choices for implementing certain kinds of logical formulae as constraints or derivation rules.

1 Introduction

For conceptual data modeling, fact-oriented approaches such as Object-Role Modeling (ORM) [11,15] and Natural Language Information Analysis Method (NIAM) [23] differ from Entity Relationship Modeling (ER) [3] and class modeling in the Unified Modeling Language (UML) [22] by uniformly modeling facts as unary or longer relationships that are instances of fact types (e.g. Person smokes, Person was born on Date) instead of modeling only some facts as relationships and others as instances of attributes (e.g. Person.isSmoker, Person.birthdate). This attribute-free approach enables all facts, constraints, and derivation rules to be verbalized naturally in sentences easily understood and validated by nontechnical business users using concrete examples, and promotes semantic stability, since one never needs to remodel existing structures in order to add facts about attributes [9]. ORM's graphical notation for data modeling is also far more expressive than that of industrial ER diagrams or UML class diagrams [15].

To ensure that fact-oriented models and queries are unambiguous and executable, it is essential to formalize them in terms of underlying logics. The first formalization of a fact-oriented approach appeared in 1989 in our doctoral thesis [5], which provided an algorithm to map an extended version of NIAM to predicate logic in which deduction trees (semantic tableaux augmented by natural deduction) were used to test whether a conceptual schema is strongly satisfiable (consistent when populated) and

P. Herrero et al. (Eds.): OTM 2012 Workshops, LNCS 7567, pp. 348–357, 2012.
© Springer-Verlag Berlin Heidelberg 2012

whether one schema implies or is equivalent to another, using conservative extensions to define predicates unique to the other schema prior to the evaluation.

In the 1990s, other formalizations of fact-oriented modeling were produced (e.g. [4, 17]), and we extended NIAM further to become ORM. In 2005 we introduced second generation ORM (ORM 2) [6], which modified the graphical notation to remove all English-specific symbols, and added features (e.g. role value constraints, semiderived fact types, asserted subtypes, and deontic constraints). Recently, various researchers have proposed ways to transform many ORM 2 constructs into description logics (e.g. [18]), noting that some features such as acyclic ring constraints have no such mapping. In 2011, ORM 2 support for ring constraints was modified and extended to cater for local reflexivity, transitivity, and strong intransitivity [14]. From now on, we use "ORM" to mean the current version of ORM 2.

The rest of this paper focuses on formalization issues for ORM, and is structured as follows. Section 2 discusses the top level partition of types, including a new treatment for relating semantic values to data values. Section 3 illustrates how fact types, fact instances, and constraints and reference schemes are formalized, including an example relating to ORM's adoption of finite model theory. Section 4 briefly discusses asserted, derived and semiderived fact types and subtypes, and practical choices for implementing certain kinds of logical propositions as constraints or derivation rules. Section 5 concludes by summarizing the main contributions, identifying areas for future research, and listing the references cited.

2 Entities, Semantic Values, and Data Values

Our original formalization [5] treated named value types (e.g. CountryCode) as subtypes of conceptual datatypes (e.g. String), as shown in the metamodel fragment Figure 1(a). Our new formalization treats semantic values as ontologically distinct from data values, replacing the subtyping relationship by a mapping relationship as in Figure 1(b).

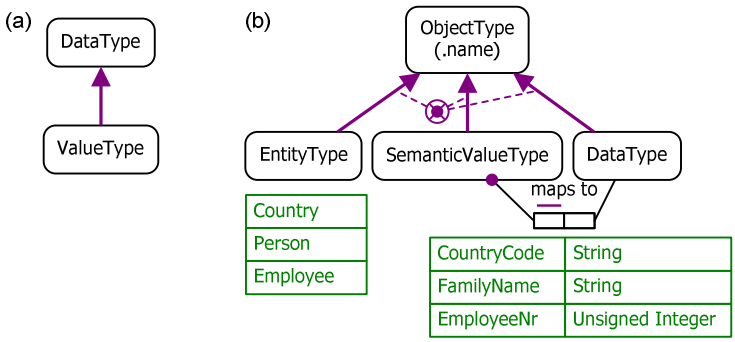

Fig. 1. Value types and data types in (a) original formalization, and (b) new formalization

In ORM, an object is the same as an individual in logic, and is either an entity, a semantic value or a data value. Figure 1(b) includes fact tables with sample populations of the meta-object types and the meta-fact type SemanticValueType maps to DataType. A complete treatment of this mapping includes discussion of finer aspects about data types, such as facets (e.g. assigning country codes a fixed length of 2 characters), but such aspects are ignored in this paper.

Our main reason for distinguishing between semantic values and data values is to ensure a formalization that is consistent with the *Principle of Indiscernibility of Identicals* (*PII:* identical objects have exactly the same properties). Using Φ as variable ranging over predicates, this may be formalized in second-order logic as follows: $\forall x,y\ [x = y \rightarrow \forall\Phi\ (\Phi x \equiv \Phi y)]$. In first-order logic this corresponds to the Substitutivity of Identicals (SI) inference rule.

For example, consider the following objects: the country code 'CH', and the IATA airline code 'CH'. Are these objects identical? If we adopt the subtyping semantics in in Figure 1(a), they are identical, since each is just the character string 'CH'. The country code 'CH' was chosen for Switzerland because the Latin name for Switzerland is '*Confederatio Helvetica*', so sentence (1) below expresses a true proposition. The IATA airline code 'CH' is used for Bemidji airlines (a small airline company based in Bemidji, Minnesota), but the choice of this airline code has nothing to do with Latin. Hence, sentence (2) below expresses a false proposition.

(1) The country code 'CH' is based on a term in Latin.
(2) The IATA airline code 'CH' is based on a term in Latin.

Since the property of being based on a Latin term holds for the country code 'CH' but not the IATA airline code 'CH', it follows from PII that the country code 'CH' is *not* identical to the IATA airline code 'CH'. Hence the subtyping pictured by the Euler diagram in Figure 2 is *incorrect* (the string 'CH' cannot be both a country code and an IATA airline code). Though the codes are not identical, their mapped data values may be equated. Any logical attempt to deny our distinction between semantic values and data values must move the semantics of the type names into the associated predicates (e.g. CountryCode is a country code based on a term in Language), but this option is clearly inferior as it hinders reuse and leads to unnatural verbalization. For some related discussion on entity/value distinctions in various modeling approaches see [10].

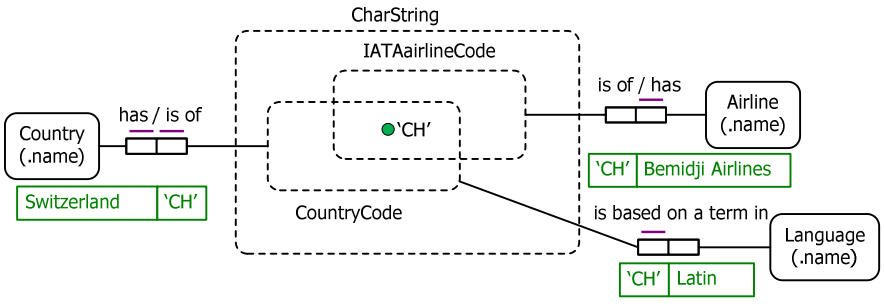

Fig. 2. A diagram that wrongly treats a semantic value as identical to a data value

Fig. 3. Graphic notation for partitioning objects into entities, semantic values and data values

At the business domain level, the type partitioning in Figure 1(b) is displayed graphically by using different line styles for the named, soft rectangles that denote the object types. Entity types use solid lines, semantic values types used dashed lines, and data types use dotted lines, as shown in Figure 3. The graphical display of data types is being added to ORM at the time of writing. ORM satisfies finite model theory [20], so for each state of the model the *population* of each of its types is finite. Although the usual axioms for data types allow infinite sets (e.g. each integer has a successor), ORM models use data values only to represent semantic values, and data types are implicitly independent. In any state, the semantic value population is finite, and hence so is the data value population in use. As in datalog [1], syntactic restrictions are placed on derivation rules and queries in ORM to ensure that use of data type operations (e.g. numeric addition or string concatenation) do not generate infinite sets.

Every ORM model implicitly includes the unary type predicates *Entity*, *SemanticValue*, and *DataValue* to convey the kind of object type. Hence the schema fragment shown in Figure 3 may be formalized as follows. In our logical notation, type predicates are written in prefix notation without parentheses. For example, "Country x" abbreviates Country(x) and is read "x is a country".

$\forall x$ (Country $x \rightarrow$ Entity x)
$\forall x$ (CountryCode $x \rightarrow$ SemanticValue x)
$\forall x$ (String $x \rightarrow$ DataValue x)

A model comprises a schema plus a population. Figure 4(a) shows an example of the simplest kind of populated ORM model: a semantic value type populated with one instance. The enclosing single quotes on the value instance indicate that the code is represented by a character string value, as shown explicitly in Figure 4(b). For explanation purposes, the *typed constant* "CountryCode 'US'" is used here for the country code in the representation relationship, to distinguish it from the character string 'US'. In typical ORM diagrams, the representation relationship is suppressed, and semantic values are normally displayed in context without their type name, as in Figure 4(a).

The identity relationship "=" is defined between objects of any type. This may be formalized by using *Object* for a universal type predicate, and adding the axiom $\forall x,y$ [$x = y \rightarrow$ (Object x & Object y)]. The usual axioms for identity (e.g. Reflexive, Symmetric, Transitive, and SI) apply. We reserve the predicate symbol $\approx$ for the *representation relationship* that provides an injective (mandatory 1:1 into) mapping from semantic values of a given type to data values, so "$x \approx y$" is read "x is represented by y".

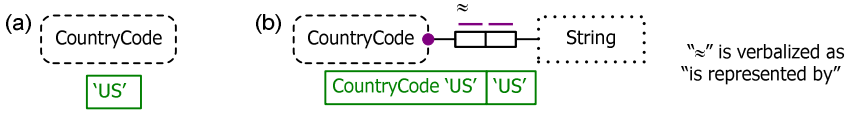

Fig. 4. A simple ORM model displayed in (a) compact form, and (b) expanded form

The representation relationship ≈ is axiomatized as follows, to allow its reuse between many different kinds of semantic values and data values. Since ≈ is predefined and is the same for all semantic value types, this formally allows us to treat semantic values simply as typed constants (e.g. "CountryCode 'CH'" may be unabbreviated to the definite description "**the** CountryCode **that** is represented by **the** String 'CH'"). Axioms A2 and A3 capture the mandatory role and uniqueness constraints for ≈ using the numeric quantifiers "$\exists^1$" (there is exactly one) and "$\exists^{0..1}$" (there is at most one). These may be unabbreviated to expressions using simple quantifiers and the identity relation thus: $\exists^{0..1}x\ \Phi x =_{df} \forall x,y\ [(\Phi x\ \&\ \Phi y) \rightarrow x = y]$; $\exists^1 x\ \Phi x =_{df} .. \exists x\ [\Phi x\ \&\ \forall y\ (\Phi y \rightarrow y = x)]$.

A1 $\forall x,y\ [x \approx y \rightarrow (\text{SemanticValue}\ x\ \&\ \text{DataValue}\ y)]$
A2 $\forall x\ (\text{SemanticValue}\ x \rightarrow \exists^1 y\ x \approx y)$
A3 $\forall y,T\ \exists^{0..1} x\ (Tx\ \&\ x \approx y)$ 2^{nd}-order axiom, where T is a type variable

For any given ORM model, the axiom A3 may be implemented in first-order logic by replacing the quantification over types by a conjunction over the finite set of domain type predicates (e.g. each string 'US' represents at most one countrycode and at most one pronoun etc.). The model in Figure 4(b) may now be formalized as follows. Here 'US' is an individual constant denoting itself (the character string 'US').

$\forall x\ (\text{CountryCode}\ x \rightarrow \text{SemanticValue}\ x)\ \&\ \forall x\ (\text{String}\ x \rightarrow \text{DataValue}\ x)$
$\forall x,y\ [(\text{CountryCode}\ x\ \&\ x \approx y) \rightarrow \text{String}\ y]$
$\exists x\ (\text{CountryCode}\ x\ \&\ x \approx \text{'US'})$

If we expand the model as shown in Figure 5, the additional part of the model may be formalized as follows. Although CountryCode and Pronoun are mutually exclusive types, their data types are compatible, so one may perform lexical comparisons between their instances by comparing the data values that represent those instances.

$\forall x\ (\text{Pronoun}\ x \rightarrow \text{SemanticValue}\ x)$
$\forall x,y\ [(\text{Pronoun}\ x\ \&\ x \approx y) \rightarrow \text{String}\ y]\ \&\ \exists x\ (\text{Pronoun}\ x\ \&\ x \approx \text{'US'})$

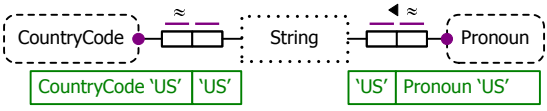

Fig. 5. A populated ORM model with two semantic value types based on the data type String

ORM includes Formal ORM Language (FORML), a textual language for verbalizing ORM models in semi-natural language that is intelligible to nontechnical users [12]. Its latest version, FORML 2, is designed for both input and output [16]. To facilitate compact expression in this language, mapping from entities to semantic values, and from semantic values to data values is often implicitly performed. For example, given the refmode declaration Gender(.code), the condition "Gender 'M'" is implicitly expanded to "Gender **that** has GenderCode **that** ≈ 'M'". Moreover, a user comparison between semantic values (e.g. CountryCode = Pronoun) is interpreted as a comparison between their data value representations, as this is the likely intent. In practice, such mappings can be implemented efficiently (e.g. using tagged types and autoboxing/unboxing).

3 Facts, Fact Types, Constraints and Reference Schemes

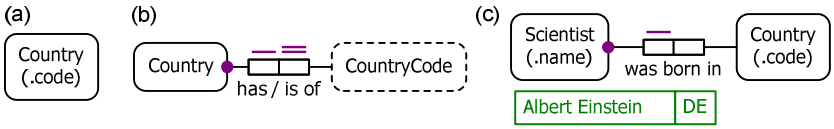

Fig. 6. A refmode in (a) compact and (b) expanded form, and (c) a populated fact type

Figures 6(a) and 6(b) display an entity type with a reference mode, in compact and expanded forms. The short predicate reading "has" is convenient for compact display, but for formalization this is auto-expanded to the predicate name "hasCountryCode". In principle, one could instead use "has" as the full predicate name, but this would often require long disjunctive type declarations for predicates, e.g. $\forall x,y$ (x has $y \rightarrow$ [(Country x & CountryCode y) $\vee$ (Scientist x & ScientistName y) $\vee$ …]). If necessary, short predicate readings are auto-expanded to fact type readings to ensure they are distinct. For example, the inverse predicate reading "is of" expands to "CountryCodeIsOfCountry" An ORM fact type corresponds to a set of one or more typed predicates. Binary predicates with no front or trailing text are placed in infix position. In general, ORM supports mixfix predicates of any arity. This schema may be formalized thus:

$\forall x$ (Country $x \rightarrow$ Entity x) & $\forall x$ (CountryCode $x \rightarrow$ SemanticValue x)
$\forall x,y$ [x hasCountryCode $y \rightarrow$ (Country x & CountryCode y)]
$\forall x$ (Country $x \rightarrow \exists^{1} y$ x hasCountryCode y)
$\forall y$ (CountryCode $y \rightarrow \exists^{0..1} x$ x hasCountryCode y)
$\forall x,y$ (x CountryCodeIsOfCountry $y \equiv y$ hasCountryCode x)
$\forall x$ (x isIdentified $\rightarrow$ Entity x)
$\forall x,y$ [x hasCountryCode $y \rightarrow x$ isIdentified]

The notion of *preferred reference* (indicated by the double uniqueness bar) is formalized by reserving the postfix predicate isIdentified for "is identified by a preferred reference scheme", allowing its use for any entity by the penultimate declaration above, and then indicating a fact type (as in the last formula above), or a conjunction or disjunction of fact types, used to provide the identification.

Figure 6(c) displays a binary fact type and sample fact. The entity type Scientist and its refmode may be formalized in a similar way to Country. The below formalization of the fact type Scientist was born in Country using the predicate name "was born in" assumes this short predicate reading is not used for another predicate in the schema (e.g. in fact types such as Horse was born in Country or "Person was born in City"). If this assumption does not hold, expand the predicate name with object type names as needed. Such expansions have no impact on ORM diagrams (which use predicate readings not predicate names) and can often be avoided by renaming (e.g. rename Horse was born in Country to Horse was foaled in Country). For discussion of an alternative approach allowing reuse of predicates with disjunctive type arguments see [8].

$\forall x,y$ [x was born in $y \rightarrow$ (Scientist x & Country y)]
$\forall x$ (Scientist $x \rightarrow \exists^{1} y$ x was born in y)

In ORM, an object type is said to be a top-level object type if it is not a subtype of any domain object type (excluding Entity, SemanticValue, DataValue, and Object if included). In ORM, top-level entity types are implicitly mutually exclusive, so if Scientist and Country in Figure 6(c) are top-level, the following constraint applies implicitly: $\forall x$ (Scientist x $\rightarrow$ ~Country x). Similarly, top-level semantic value types are implicitly mutually exclusive, so if ScientistName and CountryCode are top-level, the following constraint applies implicitly: $\forall x$ (ScientistName $x \rightarrow$ ~CountryCode x).

Given the schema formulae, *fact populations* may be formalized as existential assertions. We allow individual variables to start with any letter and include more letters or digits, permitting suggestive individual variable names. For example, using "*s*" for scientist, "*sn*" for scientist name, "*c*" for country, and "*cc*" for country code, the fact depicted in Figure 6(c) may be formalized thus: $\exists s,sn,c,cc$ (s hasScientistName sn & $sn \approx$ 'Albert Einstein' & c hasCountryCode cc & $cc \approx$ 'DE' & s wasBornIn c).

Most ORM constraints may now be formalized using formulae that are basically equivalent to those used in our doctoral thesis [5], but supplemented where needed by preferred reference declarations. For example, the preferred external uniqueness constraint in Figure 7(a) may be formalized thus:

$\forall c,sc\ \exists^{0..1} s$ (s isInCountry c & s hasStateCode sc)
$\forall s,c,sc$ [(s isInCountry c & s hasStateCode sc) $\rightarrow$ s isIdentified]

ORM allows reference schemes of arbitrary complexity, which may include entity-to-entity relationships. As a trivial example, the schema for President in Figure 7(b), which identifies a president by the country of which he/she is president, may be formalized using techniques already discussed. As a recent extension to ORM 2, Figure 7(c) shows a disjunctive reference scheme in which a chief officer is identified by the company to which he/she bears exactly one of two relationships, e.g. "the CEO of Megasoft" or "the CTO of Megasoft". The reference scheme may be formalized thus:

$\forall x,y$ [x is chief executive officer of $y \rightarrow$ (ChiefOfficer x & Company y)]
$\forall x,y$ [x is chief technical officer of $y \rightarrow$ (ChiefOfficer x & Company y)]
$\forall x$ [ChiefOfficer $x \rightarrow \exists^{1} y$ (x is chief executive officer of y
$\vee$ x is chief technical officer of y)]
$\forall x,y$ [x is chief executive officer of $y \rightarrow$ ~$\exists z$(x is chief technical officer of z)]
$\forall y$ (Company $y \rightarrow \exists^{0..1} x$ x is chief executive officer of y)
$\forall y$ (Company $y \rightarrow \exists^{0..1} x$ x is chief technical officer of y)
$\forall x,y$ [(x is chief executive officer of $y \vee x$ is chief technical officer of y)
$\rightarrow$ x isIdentified]

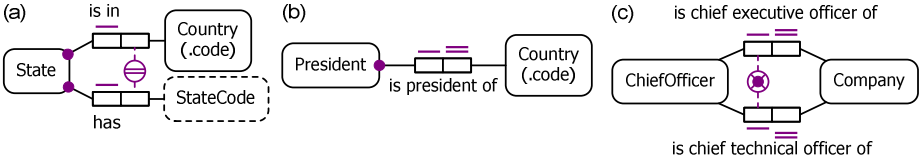

Fig. 7. Some different kinds of preferred reference schemes

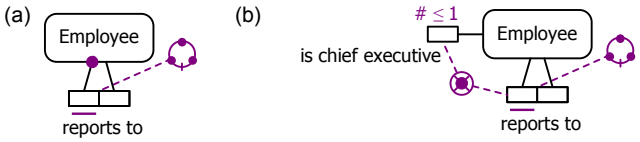

Fig. 8. (a) An incorrect ORM schema fragment, and (b) a correct ORM schema fragment

The ORM schema in Figure 8(a) includes an acyclic ring fact type. Formalization of all ring constraints in ORM is discussed in detail in [14], so is omitted here, other than noting that recursive constraints such as acyclicity and strong intransitivity are assigned fixed point semantics. The combination of mandatory role, uniqueness, and acyclicity constraints in Figure 8(a) says that each employee has exactly one employee to which he/she reports, and the reporting relation is acyclic. This is allowed in classical logic, where the Employee domain may be infinite. However, this constraint pattern is illegal in ORM because all its domain predicates (including the Employee type predicate) must be *finite*. The schema in Figure 8(b) fixes the problem by using an exclusive-or constraint to ensure that each employee is either a chief executive or reports to a manager but not both. A chief executive may now reside at the top of the reporting hierarchy without reporting to any employee.

The *role cardinality constraint* "# ≤ 1" ensures that there is at most one employee who is chief executive, i.e. $\exists^{0..1} x$ *x* is chief executive. This is one of the new kinds of constraint introduced in ORM 2, along with deontic constraints, value-comparison constraints and others. By interpreting a state of the business model as a possible world, the constraints discussed so far may be treated as alethic, implicitly prepending the alethic necessity operator □ of modal logic to the usual constraint formula, e.g. □ $\forall x$ (Scientist $x \rightarrow \exists^1 y$ *x* was born in *y*). *Deontic* constraints are obligations rather than necessities, and are formalized by prepending the deontic "it is obligatory that" operator ***O*** to the usual constraint formula. For example, ***O***$\forall x$ (Driver $x \rightarrow x$ is licensed to drive). For further discussion of deontic constraints, see [7]. There is no space here to formalize all the constraint varieties in ORM 2, but the constraint formalization patterns not discussed here are typically straightforward or are discussed elsewhere.

The formalizations provided here use unsorted logic, but may trivially be re-expressed using *sorted logic*, where the individual variables range over specific object types. For example, $\forall x$(Country $x \rightarrow$ Entity x) may be reformulated as $\forall x$:Country Entity *x*. Sorted logic formulae correspond more closely to FORML verbalizations, which use subscripted variables to distinguish variables of the same type (e.g. *x:*Person and *y:*Person may be rewritten as Person_1 and Person_2).

4 Derivation Options

ORM 2 fact types and object subtypes may be asserted, derived, or semiderived [15]. A semiderived type may have asserted instances and derived instances. Derivation rules for derived fact types and derived subtypes may be complete (**iff**-rules) or incomplete (**if**-rules). Complete derivation rules are formalized as universally quantified equivalences, e.g. $\forall x,y$[*x* is father of $y \equiv$ (*x* is a parent of *y* & *x* is male)]. Incomplete derivation rules and semiderived rules are formalized as universally quantified implications, e.g. $\forall x,y$[*x* is a grandparent of $y \leftarrow \exists z$(*x* is a parent of *z* & *z* is a parent of *y*)].

In ORM, equality and subset constraints are also formalized using universally quantified equivalences and implications respectively. For example, the conditional $\forall x[x$ is cancer prone $\leftarrow x$ smokes) could be used to formalize an ORM subset constraint or the derivation rule for an incomplete derived fact type or a semiderived fact type. Which of these three interpretations is given to the formula is an important implementation choice. For example, if we use a subset constraint then we cannot add the fact that Pat smokes without explicitly adding the fact that Pat is cancer prone. However, with a derived or semiderived fact type we can add the fact that Pat smokes, and leave it to the system to infer that Pat is cancer prone. In ORM, this implementation choice is reflected by explicitly classifying the declaration as a constraint or derivation rule, and this choice is depicted graphically in different ways on ORM diagrams. When mapping from ORM to Datalog^{LB}, we map constraints to "right-arrow rules" using the forward implication operator "->", and we map derivation rules to "left-arrow rules" using the converse implication operator "<-", since this is how Datalog^{LB} distinguishes syntactically between these choices [13].

There is no space here to give a full account of derivation in ORM, but we note in passing that the description logics and the Web Ontology Language (OWL) [24] currently have no way to choose to implement conditionals as integrity constraints [19], though some proposals have been made to extend OWL in this direction [21].

5 Conclusion

This paper extended previous work on formalization of ORM by replacing the former metalevel subtyping association between semantic value type and data type by a mapping relationship, axiomatizing a special representation relationship to support this mapping at the instance level, formalizing preferred reference schemes and various constraint and rule patterns, and briefly discussing the use of finite model theory, sorted logic, and choices for implementing certain kinds of logical formulae as constraints or derivation rules. Current and future research efforts in this regard are focused on providing a more rigorous and complete formalization and implementation of ORM derivation and query facilities, including negation, bag functions, and extended support for dynamic constraints and rules [2].

Acknowledgement. This paper's treatment of semantic values has benefited from discussion with colleagues, especially Matt Curland, Andy Carver and Lex Spoon.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading (1995)
2. Balsters, H., Halpin, T.: Formal Semantics of Dynamic Rules in ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 699–708. Springer, Heidelberg (2008)
3. Chen, P.P.: The entity-relationship model—towards a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976), http://csc.lsu.edu/news/erd.pdf
4. De Troyer, O.: On Data Schema Transformations. PhD Thesis, Tilburg University (1993) ISBN 90-9005913-X

5. Halpin, T.: A Logical Analysis of Information Systems: static aspects of the data-oriented perspective. Doctoral dissertation, University of Queensland (1989), http://www.orm.net/Halpin_PhDthesis.pdf
6. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)
7. Halpin, T.: Modality of Business Rules. In: Siau, K. (ed.) Research Issues in Systems Analysis and Design, Databases and Software Development, pp. 206–226. IGI Publishing, Hershey (2007)
8. Halpin, T.: Predicate Reference and Navigation in ORM. In: Meersman, R., Herrero, P., Dillon, T. (eds.) OTM 2009 Workshops. LNCS, vol. 5872, pp. 723–734. Springer, Heidelberg (2009)
9. Halpin, T.: Object-Role Modeling: Principles and Benefits. International Journal of Information Systems Modeling and Design 1(1), 32–54 (2010)
10. Halpin, T.: Structural Aspects of Data Modeling Languages. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) BPMDS 2011 and EMMSAD 2011. LNBIP, vol. 81, pp. 428–442. Springer, Heidelberg (2011)
11. Halpin, T.: Fact-Orientation and Conceptual Logic. In: Proc. 15th International EDOC Conference, pp. 14–19. IEEE Computer Society, Helsinki (2011)
12. Halpin, T., Curland, M.: Automated Verbalization for ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1181–1190. Springer, Heidelberg (2006)
13. Halpin, T., Curland, M., Stirewalt, K., Viswanath, N., McGill, M., Beck, S.: Mapping ORM to Datalog: An Overview. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM 2010. LNCS, vol. 6428, pp. 504–513. Springer, Heidelberg (2010)
14. Halpin, T., Curland, M.: Enriched Support for Ring Constraints. In: Meersman, R., Dillon, T., Herrero, P. (eds.) OTM-WS 2011. LNCS, vol. 7046, pp. 309–318. Springer, Heidelberg (2011)
15. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
16. Halpin, T., Wijbenga, J.P.: FORML 2. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R. (eds.) BPMDS 2010 and EMMSAD 2010. LNBIP, vol. 50, pp. 247–260. Springer, Heidelberg (2010)
17. ter Hofstede, A., Proper, H., van der Weide, T.: Formal definition of a conceptual language for the description and manipulation of information models. Information Systems 18(7), 489–523 (1993)
18. Keet, C.: Prospects for and issues with mapping the Object-Role Modeling language into DLRifd. In: 20th Int. Workshop on Description Logics (DL 2007). CEUR-WS, vol. 250, pp. 331–338 (2007)
19. Krötzsch, M., Simancik, F., Horrocks, I.: A Description Logic Primer, eprint arXiv:1201.4089 (2012), http://arxiv.org/abs/1201.4089
20. Libkin, L.: The Finite Model Theory Toolbox of a Database Theoretician. In: PODS 2009 (2009) ACM 978-1-60558-553-6/09/06
21. Motik, B., Horrocks, I., Sattler, U.: Bridging the Gap Between OWL and Relational Databases. J. of Web Semantics 7(2), 74–89 (2009)
22. Object Management Group: Unified Modeling Language Specification, version 2.4.1 (2011), http://www.omg.org/spec/UML/2.4.1/
23. Wintraecken, J.: The NIAM Information Analysis Method: Theory and Practice. Kluwer, Deventer (1990)
24. W3C: OWL 2 Web Ontology Language: Direc Semantics (2009), http://www.w3.org/TR/owl2-direct-semantics/