Structural Aspects of Data Modeling Languages

Terry Halpin

LogicBlox, Australia and INTI International University, Malaysia
e-mail: terry.halpin@logicblox.com

Abstract: A conceptual data model for an information system specifies the fact structures of interest as well as the constraints and derivation rules that apply to the business domain being modeled. The languages for specifying these models may be graphical or textual, and may be based upon approaches such as Entity Relationship modeling, class diagramming in the Unified Modeling Language, fact orientation (e.g. Object-Role Modeling), Semantic Web modeling (e.g. the Web Ontology Language), or deductive databases (e.g. datalog). Although sharing many aspects in common, these languages also differ in fundamental ways which impact not only how, but which, aspects of a business domain may be specified. This paper provides a logical analysis and critical comparison of how such modeling languages deal with three main structural aspects: the entity/value distinction; existential facts; and entity reference schemes. The analysis has practical implications for modeling within a specific language and for transforming between languages.

1 Introduction

A *conceptual data model* includes a conceptual schema (structure based on concepts that are intelligible to business users) as well as a population (set of instances that conform to the schema). A conceptual schema specifies the fact structures of interest as well as the business rules (constraints or derivation rules) that apply to the relevant business domain. Various languages are used by modelers to capture or query the data model. These languages may be graphical or textual.

In *attribute-based approaches* such as Entity Relationship modeling (*ER*) [2] and the class diagramming technique within the Unified Modeling Language (*UML*) [18]), facts may be instances of attributes (e.g. Person.isSmoker) or relationship/association types (e.g. Person drives Car). UML's Object Constraint Language (OCL) [19, 21] provides a textual means to express class diagrams as well as many additional rules.

In *fact-oriented modeling* approaches, such as Object-Role Modeling (*ORM*) [10], all facts are treated as instances of fact types, which are represented using typed, logical predicates (e.g. Person smokes, Person drives Car). Referential facts also involve existential quantification (e.g. some Country has CountryCode 'AU'). For a detailed coverage of ORM and comparisons with ER and UML see [13]. Overviews of fact-oriented modeling approaches, including history and research directions, may be found in [9, 11]. The Semantics of Business Vocabulary and Business Rules (SBVR) initiative [20] and the Object-Oriented Systems Modeling (OSM) approach [6] are also fact-based in their requirement for attribute-free constructs.

Declarative, logic-based languages are being increasingly used for data models that require rich support for logical derivation. The Web Ontology Language (*OWL*) [23], based on description logics, is designed to capture ontologies for the Semantic Web. Business intelligence tools and rule-based software are now widely used to perform predictive analytics over massive data sets and enforce complex business rules. This has led to a resurgence of interest in *datalog*, because of its powerful deductive database capability for processing complex rules, especially recursive rules [1].

Although sharing many aspects in common, these data modeling languages also differ in fundamental ways that impact not only how, but which, aspects of a business domain may be specified. This paper provides a logical analysis and critical comparison of how such modeling languages deal with three main structural aspects: the entity/value distinction; existential facts; and entity reference schemes. The analysis has practical implications for modeling within a specific language and for transforming between such modeling languages.

The rest of this paper is structured as follows. Section 2 discusses different ways in which modeling languages distinguish between entities and values, and the impact this has on modeling facts about them. Section 3 motivates the need for existential facts and the different ways (e.g. skolemization) in which these are supported (if at all) in the modeling languages. Section 4 briefly examines the relationship between skolemization and entity reference schemes. Section 5 summarizes the main contributions and outlines future research directions.

2 Entities and Values

In Chen's original ER model [2], an *entity* is defined as "a 'thing' which can be distinctly identified", a "relationship" is defined as "an association among entities", and information about entities or relationships is stored using attribute-value pairs in mathematical relations. For example, the *value* "AU" (an instance of the value set "CountryCode") may be used to represent the entity that is the country Australia. A typical, modern ER definition for the term "entity" is "a real-world object with an *independent existence*" (e.g. [3, p. 373], [5, p. 43]). Here an object may be physical (e.g. a person) or abstract (e.g. a course). Nowadays in ER modeling, the term "value" typically means a data value (instance of a value type based on a given datatype), such as a person's family name or a course code. One or more attributes or relationships of an entity are chosen to provide its primary identifier, which identifies the entity by mapping it (directly or indirectly) to its referencing value(s). In this paper, we use the term "entity" to mean an entity instance, not an entity type.

The above definitions for "entity" have issues. As a trivial issue, the world being modeled (the business domain of interest) need not be "real" in the normal sense of the word (e.g. consider a data model about fictional characters in the Harry Potter novels). As a substantive issue, the requirement of independent existence is debatable. In what sense does an entity such as a country exist independently? This notion is difficult to capture conceptually with any rigor. The motivation for the independent existence requirement seems to be to distinguish entities (e.g. countries) from attribute values (e.g. country codes), since attributes are always attributes of something and in

that sense are not independent. In the ER approach, only entities and relationships may have attributes or participate in other relationships. In practice, this seems to assume that we may record facts about entities and relationships, but not about values.

In UML, the terms “object”, “class”, and “data value” roughly correspond to “entity”, “entity type”, and “value” in ER. UML objects are assigned internal, object identifiers, but data values are not. There is no requirement in UML to provide value-based identifiers that are visible to human users. For practical data modeling however, value-based identification is typically needed to ensure that users are able to communicate about the entities of interest within a model, and to know when the same entity is referenced in different models. Hence UML data modelers typically use tools that extend UML with features such as key attributes, or they write the OCL needed to ensure value-based identification. Unlike ER entities, UML objects may include operational properties, but these operations are irrelevant to our discussion.

Consider a model in which each employee is identified by an employee number and also has exactly one honorific title (e.g. “Dr”, “Mr” or “Ms”). A Barker ER diagram is shown for this in Fig. 1(a). Here Employee is modeled as an entity type with employee number and honorific as attributes. Employee numbers are used by humans in the business domain as the preferred identifiers for employees, so they are not hidden object identifiers in the UML sense, even if auto-generated. Fig. 1(c) shows a UML class diagram for this situation. The UML diagram omits the constraint that employee numbers are identifying, but this does not concern us here. If there is no need to talk about honorifics themselves, we believe that most ER and UML modelers would naturally model honorific as an attribute rather than as an entity type or class. An honorific instance (e.g. “Dr”) is then conceived of as a value, not an entity.

In the real world, some honorifics have expansions (e.g. “Dr” expands to “Doctor” but “Ms” has no expansion), but assume initially that this is not of interest to the business domain. Now suppose that the business later discovers a need to display honorific expansions. In ER and UML, attributes can’t have attributes or participate in relationships, so the honorific attribute needs to be remodeled as an entity type or class, and a binary relationship/association must now be used to model the former facts about employee honorifics, as shown in Fig. 1(b) and Fig. 1(d). Moreover, any code or user interface that accessed the former honorific attribute needs to be modified. An honorific instance is now conceived of as an entity (or UML object), not a value.

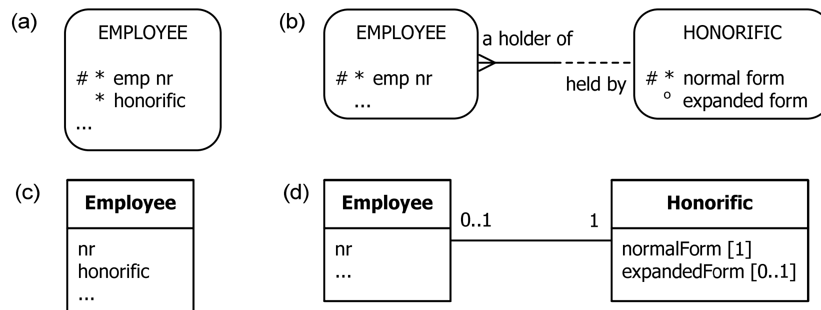

Fig. 1. Modeling employee honorifics in Barker ER (a), (b) and UML (c), (d)

Apart from the inconvenience of having to remodel already existing facts, changes of this nature have philosophical implications. If it was correct to initially treat an honorific instance as a value and also correct to finally model it as an entity, then it seems possible for a value to change to an entity. Moreover, in this case the change in an honorific's nature seems to be caused by the mere act of recording a fact about it (e.g. the fact that the Honorific "Dr" is short for the HonorificExpansion "Doctor"). This is somewhat reminiscent of Heisenberg's Uncertainty Principle, where the mere act of observing something necessarily changes it. However, *it seems implausible that a thing can change its nature (e.g. a value becomes an entity) simply because we want to talk about it*. Is there some way of drawing the entity/value distinction that does not force us into such seeming absurdities?

One extreme response might be to adopt a dynamic, relativistic theory where a thing is a value or entity only relative to a state of the business domain. So within a business domain, something is an entity at time *t* just in case the business wishes at time *t* to record facts about it (other than facts using it to reference another entity). However, this approach still has the semantic instability problem just described, and would seem to add considerable complexity to any underlying formalization.

An alternative solution that simply avoids such problems is provided by fact-oriented approaches such as ORM. In ORM a value may be defined as a *self-identifying constant* of a specified finite type, where the type name is typically informative (e.g. Honorific) or simply indicative of a conceptual datatype (e.g. CharacterString). So values can be verbalized by definite descriptions that simply include the lexical constant and a value type name (e.g. "the Honorific 'Dr'").

In contrast, an entity in ORM requires a *reference scheme* that includes at least one *referential relationship*. For example, the definite description "The Employee who has the employee number 2011" involves a specific binary relationship between the employee and the number. Moreover, entities typically change their state over time, so are not usually constant (unlike values). Hence in ORM it is impossible for a value to change to an entity.

ORM is *attribute-free*, so all facts are represented by relationships over one or more objects. In ORM, an object is the same as an *individual* in classical logic, so it can be an entity or value. Hence, in ORM *entities or values may appear in any position in a relationship*.

Fig. 2(a) shows an ORM schema and sample population for the ER example in Fig. 1(a). Entity types appear as named, solid, rounded rectangles, and value types appear as named, dashed, rounded rectangles. Relationship types are depicted using logical predicates which display as named, ordered sets of role boxes connected to the object types whose instances play those roles. An asserted fact type is either elementary or existential. A non-existential fact type is a set of one or more typed predicates, which may be unary, binary, or of higher arity. An elementary fact can't be rephrased as a conjunction of smaller facts with the same objects without information loss.

An injective relationship from an entity type to a value type that is used for entity identification is called a refmode predicate, and may be displayed in abbreviated form by enclosing the refmode in parenthesis below the entity type name. The bar over the first role of the Employee has Honorific predicate is a uniqueness constraint (each employee has at most one honorific). The solid dot on the role connector is a mandatory role constraint (each employee has some honorific).

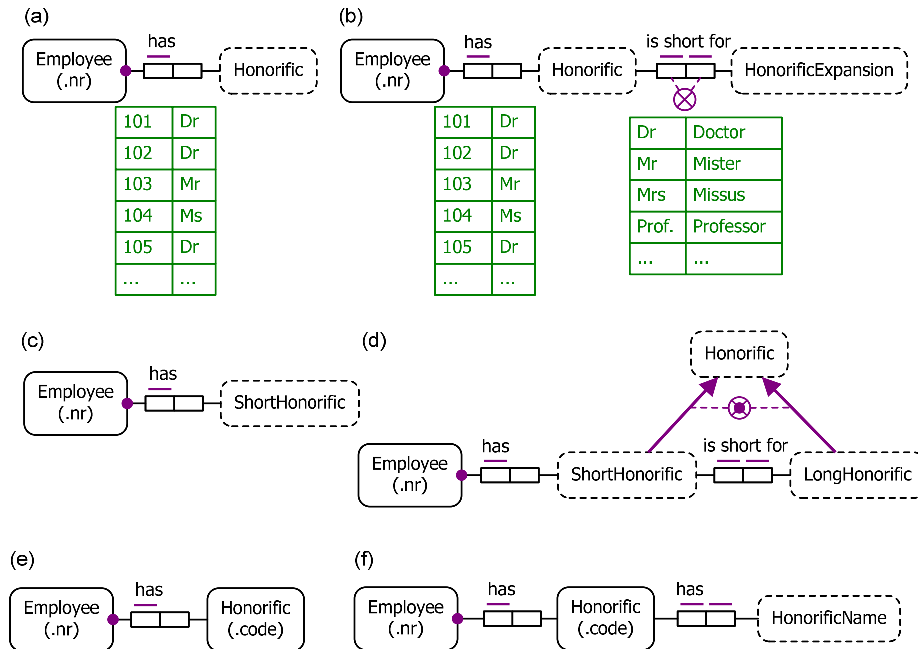

Fig. 2. Modeling employee honorifics in ORM

Fig. 2(b) adds the optional, 1:1 relationship type Honorific is short for HonorificExpansion. Notice that this addition has no impact on the original model in Fig. 2(a). This a simple illustration of the greater *semantic stability* enabled by fact-orientation in comparison with attribute-based approaches. Facts may be added about any kind of object (entity or value) without impacting the existing model. The circled cross denotes an exclusion constraint (no honorific is an honorific expansion).

The models in Fig. 2(a) and Fig. 2(b) use "honorific" in a restricted sense to mean the usual short title applied to a person's name (e.g. "Dr"). If "honorific" is used in the business domain to include longer titles (e.g. "Doctor"), then the type names should be adjusted accordingly (e.g. "ShortHonorific" and "LongHonorific"). Fig. 2(c) would then be used as the initial schema, and the lower part of Fig. 2(d) could be used as the expanded schema. If the business wishes to talk about honorifics in general, then the supertype Honorific may be introduced as in Fig. 2(d). The circled, dotted cross between the subtyping connections denotes an exclusive-or constraint (Honorific is partitioned into ShortHonorific and LongHonorific).

The ORM models in Figures 2(a)-(d) conceive of honorifics as simple labels (and hence values). However, suppose the modeler feels that "Dr" and "Doctor" are just different representations of the same honorific. With this understanding, an honorific is an entity (e.g. a personal status concept), not a value. Fig. 2(e) and Fig. 2(f) show one way to model this in ORM. Honorific is now an entity type. The short label for an honorific is called an honorific code, and the longer label is called an honorific name.

In practice, different people sometimes assign different meanings to the same term. Hence whether a "thing" is conceived of as an entity or value is sometimes relative to

the user. The honorifics example is a borderline case, where it is not unreasonable to take either view with respect to the entity/value status of honorifics. However, once the meaning of a term within the business domain is determined, it should be modeled consistently, otherwise the benefits of semantic stability are lost. With this understanding, *within a given model, no value may change to an entity, or vice versa*.

In most cases, an entity can be intuitively distinguished from a value simply by noting that it is not lexical in nature, or that it can be referenced by labels in more than one way, or that it can change over time. For example, a country can be referenced by its relationship of having a country code (e.g. 'AU') or a country name (e.g. 'Australia'), and it evolves over time. One often sees ER or UML models where gender is modeled as an attribute; but a gender is an abstract concept that can be referenced by its relationship to a gender code (e.g. 'M') or a gender name (e.g. 'Male'). Even though a gender does not change over time, it is clearly not a label like a gender code. By modeling gender and honorific details for persons as relationships between Person and Gender and Honorific object types, we can add the optional, functional relationship type Honorific is restricted to Gender, populate it with data (e.g. the honorific 'Mr' is restricted to the gender with code 'M'), and add the join subset constraint **If a** Person has **an** Honorific **that** is restricted to **some** Gender **then that** Person is of **that** Gender [14].

In practice, commercial versions of ER are typically restricted to single-valued attributes. This restriction, combined with the inability to model facts about attribute values, leads to further semantic instability. For example, suppose we began with the ER model of Fig. 1(a), and later needed to cater for the possibility of recording multiple honorifics for the same employee (e.g. 'Prof.' and 'Dr'). In Barker ER, we would need to replace the honorific attribute with an *m:n* relationship from Employee to an Honorific entity type. In ORM, we simply change the uniqueness constraint on the employee-honorific relationship to be spanning. UML can cater for this change by using a multivalued honorific attribute (change its multiplicity in Fig. 1(c) to [*]), but in practice multivalued attributes can be more trouble than they are worth ([13], p. 356).

We now discuss entities and values in logic-based languages, starting with OWL 2, the latest version of the Web Ontology Language. An overview of OWL's structure is shown in Fig. 3, which is adapted from the official specification [26].

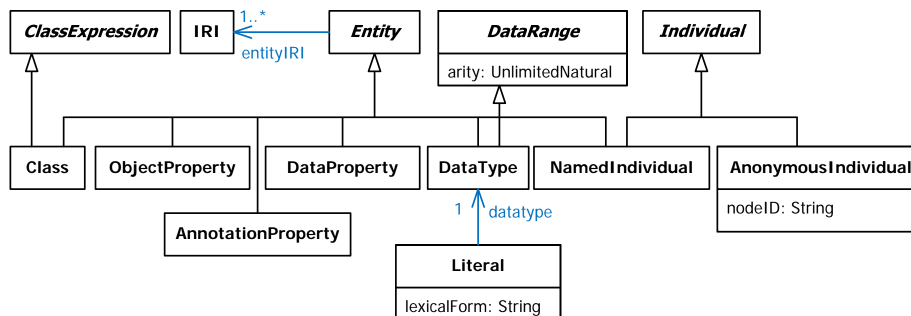

Fig. 3. UML class diagram of basic OWL concepts, adapted from [26]

OWL identifies entities by Internationalized Resource Identifiers (IRIs) [4], but unlike some approaches, OWL does not adopt the Unique Name Assumption, so the same entity may be assigned different IRIs, even within the same document. Hence the multiplicity constraint on entityIRI is 1..* (1 or more), not 1 as specified in [26].

As can be seen from Fig. 3, OWL uses the term "entity" in a much broader sense than we have been considering. For example, a class is itself an entity, and in OWL Full a class can even be an instance of itself, inviting Russell's paradox.

OWL *properties* are binary predicates, and their instantiations are treated as entities, similar to instances of class associations in UML and, to some extent, objectification in ORM. OWL *individuals* are either named or anonymous. Anonymous individuals are discussed in the next section. *Named individuals* are typical of the entities we discussed earlier, except that they are identified by an IRI. *Literals* roughly correspond to what we have been calling values. A literal has a lexical form (quoted string, for which a language tag may optionally be specified) and a datatype, which may be hidden if it is rdf:PlainLiteral (see pp. 37-39 of [26] for details).

Note that *OWL literals are not treated as individuals*, and so OWL differs from classical logic in this respect, where, for example, you can use the individual constant "AU" to refer to the individual character string inside the quotes. *Object properties* are binary predicates that relate individuals to individuals. *Data properties* are binary predicates that relate individuals to literals. For example, if within the local document "Einstein" and "Germany" serve as IRIs, then we can declare Einstein's birth country and name in Manchester Syntax [25] thus:

```
ObjectProperty: wasBornIn
DataProperty: hasName
Individual: Einstein
   Facts: wasBornIn Germany, hasName "Albert Einstein"^^xsd:string
```

OWL's distinction between entities and literals seems to be taken very loosely in practice. For example, the official OWL 2 Primer cites as examples of data values "a person's birth date, his age, his email address etc." [23, p. 21], giving the following example of a data property stating that John's age is 51:

```
Individual: John
   Facts: hasAge "51"^^xsd:integer
```

In conceptual data modeling, an age is a duration in time with a unit (e.g. years), so an age is an entity, not a data value. Similarly, a date is a 24 hour period (anchored duration in time), so is an entity, unlike a date string, which is a value. An e-mail address may be conceived as a value, though a home address could be thought of as either a physical location (an entity) or a value (possibly structured).

OWL is built on top of the Resource Description Framework (RDF), so OWL facts are expressed as subject-predicate-object triples, and *the subjects of OWL facts must be individuals, not literals*. So OWL is unable to model fact types of the form *A R B*, where *A* is a value type, such as ShortHonorific is short for LongHonorific in Fig. 2(b), or the ORM synonym fact types in Fig. 4. Here, "Word" means English word, and its instances are represented by character strings, just as they would typically be stored in a relational database. Of course, we could model words in OWL by treating them as entities, but it seems subconceptual to require an IRI in order to talk about a word.

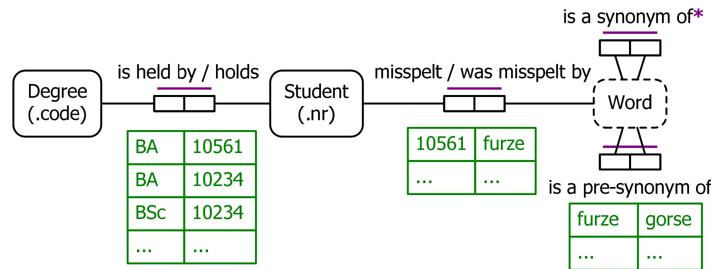

Fig. 4. An ORM model about students and word knowledge

For binary fact types in ORM, a slash may be used to separate forward and inverse predicate readings. In Fig. 4, the student degree fact type has two readings: Degree is held by Student; Student holds Degree. The fact type used to record student misspellings also has two readings: Student misspelt Word; Word was misspelt by Student. As well as supporting natural communication by allowing facts to be expressed in different ways, inverse readings often facilitate more natural verbalization of rules that involve navigation over paths that traverse multiple fact types. Barker ER supports forward and inverse relationship readings. UML supports only one association reading per association, but allows navigation in different directions across an association by use of role names. OWL supports inverses of object properties. For example, both predicates for the student-degree fact type in Fig. 4 may be declared in Manchester Syntax thus:

```
ObjectProperty: isHeldByStudent
  InverseOf: holdsDegree
```

However, while the student-misspelling fact type may be declared as a data property using the "misspelt" predicate, its inverse predicate "wasMisspeltBy" cannot be declared at all because its subject is a literal type. The only way around these problems in OWL is to remodel Word as an entity type. Even if it is reasonable to conceive of a word as an entity not a value, there are many cases where such a workaround seems unnatural. Most modelers consider names to be values, not entities, and the earlier example of using a data property to record Einstein's name is typical in OWL. However, if we do this, we cannot express the inverse relationship that would be modeled in ORM using PersonName is of Person.

Suppose we initially model names or codes etc. as values, but then wish to talk about them (e.g. record their origin, meaning, purpose, or length). Do they now suddenly become entities? We think not. Clearly there is a difference between a country and a country name or country code. You can live in a country, but you can't live in a country code. However, there are different stances one might take with respect to the nature of values themselves, as used in conceptual modeling.

Consider the country code "us" and the pronoun "us". Are these identical values? If values are simply untyped, lexical constants, then the answer is Yes: it's the same value being used for two different purposes. The value types CountryCode and Pronoun are then understood (implicitly or explicitly) to be finite, overlapping subtypes of a datatype such as CharacterString. However, suppose we populate the fact type Pronoun is plural with "us". If the pronoun "us" = the country code "us" then the principle of substitutivity of identicals entails that the country code "us" is plural, which is nonsense.

The only way to make sense of this untyped constant approach is to implicitly expand all predicate readings to include types (e.g. rewrite "is plural" as "is a plural pronoun" or "pronoun:isPlural").

If instead we consider the notion of a value to intrinsically include a specific purpose (represented by a type name), then the answer is No: pronouns and country codes are different kinds of things. This second position treats values as strongly typed constants, and a distinction may be drawn between strong identity and lexical equality when comparing values. So the pronoun "us" and the country code "us" are strictly non-identical while still being lexically equal. With this approach, value types may have a datatype but are not subtypes of datatypes. It is still possible to have overlapping value types (e.g. CountryCode overlaps with TwoLetterCode).

We now briefly discuss how entities and values are treated in *datalog*, a logic-based language that is becoming widely used in business analytics and rule-based systems. Like Prolog, datalog is a logic programming language. Unlike Prolog, it is purely declarative and has guaranteed decidability. As datalog is based on classical logic, it allows predicates to be applied to individuals that may be entities or values, with no restriction on where they may appear in predications. Datalog supports powerful inferencing capabilities, allowing complex rules (including recursive rules) to be elegantly formulated and efficiently processed. For illustration, we use Datalog^{LB}, an extended form of typed datalog. A theoretical discussion of datalog may be found in [1], and an overview of mapping ORM to Datalog^{LB} may be found in [12].

Let's see how the misspelling and synonymy fact types in Fig. 4 might be declared and populated in Datalog^{LB}. Derived fact types in ORM are indicated with an asterisk. The synonymy fact type may be derived using the following FORML [14] rule: Word$_1$ is a synonym of Word$_2$ **iff** Word$_1$ is a pre-synonym of Word$_2$ **or** Word$_2$ is a pre-synonym of Word$_1$.

ORM entity types and value types help understand and visualize models, especially how things are connected. When mapping an ORM model to a model in another language, you may leave value types implicit (e.g. by informally including the semantics in a predicate reading such as "hasCountryCode"). In the following Datalog^{LB} program, the Word type is implicitly treated as a subtype of string. The declarations constrain the types of the predicate arguments (read "->" as "implies", and "," as "and"). The derivation rules derive the inverse misspelling predicate and the synonymy predicate (read "<-" as "if" and ";" as "or"). Some data and a sample query are included. If you wish to make Word an explicit value type, declare its type as string, replace string by Word in the other declarations, and add +Word("furze"), +Word("gorse") to the data.

```
//declarations
Student(s), hasStudentNr(s:n) -> uint[32](n).
misspelt(s, w) -> Student(s), string(w).
isPreSynonymOf(w1,w2) -> string(w1), string(w2).
//rules
wasMisspeltBy(w,s) <- misspelt(s, w).
isSynonymOf(w1,w2) <- isPreSynonymOf(w1,w2);  isPreSynonymOf(w2,w1).
//data
+Student(10561), +misspelt(10561, "furze" ), +isPreSynonymOf("furze","gorse").

sample query:        isSynonymOf   ⇨  furze, gorse
                                      gorse, furze
```

3 Existential Facts

Conceptually, a fact base (as distinct from constraints or rules) may be expressed as a set of elementary or existential facts. An elementary fact is an atomic predication over named individuals (e.g. Einstein is male, Einstein was born in Germany). An *existential fact* asserts the existence an individual, typically to predicate over it (e.g. some person is male, some person was born in Germany). To facilitate a first-order formalization, we do not treat existence as a predicate. Most but not all logicians agree that if existence is treated as a predicate, it must be construed as a second-order predicate. For further discussion on "exists" as a predicate, see [17].

In typical relational database applications, simple existential facts like the examples above are never stored, even though they may be implied. If we store the fact that the politician Obama is the president of the USA, we can infer that some politician is the president of the USA. But knowing that some politician is the president of the USA doesn't enable us to infer who that is. On the surface then, it may appear that there is little reason for data models to even be concerned with existential facts.

However, there are cases where support for existential facts is vital. One case is *data exchange* between different schemas that are not logically equivalent, even when supplemented by conservative extension derivation rules (for a formalization of ORM schema equivalence under conservative extension see [8]). Rules that map data between the models may be set out as tuple-generating dependencies of the form $\forall \boldsymbol{x},\boldsymbol{y}[\Phi(\boldsymbol{x},\boldsymbol{y}) \rightarrow \exists z\Psi(\boldsymbol{x},\boldsymbol{z})]$, where $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ are variable lists, and $\Phi(\boldsymbol{x},\boldsymbol{y})$ and $\Psi(\boldsymbol{x},\boldsymbol{z})$ are conjunction of atoms from the source and target schema respectively (e.g., see [7], [16]). For example, suppose both the source and target model information about scientists, but only the second records their birth countries, and has a constraint that each scientist has a birth country, i.e. $\forall x$[Scientist(x) $\rightarrow \exists y$ wasBornInCountry(x,y)]. To map details about Einstein from the first to the second model, a *skolem constant* is introduced there to denote Einstein's birth country. Queries that include the birth country will now return the null set, but queries that project only on non-skolem attributes work fine.

A related application of existential facts is support for *updating views* that involve joins. Suppose the database includes the base relation scheme parentOf(parent, child) as well as the view grandparentOf(grandparent, grandchild) derived from the rule $\forall x,y$ [grandparentOf(x,y) $\leftarrow \exists z$(parentOf(x,z) & parentOf(z,y)]. In a normal relational database, if we attempt to insert the fact grandparentOf(Bernie, Selena) into the view, this will be rejected, since the update can't be translated into updates on the base parentOf relation. Adding the tuples parentOf(Bernie, null) and parentOf(null, Selena) won't help, even if allowed, because nulls never match (comparisons with null return unknown). For a discussion of this example in SQL see [13, p. 649].

However, if instead we use a logic-based database that supports skolem terms, we can accept the view update simply by using the same skolem term for the intermediate unknown parent. From an ORM perspective, the grandparenthood fact type is now *semi-derived*, since some of its instances may be simply asserted and other instances may be derived from parenthood facts.

Both OWL and Datalog^{LB} support existential facts, though there are some differences in their approaches. In OWL, individuals that are referenced by skolem con-

stants are called *anonymous individuals*, and correspond to blank nodes in RDF (see section 2.3 of [22]). A skolem constant itself (e.g. _:a or _:b) is called a nodeId (see Fig. 3), and may be read as "something" if it's the only skolem term in the statement; otherwise the reading should include the id name (e.g. "some a" or "some b"). In OWL, a skolem constant is simply an arbitrary constant that replaces an existential quantification within the scope of the current statement.

In an effort to support the AAA assumption (Anybody can say Anything about Anything), OWL places few restrictions on use of anonymous individuals. For example, you can simply assert that some god exists, and that some woman is the prime minister of Australia (without knowing that it's Julia Gillard). The following OWL statements in Turtle syntax do this using local nodeIds for anonymous individuals. Although these are legal in OWL, some OWL tools (e.g. Protégé) do not support use of nodeIds in this way. For a detailed overview of OWL syntaxes, see [23].

```
_:x  rdf:type  :God.
_:y  rdf:type :Woman ; :isPrimeMinisterOf :Australia.
```

Even within the OWL community, there are some who see little use for asserted existential facts, except for cases where blank nodes simply serve the purpose of joining facts, as in the RDF graphs shown in Fig. 5. By introducing the blank node "_:c" for "some city" in Fig. 5(a), we can assert that Einstein was born in a city that has a population of 121650, without knowing which city it is. We delay discussion of Fig. 5(b) till the next section, as it bears on the topic of reference schemes. For a somewhat humorous debate on the worth or otherwise of skolem terms, see the "OWL 2 Far" panel discussion segment between Stefan Decker and Ian Horrocks [15].

In classical datalog, existential facts are allowed only in the body of a rule, and a *rule* is an expression of the following form, where the *head* predicate q has as argument an ordered list of individual terms $\tau_1, \ldots \tau_n$ ($n \geq 0$), each variable of which must occur in at least one argument of the *body* predicates $p_1 \ldots p_m$ ($m \geq 0$).

$$q(\tau_1, \ldots, \tau_n) \leftarrow p_1(x_1, \ldots), \ldots, p_m(y_1, \ldots).$$

In classical datalog, a rule is treated as shorthand for a formula where the head variables are universally quantified at the top level, and any other variables introduced in the body are existentially quantified, with the existential quantifiers placed at the start of the body [1, p. 279]. For example, the datalog rule grandparentOf(x, y) $\leftarrow$ parentOf(x, z), parentOf(z, y) is interpreted as shorthand for the following predicate logic formula: $\forall x \forall y$[grandparentOf(x, y) $\leftarrow$ $\exists z$(parentOf(x, z) & parentOf(z, y))].

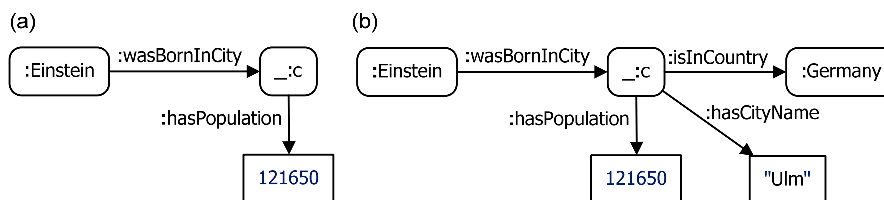

Fig. 5. RDF graphs using a blank node to assert the existence of some city

If an existentially quantified variable appears more than once in the body, a named, individual variable is used (e.g. "*z*" in the above example). If it appears only once, an anonymous variable "_" may be used, since we don't need to refer to it elsewhere. For example, consider the rule Parent(p) ← Person(p), parentOf(p,_).

What OWL calls blank nodes are thus handled in datalog using named or anonymous variables, where the variables are understood to be existentially quantified. If you mentally ignore the implicit existential quantifiers, you may think of these variables as skolem constants, except that multiple anonymous variables in the same rule must be treated as distinct (i.e. they may or may not be equal). For example, consider the datalog rule CarDrivingParent(p) ← Person(p), parentOf(p,_), drivesCar(p,_). In OWL, we could distinguish the anonymous variables by using different nodeIds (e.g. _:x, _:y).

While classical datalog allows existential facts only in rule bodies, there are good reasons to allow them in rule heads, such as supporting the data exchange and view updatability cases mentioned earlier. Support for head existentials in just one area in which Datalog^{LB} extends classical datalog, and in the next section we discuss an application of this feature that is conceptually linked to the notion of reference schemes.

4 Reference Schemes and Head Existentials

One common use of blank nodes in OWL is to model entities that have a composite reference scheme. For example, if cities can be identified just by their name and country then the RDF graph in Fig. 5(b) could be used to model the object property isInCountry and the data property hasCityname, which could then be constrained as mandatory, functional, key properties for City to provide a natural, value-based identification scheme. In practice, city names might not be unique to their country, but they are usually unique to their state.

Fig. 6 shows an ORM model in which states are identified by combining their state code and country. A uniqueness constraint shown with a double-bar indicates its use for a preferred reference scheme. Countries are identified simply by their country code (this reference scheme is shown in expanded form for discussion purposes). Finally, head politicians (e.g. presidents or prime ministers) are identified by the country that they head. With this schema we could record that fact that the government head of Australia was born in Wales, even if we don't know who he/she is (currently, it's Julia Gillard).

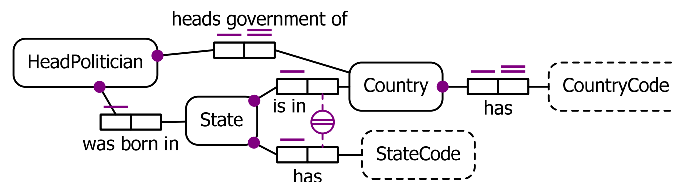

Fig. 6. An ORM model with some simple and compound reference schemes

In OWL, a head politician would be modeled as a blank node, and so would a state unless we have a natural IRI for it. For most states, a name based IRI could be used if known (e.g. :WashingtonState and :WestAustralia both have state code "WA"), but some states do have the same name, so this doesn't always work. Country names are identifying, so countries would typically be identified by an IRI (e.g. :Australia). However, suppose that we want to talk about a country with country code "AU", but don't know its name. It would be strange to use an IRI such as :AU, so unless we are able to base an IRI on some Website fragment dealing with countries, we could then choose a blank node for countries as well. In that situation, a population of the model in Fig. 6 would include values for country codes and state codes, but all the other entities (in the normal sense of the word) would effectively be existentially asserted using reference predicates to provide definite descriptions that relate them to these values.

IRIs are essentially scoped, individual constants that are identifying within their namespace. If we don't have an IRI for an entity, in order to talk about it we must provide a definite description for it, and this always involves at least one reference predicate. The most general form of reference scheme is disjunctive reference, where each instance of an entity type is ultimately 1:1 mapped onto one or more values via reference predicates [13, pp. 187-188].

Fig. 7 shows a much simplified fragment of an ORM model to automatically generate verbalizations of ORM constraints. For example, the uniqueness constraint on the modality fact type has a negative verbalization that renders as "It is impossible that some Constraint has more than one Modality". The components of the verbalization (only the modal text part shown here) can all be derived from properties of the constraint.

In Datalog^{LB}, once the shaded predicate is declared as a skolem predicate and the verbalization's storage structure is declared as ScalableSparse, the fact type for the modal text can be derived using rules that existentially quantify the verbalization in the rule head, e.g. NegativeVerbalization(v), hasNegativeVerbalization[c]=v, hasModalText[v]="It is impossible that " <- hasModality[c]= "Alethic". This has the form $\forall c\ (\exists v \Phi vc \leftarrow \Psi c)$.

Currently, to generate the datalog code from ORM, the constraint verbalization entity type must be assigned an autogenerated id, which is used as a type specific skolem constant. Conceptually, the situation may be viewed as analogous to the head of government reference scheme in Fig. 6, where a definite description such as "the negative verbalization of the constraint with constraint number *n*" suffices. The autogenerated verbalization id may then be viewed as an implementation issue rather than as part of the pure conceptual model, allowing the conceptually preferred reference scheme to then be indicated by using a double-bar for the uniqueness constraint on Constraint's role in the skolem predicate.

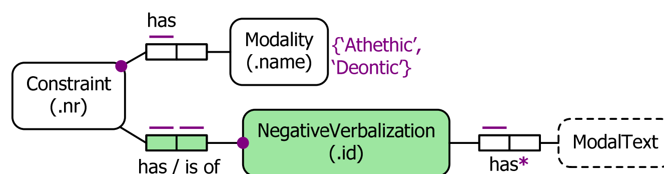

Fig. 7. A simplified ORM schema fragment involving skolemization

Even simple reference schemes such as the refmode predicate used to identify countries in Fig. 6 may be viewed as involving existential facts. As formalized in [8], the fact entry +Country("AU") asserts that there exists some country that has the country code "AU". This existential fact, when combined with the injective nature of the refmode predicate, licenses use of definite descriptions such as "the country that has country code "AU" for identifying entities. Viewed in this light, all data modeling approaches make use of existential facts, even though the approaches differ in the range of such facts that can be expressed and where they may appear in rules.

5 Conclusion

Although terms like "entity" and 'value" are often used in the data modeling community, they may have different meanings in different modeling approaches. This paper reviewed these notions within different modeling languages, and opted for a semantically stable approach that draws the entity/value distinction on fundamental representational grounds rather than subjective and possibly changing viewpoints on what features one wishes to record facts about. Although the semantic instability of attribute-based approaches like ER and UML is well known, in this paper we showed that this semantic instability problem relates more fundamentally to an unwillingness to allow values to be subjects of facts. Hence, OWL also suffers from this instability.

The paper also provided a motivation for existential fact support, discussed some different ways in which this is provided in logic-based languages, and examined some connections between skolemization and reference schemes. Although languages like OWL and Datalog^{LB} provide basic support for these features, more work needs to be done to provide a comprehensive and purely conceptual approach that can be mapped to such languages for execution. Understanding the different ways in which modeling approaches deal with entity/value distinctions and existential facts is important not only for modeling within a given approach but for transforming between approaches.

Owing to space considerations, the coverage of values focused mainly on string-based representations, but even within this limited scope there is room for further analysis. For example, a simple definition of a lexical value is "something that you can write down", but you can never write down a character string, only an occurrence of a representation of one. A full analysis of the entity/value distinction needs to embrace other kinds of data values (e.g. numeric and temporal), and properly account for unit-based reference. Different positions can also be taken on whether values can have conceptual structure. For example, is a person name composed of a given name and family name an entity or a "structured value"? As ongoing research not discussed here we are also refining the conceptual presentation of disjunctive reference schemes involving a partition of 1:1 predicates, as well as related subtyping alternatives.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading MA (1995)
2. 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.

3. Connolly, T., Begg, C.: Database Systems, 5th edn. Pearson Education, Boston (2010)
4. Duerst, M., Suignard, M.: RFC 3987: Internationalized Resource Identifiers (IRIs). IETF (January 2005), http://www.ietf.org/rfc/rfc3987.txt.
5. Elmasri, R., Navathe, S.: Fundamentals of Database Systems, 2nd edn. Addison-Wesley, Menlo Park (1994)
6. Embley, D.: Object Database Management. Addison-Wesley, Reading (1998)
7. Green, T., Karvounarakis, G., Ives, Z., Tannen, V.: Update Exchange with Mappings and Provenance. Technical Report MS-CIS-07-26, Dept. of Computer and Information Science, University of Pennsylvania (2007)
8. 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.
9. Halpin, T.: Fact-Oriented Modeling: Past, Present and Future. In: Krogstie, J., Opdahl, A., Brinkkemper, S. (eds.) Conceptual Modelling in Information Systems Engineering, pp. 19-38. Springer, Berlin. (2007)
10. Halpin, T.: Object-Role Modeling. In: Liu, L., Tamer Ozsu, M. (eds.) Encyclopedia of Database Systems. Springer, Berlin (2009)
11. Halpin, T.: Object-Role Modeling: Principles and Benefits. International Journal of Information Systems Modeling and Design 1(1), 32-54 (2010)
12. 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 Workshops. LNCS, vol. 6428, pp. 504-513, Springer. Heidelberg (2010)
13. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
14. Halpin, T., Wijbenga, J.: FORML 2. In: Bider, I. et al. (eds.) Enterprise, Business-Process and Information Systems Modeling, LNBIP, vol. 50, pp. 247–260. Springer, Berlin (2010)
15. ISWC 2008 Panel discussion: An OWL 2 Far?, http://videolectures.net/iswc08_panel_schneider_owl/.
16. Kolaitis, P.: Schema Mappings, Data Exchange, and Metadata Management. PODS 2005, Baltimore, Maryland, ACM 1-59593-062-0/05/06 (2005)
17. Miller, B.: Existence. Stanford Encylopedia of Philosophy, http://plato.stanford.edu/entries/existence/. 2002
18. Object Management Group: UML 2.0 Superstructure Specification, http://www.omg.org/uml (2003)
19. Object Management Group: UML OCL 2.0 Specification, http://www.omg.org/docs/ptc/05-06-06.pdf (2005)
20. Object Management Group: Semantics of Business Vocabulary and Business Rules (SBVR), http://www.omg.org/spec/SBVR/1.0/ (2008)
21. Warmer, J., Kleppe, A.: The Object Constraint Language, 2nd edn. Addison-Wesley, Reading (2003)
22. W3C: RDF Primer, http://www.w3.org/TR/rdf-primer/ (2004)
23. W3C: OWL 2 Web Ontology Language: Primer, http://www.w3.org/TR/owl2-primer/ (2009)
24. W3C: OWL 2 Web Ontology Language: Direct Semantics, http://www.w3.org/TR/owl2-direct-semantics/ (2009)
25. W3C: OWL 2 Web Ontology Language Manchester Syntax, http://www.w3.org/TR/owl2-manchester-syntax/ (2009)
26. W3C: OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, http://www.w3.org/TR/owl2-syntax/ (2009)