Entity Relationship modeling from an ORM perspective: Part 2

Terry Halpin
Microsoft Corporation

Introduction

This article is the second in a series of articles dealing with Entity Relationship (ER) modeling from the perspective of Object Role Modeling (ORM). Part 1 provided a brief overview of the ER approach, and then covered the basics of the Barker ER notation [1], that has long been supported in CASE tools from vendors such as Oracle Corporation. In this notation, entity types are depicted as named, soft rectangles, and binary relationships are shown as lines with forward and inverse names. If shown, attributes are listed below the entity type name. A "#" indicates that an attribute is [part of] the primary identifier of the entity type, a "*"indicates the attribute is mandatory and a "°" indicates the attribute is optional. Only binary relationships are allowed, so a role corresponds to a half-line (half a relationship line). If a role is mandatory, its half-line is solid. If a role is optional, its half-line is dashed. For role cardinality, a crows-foot at the end indicates "many" and its absence indicates "1". A bar "|" across one end of a relationship indicates that the relationship is a component of the primary identifier for the entity type at that end. Part 1 discussed examples of the above notations and compared them with the corresponding ORM notations. This second article briefly discusses verbalization, then examines the Barker ER notation for exclusion constraints, frequency constraints, subtyping and non-transferable relationships.

Barker ER: verbalization

To enable the optionality and cardinality settings to be verbalized, Barker [1, p. 3-5] recommends the following *naming discipline for relationships*. Let *A R B* denote an infix relationship *R* from entity type *A* to entity type *B*. Name *R* in such a way that each of the following four patterns results in an English sentence:

each A (**must** | **may) be** *R* (**one and only one** *B* | **one or more** *B-plural-form*)

Use "must" or "may" when the first role is mandatory or optional respectively. Use "one and only one" or "one or more" when the cardinality on the second role is one or many respectively. For example, the optionality/cardinality settings in Figure 1(a) verbalize as: **each** Employee **must be** an occupier of **one and only one** Room; **each** Room **may be** occupied by **one or more** Employees.

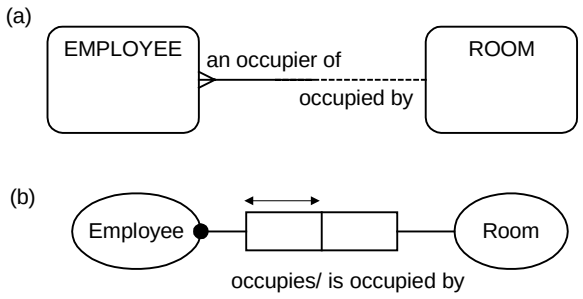

Figure 1 The ER diagram (a) is equivalent to the ORM diagram (b)

The constraints on the left hand role in the equivalent ORM model shown in Figure 1(b) verbalize as: **each** Employee occupies **some** Room; **each** Employee occupies **at most one** Room. If desired, these constraints may be combined to verbalize as: **each** Employee occupies **exactly one** Room. Since the right-hand role has no constraints, this is not normally verbalized in ORM (unlike Barker ER). However the lack of any uniqueness constraint could be verbalized explicitly as: **it is possible that the same** Room is occupied by **more than one** Employee. If no inverse reading is available, it can be verbalized as: **it is possible that more than one** Employee occupies **the same** Room. If you would like this explicit verbalization capability added as a configurable option to the verbalizer in Visio Enterprise, please email me at TerryHa@microsoft.com.

Regarding the lack of an explicit mandatory role constraint on the right-hand role, I am less inclined to want that verbalized explicitly, because it may well be unstable. If Room plays no other fact roles, the role is mandatory by implication (Room has not been declared independent), so verbalization may well confuse here. If Room does play another fact role, and we decide that some rooms may be unoccupied, we could declare this explicitly as: **it is possible that some** Room is occupied by **no** Employee. Or equivalently: **it is not necessary that each** Room is occupied by **some** Employee. If no inverse reading is available, it could be verbalized thus: **it is possible that no** Employee occupies **some** Room. If you would like an option for explicit verbalization of optional roles in Visio Enterprise, please email me your thoughts on this.

To its credit, the Barker verbalization convention is good for basic mandatory and uniqueness constraints on infix binaries. However it is far less general than ORM's approach, which applies to instances as well as types, for predicates of any arity, infix or mixfix, and covers many more kinds of constraint, with no need for pluralization. As a trivial example, the fact instance "Employee '101' an occupier of Room 23" is not proper English, but "Employee '101' occupies Room 23" is good English.

Exclusion constraints

In Barker ER notation, an exclusion constraint over two or more roles is shown as an "*exclusive arc*" connected to the roles with a small dot or circle. For example, Figure 2(a) includes the constraint that no employee may be allocated both a bus pass and a parking bay. In ORM this constraint is depicted by connecting "⊗" to the relevant roles by a dotted line, as shown in Figure 2(b).

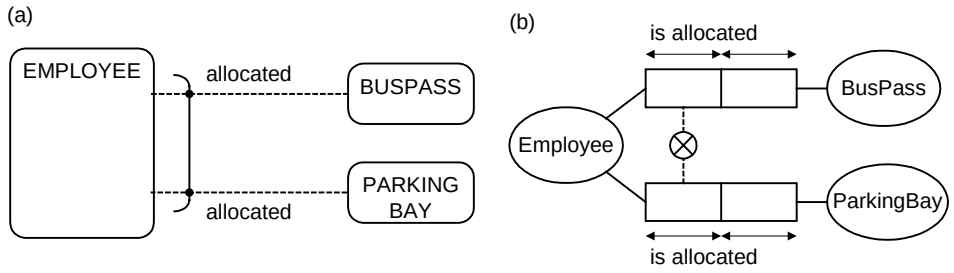

Figure 2 A simple exclusion constraint in (a) Barker ER notation and (b) ORM

To declare that two or more roles are mutually exclusive and disjunctively mandatory, the Barker notation uses the exclusive arc, but each role is shown as mandatory (solid line). For example, in Figure 3(a) each account is owned by a person or a company, but not both. This notation is liable to mislead, since it violates the orthogonality principle in language design. Viewed by itself, the first role of the association Account owned by Person would appear to be mandatory, since a solid line is used. But the role is actually optional, since superimposing the exclusive arc changes the semantics of the solid line to mean the role belongs to a set of roles that are disjunctively mandatory.

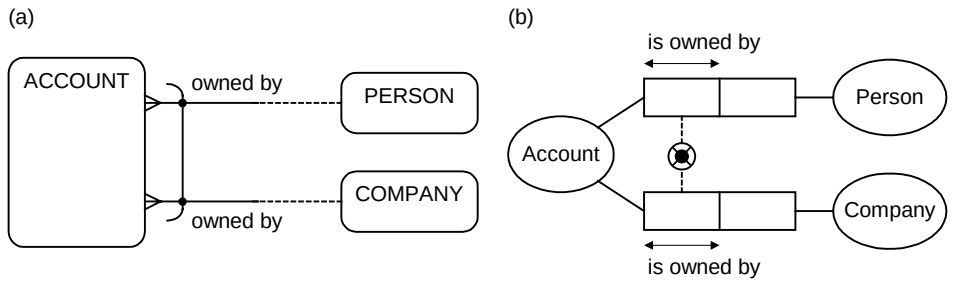

Figure 3 An exclusive-or constraint in (a) Barker ER notation and (b) ORM

Contrast this with the equivalent ORM model shown in Figure 3(b). Here an exclusion constraint ⊗ is orthogonally combined with a disjunctive mandatory (inclusive-or) constraint ⊙ to produce an exclusive-or constraint, shown here by the "lifebuoy" or partition symbol formed by overlaying one constraint symbol on the other. As an alternative, the inclusive-or and exclusion constraints may be displayed separately.

The ORM notation makes it clear that each role is individually optional, and that the exclusive-or constraint is a combination of inclusive-or and exclusion constraints. Suppose we modified our business so that the same account could be owned by both a person and a company. Removing just the exclusion constraint from the model leaves us with the inclusive-or constraint ⊙ that each account is owned by a person or company. Like UML, the Barker ER notation doesn't even have a symbol for an inclusive-or constraint, so is unable to diagram this or the many other cases of this nature that occur in practice.

In the Barker notation, a role may occur in at most one exclusive arc. ORM has no such restriction. For example, in Figure 4(a) no student can be both ethnic and aboriginal, and no student can be both an aboriginal and a migrant (these rules come from a student record system in Australia). Even if Barker notation supported unaries (it doesn't) this situation could not be handled by exclusive arcs. Like UML, Barker ER does not provide a graphic notation for exclusion constraints over role-sequences. For instance, it cannot capture the ORM pair-exclusion constraint in Figure 4(b), which declares that no person who wrote a book may review the same book. Such rules are very common. Moreover, the Barker notation cannot express any ORM subset or equality constraints at all, even over simple roles.

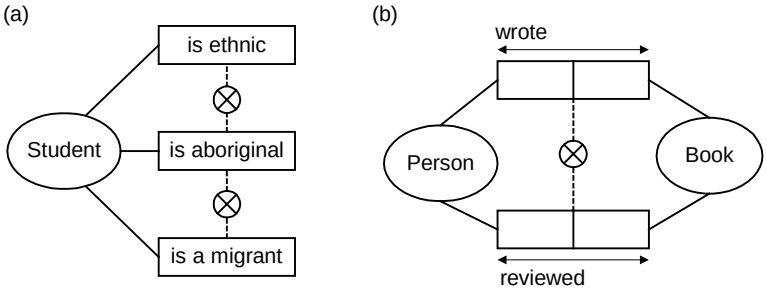

Figure 4 Some ORM exclusion constraints not handled by exclusive arcs in Barker ER notation

Frequency constraints

The Barker ER notation allows simple frequency constraints to be specified. For any positive integer *n*, a constraint of the form $= n$, $< n$, $\leq n$, $> n$, $\geq n$ may be written beside a single role to indicate the number of instances that may be associated with an instance playing the other role. For example, the frequency constraint "≤ 2" in Figure 5 indicates that each person is a child of at most two parents. In the Barker notation, this constraint is placed on the parent role, making it easy to read the constraint as a sentence starting at the other role. In ORM the constraint is placed on the child role, making it easy to see the impact of the constraint on the population (each person appears at most twice in the child role population). Unlike the Barker notation, ORM allows frequency constraints to include ranges (e.g. 2-5) and to apply to role-sequences.

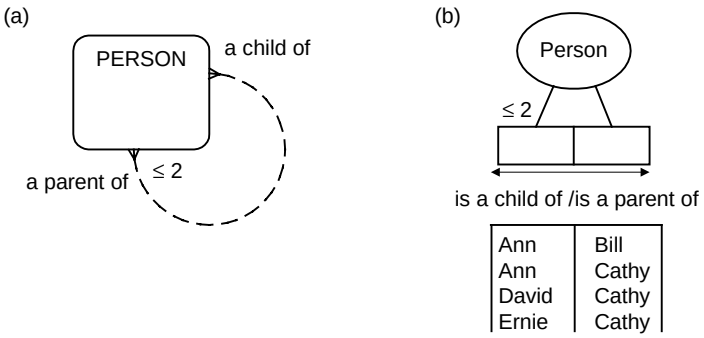

Figure 5 A simple frequency constraint in (a) Barker ER notation and (b) ORM

Subtyping

In Barker ER notation, subtyping is depicted with a version of Euler diagrams. In effect, only partitions (exclusive and exhaustive) can be displayed. For example, Figure 6(a) indicates that each patient is a male patient or female patent but not both. Like UML, ORM displays subtyping using directed acyclic graphs (DAGs). Subtype exclusion and exhaustion constraints are normally omitted in ORM, as in Figure 6(b), since they are implied by the subtype definition and other constraints (e.g. mandatory, uniqueness and value constraints on Patient is of Gender). However they can be explicitly displayed as in Figure 6(b).

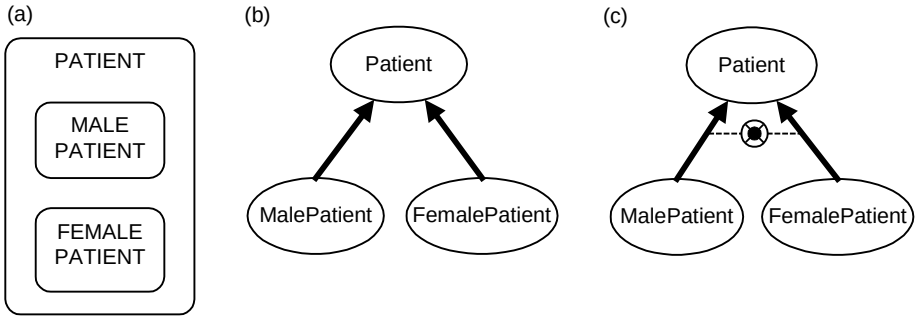

Figure 6 A subtype partition in (a) Barker ER, (b) implicit ORM and (c) explicit ORM notation

Euler diagrams are good for simple cases, since they intuitively show the subtype inside its supertype. However unlike DAGs, they are hopeless for complex cases (e.g. many overlapping subtypes), and they make it inconvenient to attach details to the subtypes. For the latter reason, attributes are often omitted from subtypes when the Barker notation is used.

In the Barker notation, if the original subtype list is not exhaustive, an "Other" subtype is added to make it so, even if it plays no specific role. For example, in Figure 7 a vehicle is a car or truck or possibly something else, and a car is a sedan or wagon or possible something else.

A major problem with the Barker notation for subtyping is that it does not depict overlapping subtypes (e.g. Manager and FemaleEmployee as subtypes of Employee) or multiple inheritance (e.g. FemaleManager as a subtype of FemaleEmployee and Manager). While it is possible to implement multiple inheritance in single inheritance systems (e.g. Java) by using some low level tricks, for conceptual modeling purposes multiple inheritance should be simply modeled as multiple inheritance.

As a final comparison point about subtyping, Barker ER lacks ORM's capability for formal subtype definitions and context-dependent identification schemes.

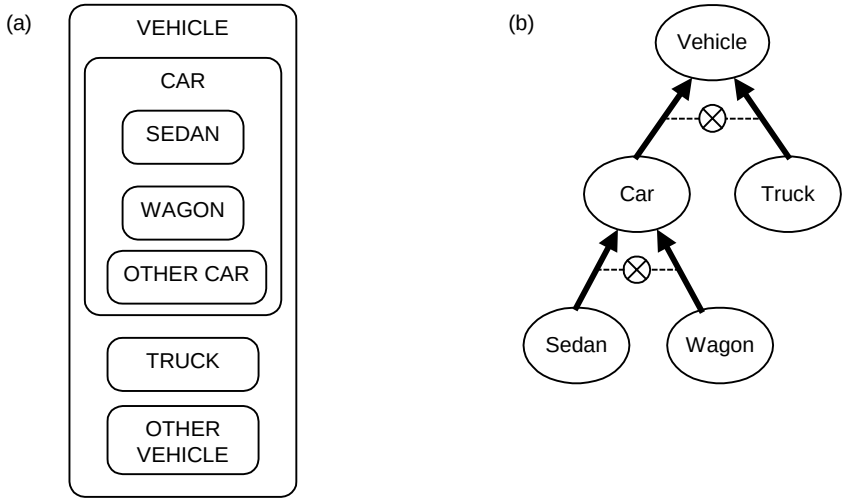

Figure 7 Non-exhaustive, exclusive subtypes in (a) Barker ER and (b) ORM

Non-transferable relationships

In addition to its static constraint notation, Barker ER includes a dynamic "changeability constraint" for marking "*non-transferable relationships*". This constraint declares that once an instance of an entity type plays a role with an object, it cannot ever play this role with another object. This is indicated by adding an open diamond to the constrained role. For example, Figure 8(a) declares that the birth country of a person is non-transferable.

As indicated in Figure 8(b), ORM does not currently include a notation for this constraint. It would be possible to add a notation for this (as well as UML's changeability settings of changeable, frozen, addOnly), but it is at least debatable whether this is advisable. If we were to add such a notation, we would need to ensure that the implemented model is still open to error corrections by duly authorized users. For example, if my birth country was mistakenly entered as Austria, it should be possible to change this to Australia. For further discussion on this issue, see my comments on changeability properties in [2]. If you have some strong views on this issue, please email me your thoughts.

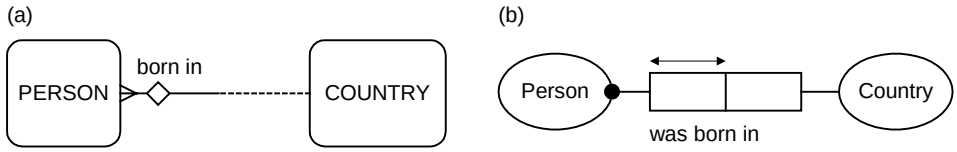

Figure 8 Non-transferable nature of a relationship is declared in (a) Barker ER, but not in (b) ORM

Conclusion

The Barker ER notation does a good job of expressing simple mandatory, uniqueness, exclusion and frequency constraints, simple subtyping and also non-transferable relationships. However, if a feature is modeled as an attribute instead of as a relationship, very few of these constraints can be specified for it. In contrast to ORM, the Barker ER notation does not support unary, *n*-ary or objectified associations (nesting). Moreover it lacks support for most of the advanced ORM constraints (e.g. subset, multi-role exclusion, ring constraints and join constraints). It does not include a formal textual language such as ConQuer for specifying queries, other constraints and derivation rules at the conceptual level. Nevertheless it is better than many other notations for ER modeling, and is still widely used. If you ever need to specify a model in Barker ER notation, I suggest you first do the model in ORM, then map it to the Barker notation and make a note of any rules that can't be expressed there diagrammatically.

Next issues

Later articles in this series will examine the Information Engineering notation for ER, before concluding with a discussion of IDEF1X.

References

1. Barker, R. 1990, *CASE*Method: Tasks and Deliverables*, Addison-Wesley, Wokingham, England.
2. Halpin, T.A. 1999, 'UML data models from an ORM perspective: Part 10', *Journal of Conceptual Modeling*, InConcept, Minneapolis USA.