Entity Relationship modeling from an ORM perspective: Part 1

by Dr. Terry Halpin

Director of Database Strategy, Visio Corporation

This is a revised version of a paper that first appeared in the December 1999 issue of the Journal of Conceptual Modeling*, published by InConcept.*

This paper is the first in a series of articles examining data modeling in the Entity Relationship (ER) approach from the perspective of Object Role Modeling (ORM). This article examines basic aspects of the Barker notation for ER.

Introduction

Entity Relationship modeling (ER) views the application domain in terms of entities that have attributes and participate in relationships. For example, the fact that an employee was born on a date is modeled by assigning a birthdate attribute to the Employee entity type, whereas the fact that an employee works for a department is modeled as a relationship between them. This view of the world is quite intuitive, and in spite of the recent rise of UML for modeling object-oriented applications, ER is still the most popular data modeling approach for database applications.

The ER approach was originally proposed by Peter Chen in 1976, in the very first issue of an influential ACM journal [2]. As shown in Figure 1, Chen's original notation used rectangles for entity types, diamonds for relationships, and ellipses for attributes. The double ellipse indicates unique identifier attributes, and the "n and "1" indicate the relationship is many to one (each employee works for at most one department, but many employees may work for the same department).

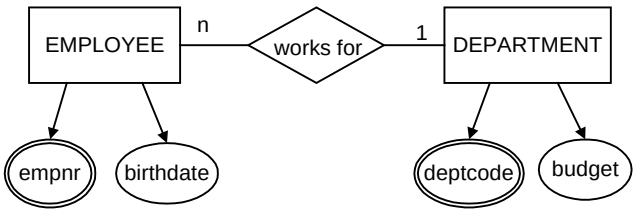

Figure 1 An early ER notation used by Chen

The direction in which relationship names are to be read is formally undecided, unless we add some additional marks (e.g. arrows) or rules (e.g. always read from left to right and from top to bottom). For example, does the employee work for the department, or does the department work for the employee? Although we can use our background knowledge to informally disambiguate this example, it is quite common nowadays to see ER models with relationships whose intended direction can only be guessed at by anybody other than the model's creator. For example, consider the impact of misreading the intended direction for the following: Person killed Animal; Person is loved by Person. This problem is exacerbated if the verb phrase used to name the relationship is shortened to one word (e.g. "work", "love"), unfortunately still a fairly common practice.

Chen's notation evolved over time. His current ER-Designer tool uses hexagons instead of diamonds, and supports n-ary relationships. Outside academia, Chen's notation seems to be rarely used nowadays, so I'll say no more about it here. One of the problems with the ER approach is that there are so many versions of it, with no single standard. In industrial practice, the most popular versions of ER are the Barker and Information Engineering (IE) notations. Another popular data modeling notation is IDEF1X, but since this is a hybrid of ER and relational notation, I don't count it as a true ER representative. As discussed elsewhere [4], UML class diagrams can be regarded as an extended version of ER. The rest of this article focuses on basic aspects of the Barker notation for ER. Later articles will examine IE and IDEF1X.

Barker ER: the basics

I use the term "Barker ER" for the ER notation discussed in the classic treatment by Richard Barker [1]. While Oracle Corporation has long used this notation in its CASE tools, Oracle's Object Designer tool now supports UML as an alternative to its traditional ER notation. For database applications, many modelers still prefer the Barker ER notation in preference to UML, and it will be interesting to see whether this changes over time. Dave Hay, an experienced modeler and ardent fan of the Barker ER notation, argues that "there is no such thing as 'object-oriented analysis'" [6], only object-oriented design, and that "UML is … not suitable for analyzing business requirements in cooperation with business people"[7].

While I agree with Dave Hay that UML class diagrams are less than ideal for data modeling, I feel that his preferred ER notation shares some of UML's weaknesses in being attribute-based. As I've discussed before in a UML context [4, 5], using attributes in a base conceptual model adds complexity and instability, while making it harder to validate models with domain experts using verbalization and sample populations. Attributes are great for logical design, since they allow compact diagrams that directly represent the data structures (e.g. relations or object-relations) used for the actual design. However when I'm performing conceptual analysis, I just want to know what the *facts* and *rules* are about the business, and I want to communicate this information in sentences, so that the

model can be understood by the domain experts. I sure don't want to bother about how facts are grouped into multi-fact structures. Whether some fact will end up in the design as an attribute is not a conceptual issue to me. As Ron Ross says, "Sponsors of business rule projects must sign off on the sentences—not on graphical data models. Most methodologies and CASE tools have this more or less backwards" [7, p.15]. The ORM reporting facilities in Visio Enterprise allow the domain expert to inspect ORM models fully verbalized into sentences with examples, making validation much easier and safer.

Now that I've stated my bias up front, let's examine the Barker ER notation itself. The basic conventions are illustrated in Figure 2. *Entity types* are shown as soft rectangles (rounded corners) with their name in capitals. *Attributes* are written below the entity type name. Some constraint information may appear before an attribute name. A "#" indicates that the attribute is the primary identifier of the entity type, or at least a component of its primary identification scheme. A "*" or heavy dot "•" indicates the attribute is mandatory (i.e. each instance in the database population of the entity type must have a non-null value recorded for this attribute). A "°" indicates the attribute is optional. Some modelers also use a period "." to indicate the attribute is not part of the identifier.

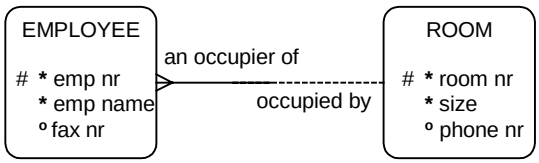

Figure 2 The basic Barker ER notation

Relationships are restricted to binaries (no unaries, ternaries or longer relationships), and are shown as lines with a relationship name at the end from which that relationship name is to be read. This name placement overcomes the ambiguous direction problem mentioned earlier. Both forward and inverse readings may be displayed for a binary relationship, one on either side of the line. This makes the Barker notation superior to UML for verbalizing relationships.

From an ORM perspective, each end (or half) of a relationship line corresponds to a role. Like ORM, Barker ER treats role *optionality* and *cardinality* as distinct, orthogonal concepts, instead of lumping them together into a single concept (e.g. multiplicity in UML). A solid line-half denotes a mandatory role, and a dotted line-half indicates an optional role. For cardinality, a crow's foot intuitively indicates "many", by its many "toes". The absence of a crow's foot intuitively indicates "one". The crow's foot notation was invented by Gordon Everest, who originally used the term "inverted arrow" [3] but now calls it a "fork". Figure 3 shows the correspondence with the ORM notation for uniqueness and mandatory role constraints.

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 3(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. This 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, 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.

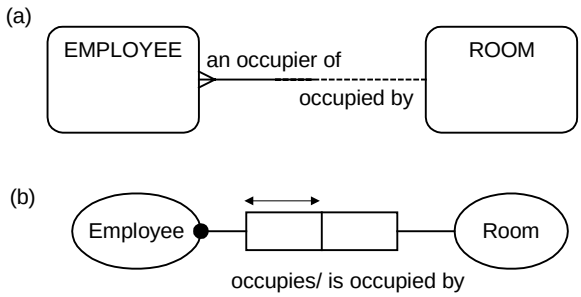

Figure 3 The ER diagram (a) is equivalent to the ORM diagram (b)

If each of the two roles in a binary association may be assigned one of optional/mandatory and one of many/one, there are sixteen patterns. The equivalent Barker ER and ORM diagrams for the first eight of these cases are shown in Figure 4.

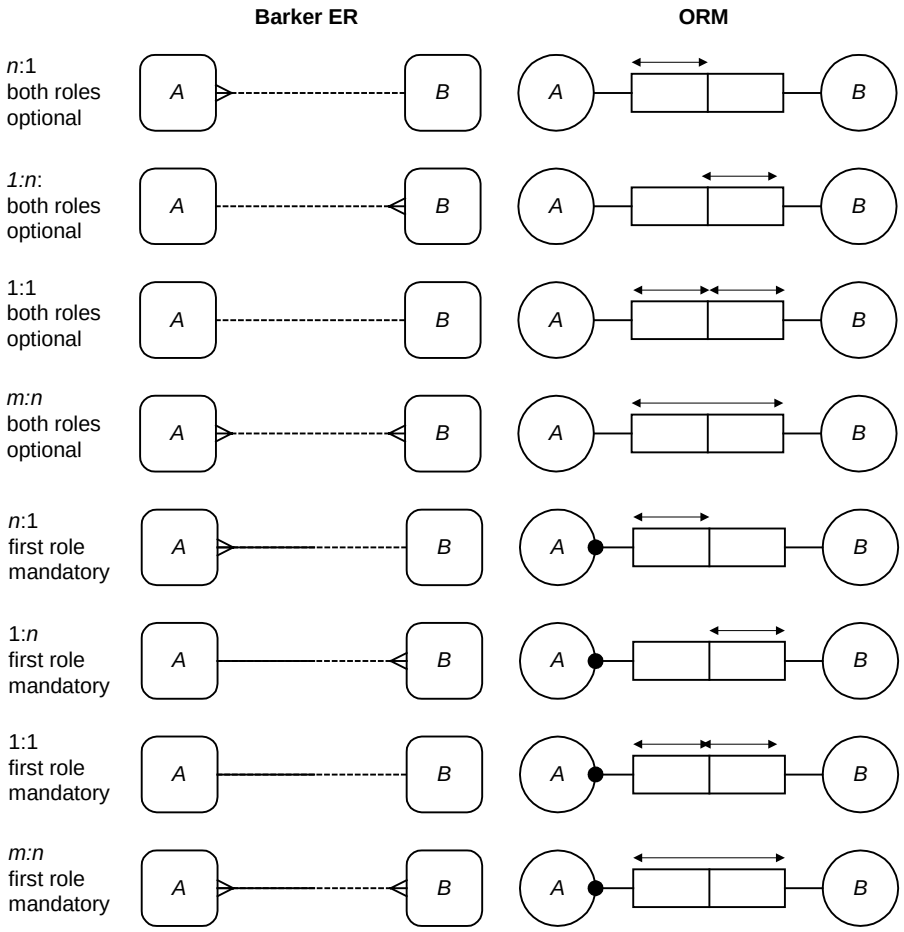

Figure 4 Some equivalent cases

The other eight cases are shown in Figure 5. Although all eight are legal in ORM, the last case where both roles of a many:many relationship are mandatory is considered illegal by Barker.

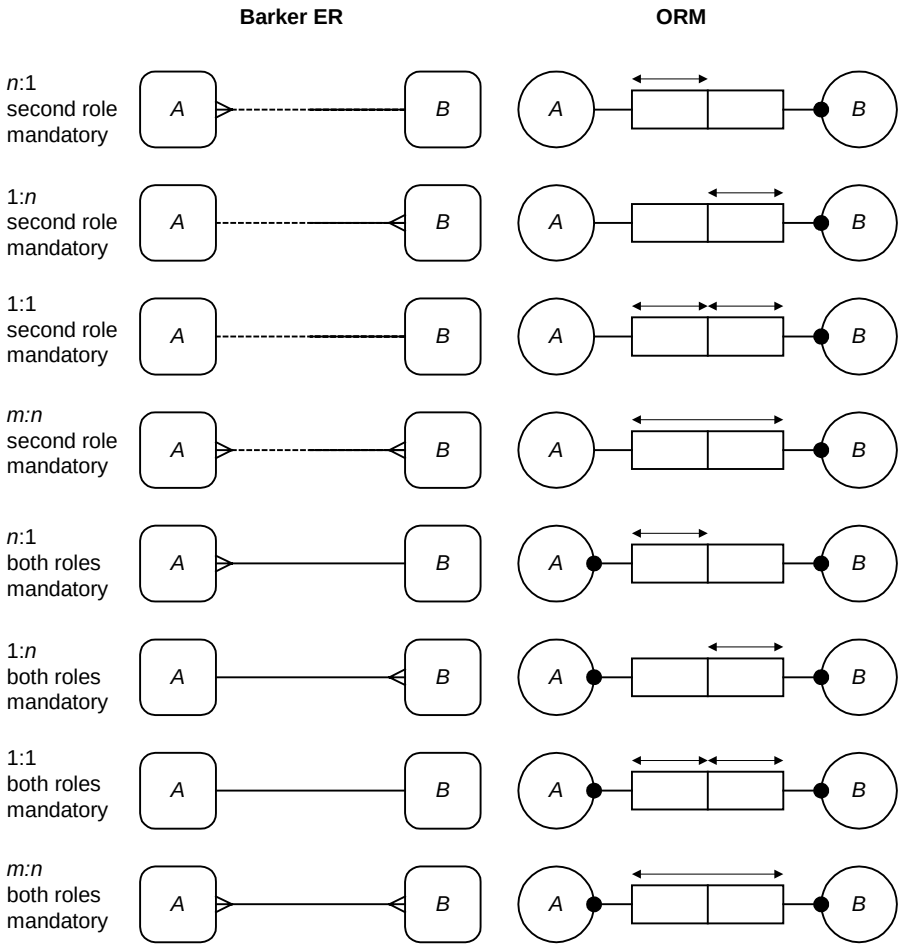

Figure 5 Other equivalent cases

Ring associations that considered illegal by Barker are shown in Figure 6(a). Although rare, they sometimes occur in reality, so should be allowed at the conceptual level, as permitted in ORM. As an exercise, you may wish to invent satisfying populations for the ORM associations in Figure 6 (b). Although considered illegal by Barker, at least some of these patterns are allowed in Oracle's CASE tools.

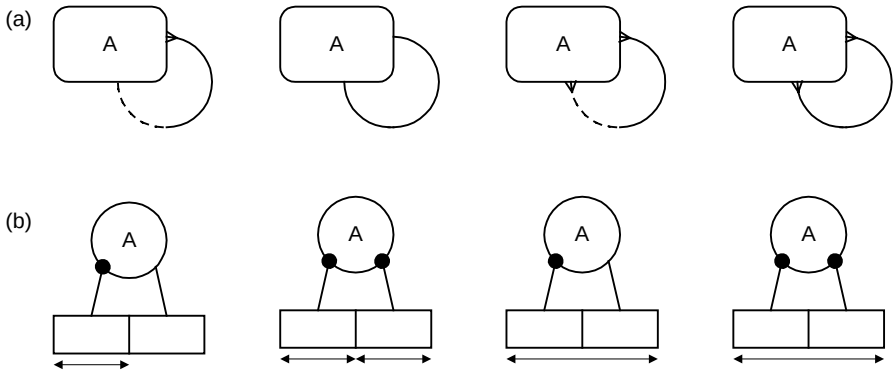

Figure 6 Illegal ring associations in Barker ER (a) that are rare but allowed in ORM (b)

In Barker ER, 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. In Figure 7 for example, Employee and Building have simple identifiers, but Room has a composite reference scheme, being identified partly by its room number and partly by the building in which it is included.

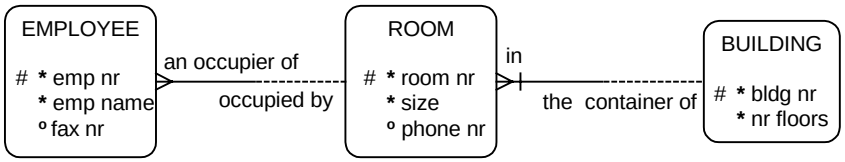

Figure 7 Room is identified by combining its room nr and its relationship to Building

The use of identification bars provides some of the functionality afforded by external uniqueness constraints in ORM. For example, the schemas in Figure 8 are equivalent. The other attributes of Room and Building in ORM would be modeled in ORM as relationships. ORM's external uniqueness notation seems to me to convey more intuitively the idea that each RoomNr, Building combination is unique (i.e. refers to at most one room). But maybe I'm biased. At any rate, this constraint (as well as any other graphic constraint) can be automatically verbalized in natural language.

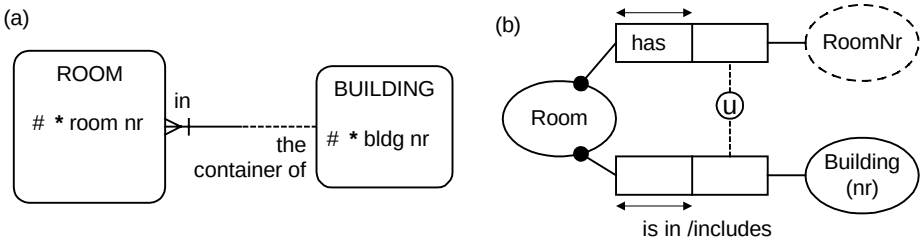

Figure 8 Composite identification in Barker ER (a) and ORM (b)

Some people misread the bar notation for composite identification as a "1", since this is what the symbol means in many other ER notations. But this isn't a problem if you don't have to work with multiple versions of ER. The main problem with the "#" and bar notations is that they cannot be used to declare uniqueness constraints that are not used for a primary identification scheme. A second problem is that they are two very different notations for the same fundamental concept (uniqueness). Because ORM allows constraints to be used wherever they make sense, and always uses relationships instead of attributes, it doesn't have these problems. An example may help illustrate some of these ideas. Suppose we wanted to model the information shown in Table 1, as well as other facts about rooms.

Table 1 A simple data use case for room scheduling

Room	*Time*	*ActivityCode*	*ActivityName*
20	Mon 9 am	VMC	VisioModeler class
20	Tue 2 pm	VMC	VisioModeler class
33	Mon 9 am	AQD	ActiveQuery demo
33	Fri 5 pm	SP	Staff party
…	…	…	…

The table suggests that rooms can be simply identified by room numbers, so let's accept that. One way of modeling the situation in Barker ER is shown in Figure 9. Here the bar notation is used to show that RoomTimeSlot is identified by combining its time and room number.

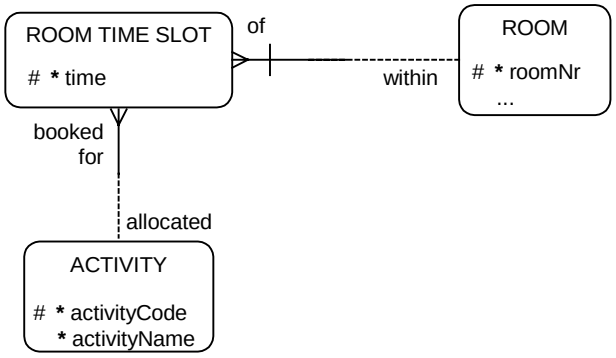

Figure 9 An ER diagram for room scheduling

The use of attributes in this model makes it hard to verbalize and populate the schema for validation purposes. Moreover, there is at least one constraint missing. Compare this with the populated ORM model for the same situation (Figure 10). Here the facts are naturally verbalized as a ternary (Room at Time is booked for Activity) and a binary (Activity has ActivityName). The associated fact tables include the original facts, as well as counter-facts (italicized) to test the constraints. The first counter-row (20, Mon 9 am, AQD) tests the uniqueness constraint that a room at a time is booked for at most one activity. The second counter-row tests the uniqueness constraint that at most one room can be booked for a given activity at a given time. This constraint may well be wrong, but

at least we can express it and test it in ORM. With the ER model there is no way of even specifying the constraint, much less testing it.

The counter rows (SP, Sales phonecalls) and (PTY, Staff party) are designed to check the uniqueness constraints that each Activity has at most one Activity name and vice versa. If these are rejected, the association really is 1:1, as its basic population suggests. Since the ER notation being discussed doesn't include a way of indicating that attributes other than the primary identifier are unique, it isn't very helpful here. As a small point, the Y2K row has been added to the original population to indicate that it is possible for some listed activities to be unscheduled.

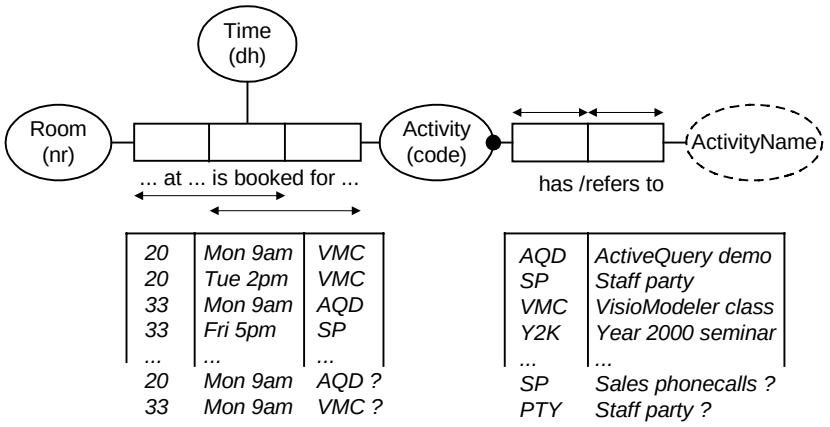

Figure 10 An ORM diagram for room scheduling, with sample and counter data

In case it looks like I'm just bashing attribute-based approaches like ER in this article, let me say again that I find attribute-based models useful for compact overviews and for getting closer to the implementation model. However I generate these by mapping from ORM, which I use exclusively for conceptual analysis. This makes it easier to get the model right in the first place, and to modify it as the underlying domain evolves. Unlike ER (and UML for that matter), ORM was built from a linguistic basis, and its graphic notation was carefully chosen to exploit the potential of sample populations. To reap the benefits of verbalization and population for communication with and validation by domain experts, it's better to use a language that was designed with this in mind. An added benefit of ORM is that its graphic notation can capture many more business rules than popular ER notations.

Next issues

Later articles in this series will consider more advanced aspects of the Barker ER notation, including exclusion constraints, frequency constraints, subtyping and non-transferable relationships, and then 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. Chen, P.P. 1976, 'The entity-relationship model—towards a unified view of data', *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36.

3. Everest, G. 1976, 'Basic Data Structure Models Explained with a Common Example', *Proc. Fifth Texas Conference on Computing Systems*, (Austin, TX, 1976 October 18-19), IEEE Computer Society publications office, Long Beach, CA, pp. 39-45.

4. Halpin, T.A. 1998-9, 'UML data models from an ORM perspective: Parts 1-10', *Journal of Conceptual Modeling*, InConcept, Minneapolis USA.

5. Halpin, T.A. & Bloesch, A.C. 1999, 'Data modeling in UML and ORM: a comparison', *Journal of Database Management*, vol. 10, no. 4, Idea group Publishing Company, Hershey, USA, pp. 4-13.

6. Hay, D.C. 1999, 'There is no object-oriented analysis', *DataToKnowledge Newsletter*, vol. 27, no. 1, Business Rule Solutions, Inc., Houston TX, USA.

7. Hay, D.C. 1999, 'Object orientation and information engineering: UML', *The Data Administration Newsletter*, no. 9, (June 1999), ed. R.S. Reiner, available online at www.tdan.com.

8. Ross, R.G. 1998, *Business Rule Concepts*, Business Rule Solutions, Inc., Houston TX, USA.