UML Data Models From An ORM Perspective: Part 3

by Dr. Terry Halpin, BSc, DipEd, BA, MLitStud, PhD
Director of Database Strategy, Visio Corporation

This paper appeared in the June 1998 issue of the Journal of Conceptual Modeling
published by Information Conceptual Modeling, Inc. and is reproduced here by permission.

This paper is the third in a series of articles examining data modeling in the Unified Modeling Language (UML) from the perspective of Object Role Modeling (ORM). Part 1 provided some historical background on both approaches, identified design criteria for modeling languages, and discussed how object reference and single-valued attributes are modeled in both. Part 2 compared UML multi-valued attributes with ORM relationship types, and discussed basic constraints on both, as well as instantiation using UML object diagrams or ORM fact tables. This third issue compares UML associations and related multiplicity constraints with ORM relationship types and related uniqueness, mandatory role and frequency constraints. It also contrasts instantiation of associations using UML object diagrams and ORM fact tables.

Associations

Before discussing UML associations in detail, we review some ideas discussed previously. Attributes in UML are depicted as relationship types in ORM. A *relationship instance* in ORM is called a *link* in UML (e.g. Employee 101 works for Company 'Visio'). A *relationship type* in ORM is called an *association* in UML (e.g. Employee works for Company). In both UML and ORM, a *role* is a part played in a relationship. The number of roles in a relationship is its *arity*.

ORM allows relationships of any arity. Each relationship type has at least one reading or predicate name. An *n*-ary relationship may have up to *n* readings (one starting at each role), to provide natural verbalization of constraints and navigation paths in any direction. A predicate is an elementary sentence with holes in it for object terms. In ORM these object holes may appear at any position in the predicate (*mixfix* notation), and are denoted by an ellipsis "… " if the predicate is not infix-binary. Mixfix notation enables natural verbalization of sentences in any language (e.g. in Japanese, verbs come at the end of sentences).

ORM sentence types and constraints may be specified either textually or graphically. Visio's InfoModeler can automatically transform between the graphical and textual representations. In an ORM diagram, roles appear as *boxes*, connected by a line to their object type. InfoModeler allows role names to be added on a properties sheet rather than on the diagram; in principle however, an ORM tool could display role names directly on the diagram (preferably with the display toggled by the user to avoid clutter). A *predicate* appears as a named, contiguous sequence of role boxes. Since these boxes are set out in a

line, fact types may be conveniently populated with tables holding multiple fact instances, one column for each role. This allows all fact types and constraints to be validated by verbalization as well as sample populations. Communication between modeler and domain expert takes place in a familiar language, backed up by population checks.

UML uses Boolean attributes instead of unary relationships, but allows relationships of all other arities. Each association may be given at most one name, and this is optional. Association names are normally shown in italics, starting with a capital letter. Binary associations are depicted as *lines* between classes. Association lines may include elbows to assist with layout or when needed (e.g. for ring relationships). Association roles appear simply as line ends instead of boxes, but may optionally be given role names. Once added, role names may not be suppressed. Verbalization into sentences is possible only for infix binaries, and then only by naming the association with a predicate name (e.g. "Employs") and using an optional marker "**}**" to denote the direction.

Figure 1 depicts two binary associations in both UML and ORM. On the UML diagram we have chosen to display the association names, their directional markers and all the role names: all of these could have been omitted. To avoid ambiguity however, either the directed association name or the role names should be shown. In the ORM diagram, both forward and inverse predicate names have been shown: at most one of these may be omitted. Role names are not displayed on the ORM diagram but may be added (as discussed above).

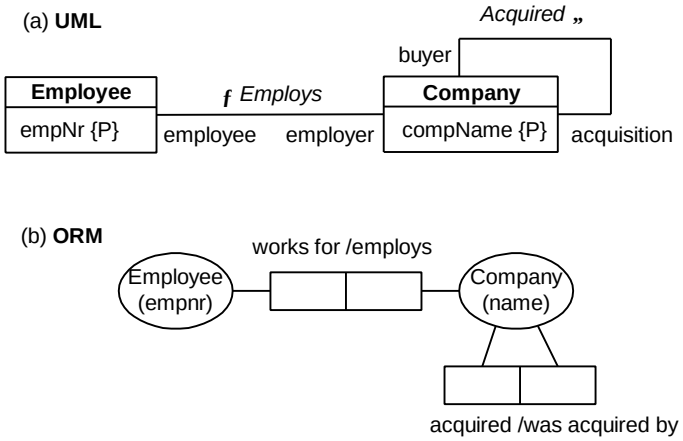

Figure 1: Binary associations in (a) UML and (b) ORM

Ternary and higher arity associations in UML depicted as a *diamond* connected by lines to the classes. Because many lines are used to denote the association, directional verbalization is ruled out, so the diagram can't be used to communicate in terms of sentences. This non-linear layout also often makes it impractical to conveniently populate associations with multiple instances. Add to this the impracticality of displaying multiple populations of attributes, and it is clear that class diagrams are of little use for population checks.

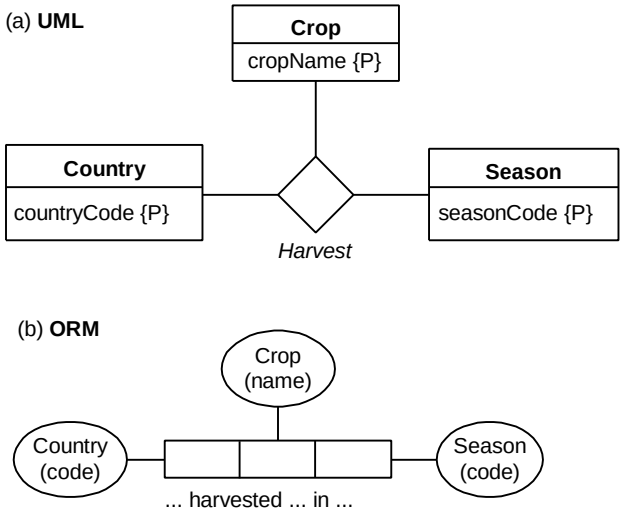

Figure 2: A ternary association in (a) UML and (b) ORM

As discussed in the previous issue, UML does provide *object diagrams* for instantiation, but these are convenient only for populating associations with a *single* instance. Adding multiple instances leads to a mess (e.g. [0], p. 31). Hence, as noted in the UML Notation Guide, "the use of object diagrams is fairly limited".

Multiplicity constraints on associations

The previous issues discussed how UML depicts multiplicity constraints on attributes. A similar notation is used for associations, where the relevant multiplicities are written beside the relevant roles. Figure 3(a) adds the relevant multiplicity constraints to Figure 1(a). A "*" abbreviates "0..*", meaning "zero or more", "1" abbreviates "1..1", meaning "exactly one", and "0..1" means "at most one". Unlike some ER notations, UML places each multiplicity constraint on the "far role", in the direction in which the association is read. Hence the constraints in this example mean: each company employs zero or more employees; each employee is employed by exactly one company; each company acquired zero or more companies; and each company was acquired by at most one company.

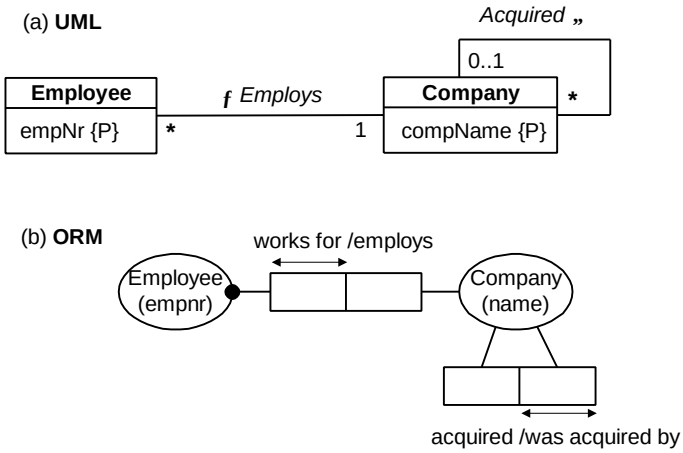

Figure 3: UML multiplicity constraints captured in ORM by uniqueness and mandatory role constraints.

The corresponding ORM constraints are depicted in Figure 3(b). Recall that multiplicity covers both cardinality (frequency) and optionality. Here the mandatory role constraint indicates that each employee works for at least one company, and the uniqueness constraints indicate that each employee works for at most one company, and each company was acquired by at most one company. As usual, the ORM notation facilitates verbalization and population. InfoModeler allows these constraints to be entered graphically, or by answering multiplicity questions, or by induction from sample populations, and can automatically verbalize the constraints.

For binary associations, there are four possible uniqueness constraint patters (*n*:1, 1:*n*; 1:1, *m*:*n*) and four possible mandatory role patterns (only the left role mandatory, only the right role mandatory, both roles mandatory, both roles optional). Hence if we restrict ourselves to a maximum frequency of one, there are 16 possible multiplicity combinations for binary associations. The first four of these are shown in Figure 4, covering the cases where both roles are optional.

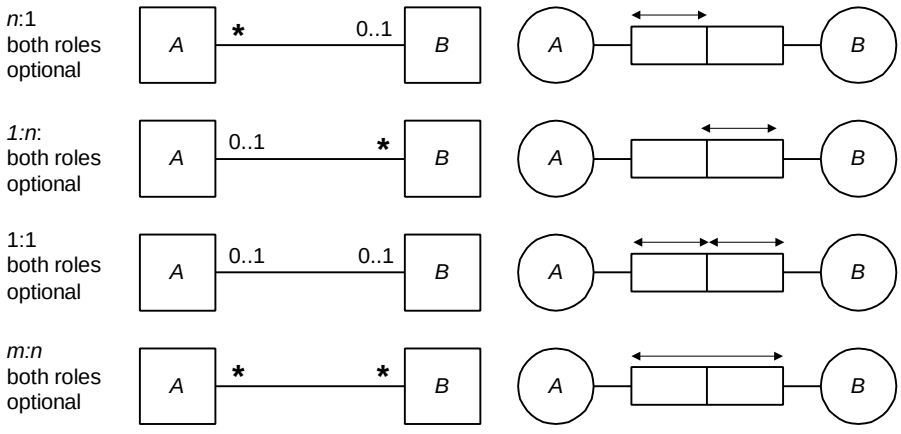

Figure 4: Equivalent constraint patterns in UML and ORM, where both roles are optional

The next four cases, shown in Figure 5, cover the situation where the first role is mandatory and the second role is optional.

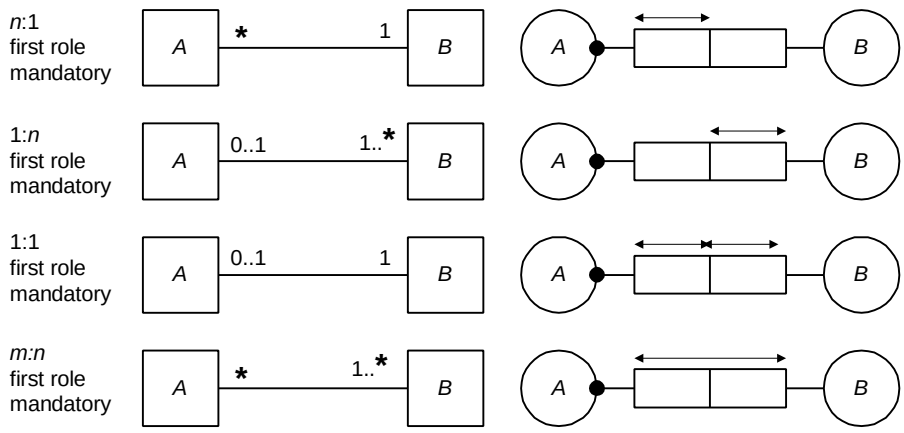

Figure 5: Equivalent constraint patterns in UML and ORM, where first role is mandatory

The next four cases, shown in Figure 6, cover the situation where the second role is mandatory and the first role is optional. Finally, Figure 7 covers the four cases where both roles are mandatory.

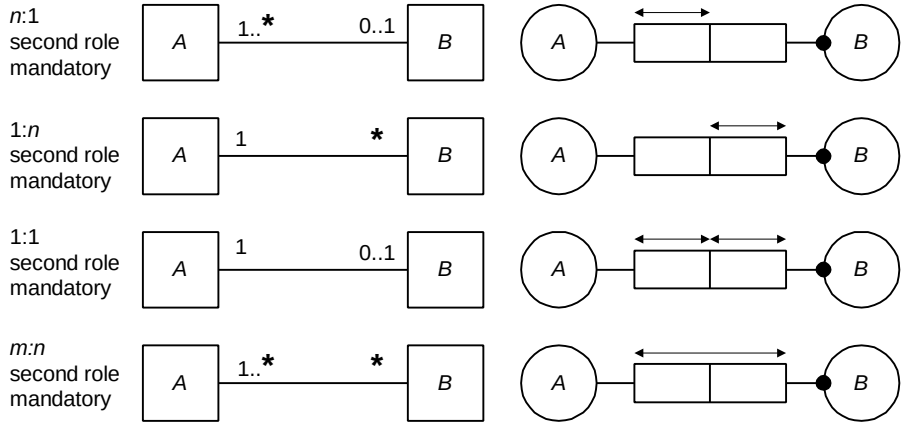

Figure 6: Equivalent constraint patterns in UML and ORM, where second role is mandatory

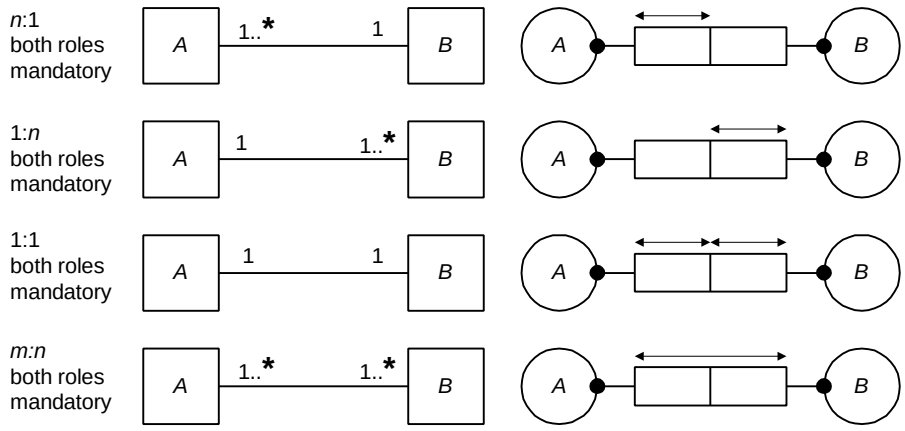

Figure 7: Equivalent constraint patterns in UML and ORM, where both roles are mandatory

In the previous issue, we discussed attribute multiplicity constraints involving occurrence frequency lists and/or ranges containing frequencies other than zero or one (e.g. "1..7, 10"). For such cases, ORM uses general frequency constraints instead of uniqueness constraints. In a similar way, both UML and ORM cater for multiplicity constraints of arbitrary complexity on single roles. As discussed in a later issue, ORM is actually more expressive in this regard since it can apply such constraints to arbitrary collections of roles.

An *internal constraint* applies to roles in a single association. For an elementary *n*-ary association, each internal uniqueness constraint must span at least *n*-1 roles. Unlike many ER notations, UML and ORM can express all possible internal uniqueness constraints. In UML, a multiplicity constraint on a role of an *n*-ary association effectively constrains the population of the other roles combined. For example, Figure 8 is a UML diagram for a ternary association in which both Room-Time and Time-Activity pairs are unique. For simplicity, we have omitted the conceptual reference schemes for the classes.

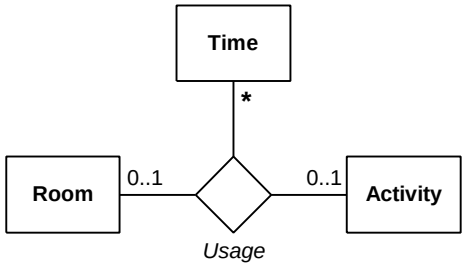

Figure 8: Multiplicity constraints on a ternary in UML

An ORM depiction of the same association is shown in Figure 9, including the reference schemes and sample population. The left-hand uniqueness constraint indicates that Room-Time is unique (i.e. for any given room and time, there is at most one activity). The right-hand uniqueness constraint indicates that Time-Activity is unique (i.e. for any

given time and activity, at most one room is used). Note how useful the population of the ternary is for checking the constraints. For example, if Time-Activity is not really unique, this can be tested by adding a counterexample and asking the client whether this is possible.

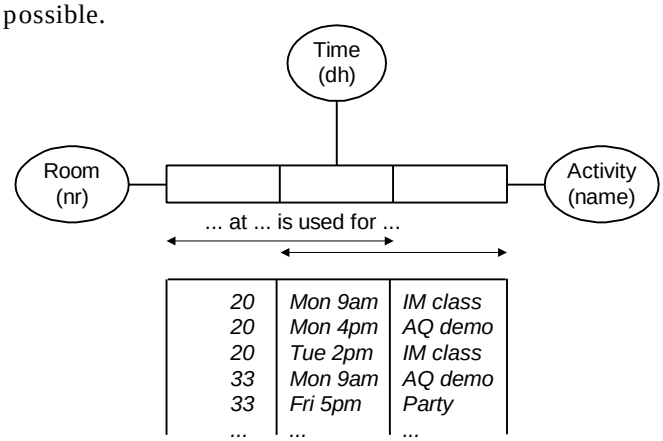

Figure 9: An ORM ternary with a sample population

Later issues

The next issue looks at associations in more detail, covering some advanced constraints, and then contrasts ORM nesting with UML association classes, and ORM co-referencing with UML qualified associations. Later issues discuss more advanced constraints, aggregation, subtyping, derivation rules and queries.

References

Blaha, M. & Premerlani, W. 1998, Object-Oriented Modeling and Design for Database Applications, Prentice Hall, New Jersey.