UML data models from an ORM perspective: Part 4

by Dr. Terry Halpin

Director of Database Strategy, Visio Corporation

This article first appeared in the August 1998 issue of the Journal of Conceptual Modeling*, published by InConcept.*

This paper is the fourth 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 historical background and design criteria for modeling languages, and discussed object reference and single-valued attributes. Part 2 discussed multi-valued attributes, basic constraints, and instantiation using UML object diagrams or ORM fact tables. Part 3 compared UML associations and related multiplicity constraints with ORM relationship types and related uniqueness, mandatory role and frequency constraints; it also contrasted instantiation of associations using UML object diagrams and ORM fact tables. In Part 4 we look at associations in more detail, contrasting ORM nesting with UML association classes, and ORM co-referencing with UML qualified associations, then discuss exclusion constraints, and summarize how the two methods compare with respect to terms and notations for data structures and instances.

Association classes

Unlike many ER versions, both UML and ORM allow associations to be objectified as first class object types, *called association classes* in UML and *nested object types* (or *objectified relationship types*) in ORM. UML requires the same name to be used for the original association and the association class, impeding natural verbalization of at least one of these constructs. In contrast, ORM nesting is based on linguistic *nominalization* (a verb phrase is objectified by a noun phrase), thus allowing both to be verbalized naturally, with different names for each. When an association is objectified, VisioModeler automatically creates a name for the nested object type, which you are free to edit. UML allows the association class name to be displayed on the association or the association class, or both.

In spite of identifying association classes with their underlying association, UML displays them separately, making the connection by a dashed line (see Figure 1). Each person may write many papers, and each paper is written by at least one person. In the UML depiction, we have used "{P}" to indicate the primary reference attributes used for

human communication about persons and papers. Since authorship is *m*:*n*, the association class Writing has a primary reference scheme based on the combination of person and paper (e.g. the writing by person 'Norma Jones' of paper 33). The optional period attribute stores how long that person took to write that paper. Instead of distancing the objectified association from its underlying association, ORM intuitively envelops the association with an object type frame. Writing is marked independent (displayed with "!") to indicate that a writing object may exist, independently of whether we record its period. ORM displays Period as an object type, not an attribute, and includes its unit.

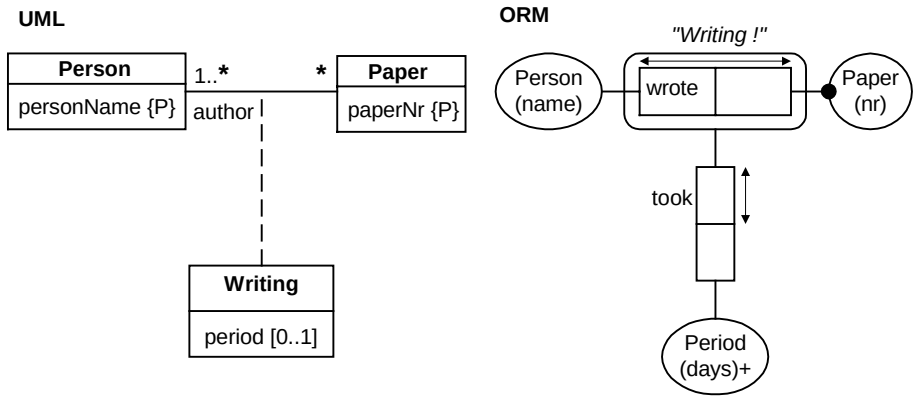

Figure 1: Writing is depicted as an objectified association in UML and ORM

Objectified relationships in standard ORM must have at least two roles, and must either have a single, spanning uniqueness constraint or be a 1:1 binary. A Dutch variant of ORM known as FCO-IM allows unaries to be objectified, but this adds no extra expressibility and is not supported in Visio technology. UML allows any association (binary and above) to be objectified into a class, regardless of its multiplicity constraints. In particular UML allows objectification of *n*:1 associations, unlike ORM (see Figure 2).

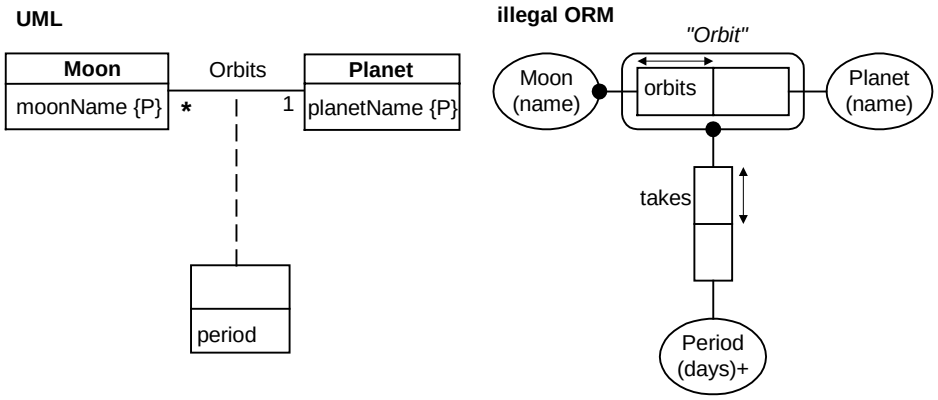

Figure 2: Objectification of *n*:1 associations is allowed in UML but not ORM

ORM currently forbids such cases, mainly to encourage modelers to conceptualize facts in elementary rather than compound form. For example, since each moon orbits only one planet, we can specify its orbital period without having to mention its planet. Hence

ORM requires this case to be modeled using two separate fact types, as shown in Figure 3. This also facilitates removal/addition of mandatory role constraints on the fact types independently (e.g. the nested version has to be completely remodeled if we now decide to keep period facts mandatory but make planet facts optional). However, if an experienced modeler aware of the implications still finds it easier to think about a situation as a nested *n*:1 association, there may be some argument for relaxing ORM's restriction, just as we relaxed it for 1:1 cases to avoid arbitrary decisions about relative importance. If enough people feel this way, ORM could be relaxed to downgrade this error to a warning, and mapping algorithms would add a pre-processing step to re-attach roles and adjust constraints internally.

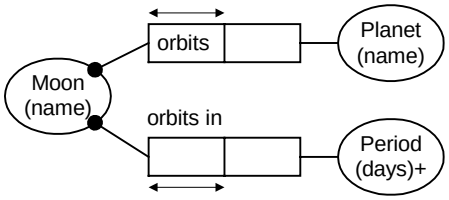

Figure 3: ORM models *n*:1 association classes instead as separate, elementary fact types

Qualified associations

In Part 2 of this series, we saw that UML has no graphic notation to capture ORM external uniqueness constraints across roles that are remodeled as attributes in UML. Hence we introduced our own {U*n*} notation to append as textual constraints to the constrained attributes (see Part 2, Figures 4 and 5). Simple cases where ORM uses an external uniqueness constraint for *co-referencing* can also be modeled in UML using *qualified associations*. Here, instead of depicting the relevant ORM roles or object types as attributes, UML uses a class, adjacent to a *qualifier*, through which connection is made to the relevant association role. A qualifier in UML is a set of one or more attributes, whose values can be used to partition the class, and is depicted as a rectangular box enclosing its attributes. Figure 4 is based on an example from the UML standard document [4], along with the ORM counterpart.

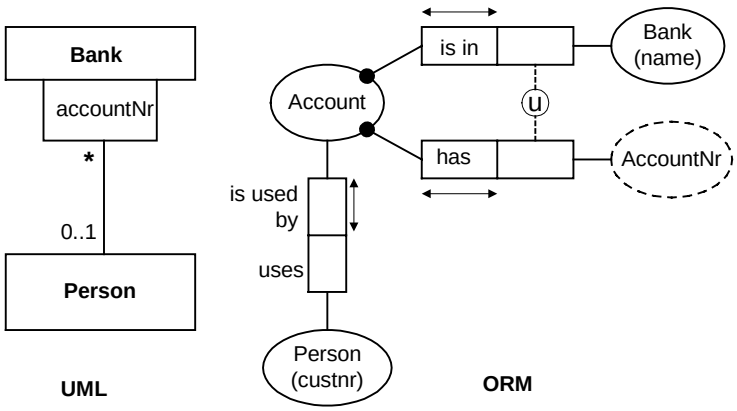

Figure 4: Qualified association in UML, and co-referenced object type in ORM

Here each bank account is used by at most one person, and each person may use many accounts. In the UML model, the attribute accountNr is used as a qualifier on the association, effectively partitioning each bank into different accounts. In the ORM model, an Account object type is explicitly introduced, and is referenced by combining its bank with its (local) account number. The circled "u" may be replaced by a "P" to indicate primary reference.

The UML notation is not only less clear, but less adaptable. For example, if we now want to record something about the account (e.g. its balance) we need to introduce an Account class, and the connection to accountNr is unclear. For a similar example, see [2] (p. 92, Fig. 5.10), where product is used with Order to qualify an order line association: again, this is unfortunate, since we would normally introduce a Product class to record data about products, and relevant connections are then lost. As a complicated example of this deficiency, see [1] (p. 51, Fig. 3.14) where the semantic connection between Node and nodeName is lost. The problem can be solved in UML by using an association class instead, though this is not always natural. The use of qualified associations in UML is hard to motivate, but may be partly explained by their ability to capture some compound uniqueness constraints in the standard graphic notation, rather than relying on non-standard textual notations (such as our {U*n*} notation).

ORM's concept of an external uniqueness constraint that may be applied to a set of roles in one or more predicates provides a simple, uniform way to capture all of UML's qualified associations and unique attribute combinations, as well as other cases not expressible in UML graphical notation (e.g. cases with *m:n* predicates or long join paths). As always, the ORM notation has the further advantage of facilitating validation through verbalization and multiple instantiation.

Or-associations

UML uses the term *or-association* for one of many associations stemming from a class, where at any given time each class member may participate in at most one of these associations. To indicate this, UML uses what it calls an *or-constraint* between the associations, attaching the constraint string "{or}" to a dotted line connecting the relevant associations. Figure 5 is based on an example from the UML standard. For simplicity, reference schemes and other constraints are omitted.

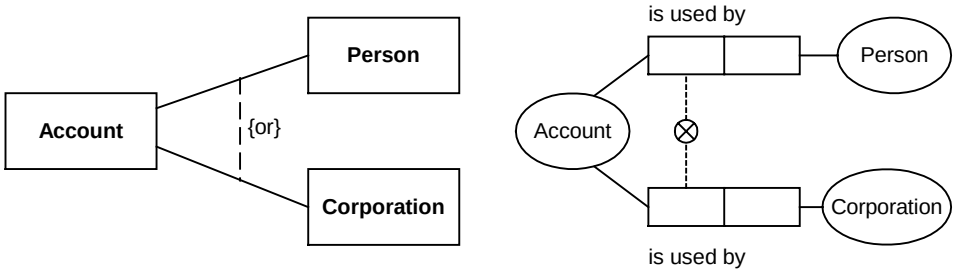

Figure 5: No account is used by both a person and a corporation

UML's use of "or" for this constraint is confusing because it is used in an *exclusive* instead of inclusive sense (in contrast to virtually all computer languages). An alternative such as "xor" would be less ambiguous, and hence safer, even if artificial.[1] There is another possible confusion arising from the standard document itself. A literal reading of the latest version (1.2) of the UML standard indicates that the constraint simply means that an account is used by *at most one* of the two choices (person or corporation). However, some authors argue that its use in OMT (a precursor of UML) means each account must be used by *exactly one* of these choices ([1], p. 50). If this is the case, the constraint means that the disjunction is both exclusive and mandatory. Given that the lengthy UML standard currently contains a number of ambiguities and inconsistencies, I'm not sure which reading is actually correct. For now, I'll assume that the weaker reading (exclusive) is correct. In this case, the constraint is captured in ORM by an *exclusion constraint*, shown by connecting "⊗" by dotted lines to the relevant roles (see above figure). If the stronger reading is correct[2], a disjunctive mandatory role constraint needs to be added as well (see Part 1).

UML or-constraints apply between single roles. The standard seems to imply that these roles must belong to different associations. If so, UML cannot use an or-constraint between roles of a ring fact type (e.g. between the husband and wife roles of a marriage association). ORM exclusion constraints cover this case, as well as many other cases not expressible in UML graphic notation. ORM exclusion constraints may apply to any set of compatible *role-sequences*, by connecting "⊗" by dotted lines to the relevant role-sequences. As a trivial example, consider the difference between the following two constraints: no person both wrote and reviewed a book; no person wrote and reviewed the same book. ORM clearly distinguishes these by noting the precise arguments of the constraint (see Figure 6).

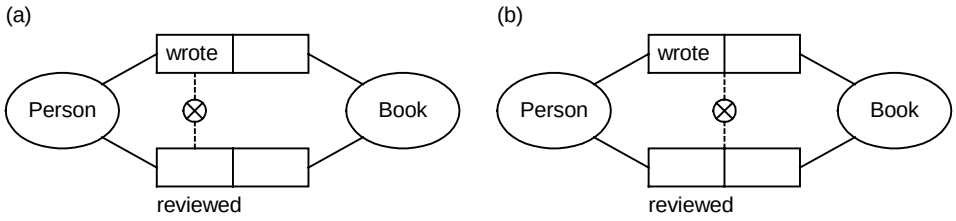

Figure 6: (a) no person wrote and reviewed; (b) no person wrote and reviewed *the same* book

The pair-exclusion constraint in Figure 6(b) can be expressed in UML by adding a comment box that includes a textual constraint written in some language (e.g. OCL), and connecting this by dotted lines to the two associations. However this notation is both cluttered and non-standard (since UML allows users to pick their own language to write textual constraints).

UML has no graphic notation for exclusion between attributes, or between attributes and associations. In Figure 7(a), the unary predicate must be modeled in UML as a

[1] After the original publication of this article, UML 1.3 replaced the "or" constraint notation by "xor"
[2] In UML 1.3, the xor constraint was clarified to mean the stronger reading, i.e. "exactly one"

Boolean attribute, and the contract predicate would probably be modeled as a contractDate attribute. In Figure 7(b), the completion predicate would be modeled in UML as a completionDate attribute of the Project class, while resource usage would normally be modeled as an association between Project and Resource classes. If we made these modeling choices in UML, we must resort to non-standard notations or textual constraints to add exclusion constraints between attributes (a) or between an attribute and association (b). There are alternative ways to model these cases in UML (e.g. using subtypes) that offer more chance to capture the constraints graphically, but it is clear that UML's or-constraint is far less expressive than ORM's exclusion constraint.

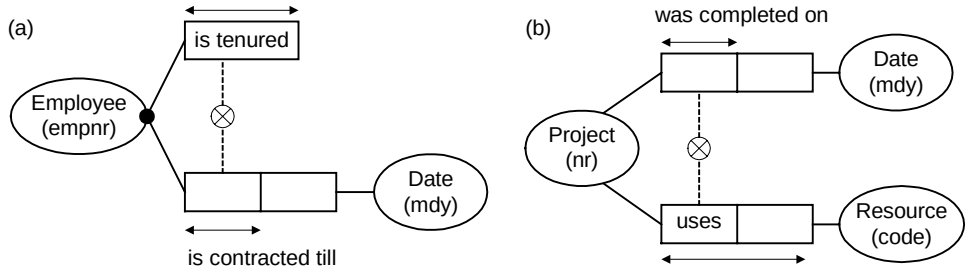

Figure 7: A comparative summary of data structure concepts

We've now covered essentially all the high level data structures that can be specified in graphic notation on ORM data models and UML class diagrams. As we discuss in a later issue, collection types may also be specified in both ORM and UML via textual annotations. Table 1 summarizes the differences between the two modeling methods with respect to terms and graphic (not textual) notations for data instances and structures. We still have several constraints to discuss, so will delay provision of a summary table about constraints till a later issue.

Table 1: Basic correspondence between ORM and UML conceptual data concepts

Data instances/structures	
ORM	***UML***
Entity	Object
Value	Data value
Object	Object or Data value
Entity type	Class
Value type	Data type
Object type	Class or Data type
— { use relationship type }	Attribute
Unary relationship type	— { use Boolean attribute }
2^+-ary relationship type	Association
2^+-ary relationship instance	Link
Nested object type	Association class
Co-reference	Qualified association §

§ = incomplete coverage of corresponding concept

Later issues

Later issues will discuss more advanced graphic constraints in both ORM and UML (subset, equality, aggregation, ring, join etc.), subtyping, derivation rules and queries.

References

1. Blaha, M. & Premerlani, W. 1998, Object-Oriented Modeling and Design for Database Applications, Prentice Hall, New Jersey.
2. Fowler, M. with Scott, K. 1997, UML Distilled, Addison-Wesley.
3. Halpin, T. 1995, Conceptual Schema and Relational Database Design, 2nd edn, Prentice Hall Australia.
4. OMG-UML v1.2, OMG UML Revision Task Force website, http://uml.systemhouse.mci.com/.