UML data models from an ORM perspective: Part 8

by Dr. Terry Halpin

Director of Database Strategy, Visio Corporation

This paper first appeared in the April 1999 issue of the Journal of Conceptual Modeling*, published by InConcept.*

This paper is the eighth in a series of articles examining data modeling in the Unified Modeling Language (UML) from the perspective of Object Role Modeling (ORM). Part 1 discussed historical background, design criteria for modeling languages, object reference and single-valued attributes. Part 2 covered 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, as well as how associations may be instantiated. Part 4 contrasted ORM nesting with UML association classes, ORM co-referencing with UML qualified associations, and ORM exclusion constraints with UML or-constraints. Part 5 discussed subset and equality constraints. Part 6 discussed subtyping. Part 7 discussed value, ring and join constraints. Part 8 covers some recent updates to the UML standard, then discusses aggregation.

Updates to the UML standard

Recently Visio became a member of the Object Management Group (OMG), and began participating in the ongoing work to refine the UML standard. Within the OMG, the UML standard is the responsibility of the Analysis and Design Task Force (ADTF, formerly OOA&DTF), chaired by Jim Odell and Cris Kobryn. Minor changes to the UML standard that lead to point releases (e.g. 1.1, 1.2, 1.3) are managed by a subgroup of the ADTF known as the UML RTF (Revision Task Force), chaired by Cris Kobryn. The latest release of UML (version 1.2) is fully supported by Visio Enterprise, including all nine diagram types. Currently, the UML RTF is working on a draft of version 1.3, and some further point releases might be considered later (e.g. version 1.4). The next major release (2.0) is not expected to be forthcoming from the ADTF for quite some time (e.g. late 2001). As a result of recent email discussions and meetings of the UML RTF team, several revisions to the UML 1.3 draft have been made, including two that I will comment on here, since they relate to issues discussed in earlier articles in this series.

The “{or}” constraint discussed in Part 4 of this series has been renamed “{xor}” (short for “exclusive or”), and has been redefined to mean *exactly one* of the association roles is chosen. This means it is equivalent to ORM’s exclusive-or constraint, which is a combination of a disjunctive mandatory role constraint and an exclusion constraint. For example, consider the following constraint

(1) each Vehicle is either leased from a Company or was purchased from a Company, but not both.

In ORM, this may be expressed by the following two constraints:

(2) **each** Vehicle is leased from **a** Company **or** was purchased from **a** Company.
(3) **no** Vehicle is leased from **a** Company **and** was purchased from **a** Company.

Constraint (2) is a disjunctive mandatory role constraint, shown as a black dot on the object type connected to the two roles, or by a circled black dot “⦿” connected to the roles. Constraint (3) is an exclusion constraint, shown as a circled ex “⊗” connecting the two roles. The constraints are orthogonal, and may be shown either separately as in Figure 1(a) or by combining the two symbols as in Figure 1 (b).

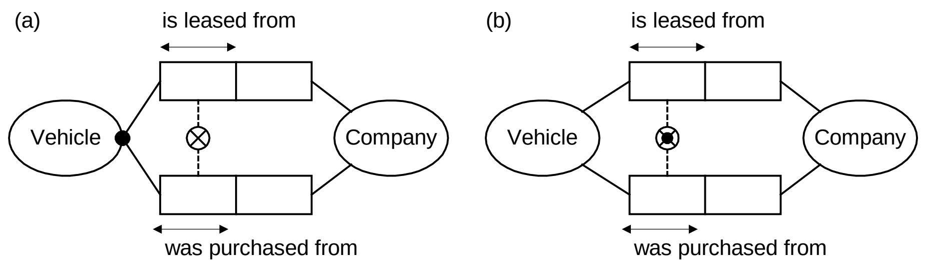

Figure 1: Exclusive-or constraint depicted in ORM using (a) separate or (b) combined symbols

In UML, the constraint is displayed by connecting the relevant association-ends (roles in ORM) by a dashed line, labeled “{xor}” (see Figure 2). Although the current wording of the UML standard describes the constraint as applying to a set of associations, we need to apply the constraint to a set of association-ends to avoid ambiguity in cases like this with multiple common classes. Visually this could be shown by attaching the dashed lines near the relevant ends of the associations, as we have done here.

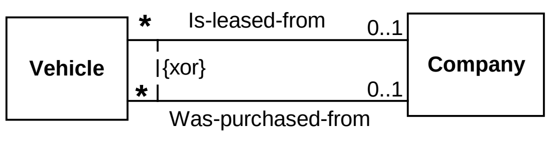

Figure 2: Exclusive-or constraint depicted in UML

As discussed in previous issues, UML has no symbols for exclusion or disjunctive mandatory role constraints. If ever UML symbols for these constraints are considered, then “{x}” and “{or}” respectively seem appropriate—this choice also exposes the composite nature of “{xor}”. Even if such a proposal were accepted as a UML extension, this would capture only a fragment of ORM’s expressive power in this area—recall that ORM’s exclusion constraint applies not just to a set of roles, but a set of role-sequences, and hence is far more general than the kind of case considered here. Moreover, ORM roles include unary predicates, and ORM needs no additional notations to constrain attributes.

The second proposed revision to UML concerns the semantics of the “{complete}” constraint for subtyping. This constraint, discussed in Part 6, was formerly described as indicating that the modeler intended to add no more subtypes. This weak notion of completeness does not entail that the constrained subtypes collectively exhaust the supertype, but this latter notion is far more useful in practice and is called a totality constraint in ORM. Although typically implied by other constraints, a totality constraint may be explicitly depicted in ORM by connecting the mandatory symbol “⦿” to the relevant subtype links (it is mandatory for each instance of the supertype to be an instance of at least one of the subtypes). Hence the supertype equals the union of the constrained subtypes. Recall that a type is the set of all possible instances, while a population is the set of current instances. The practical way to enforce the constraint is to check that for each state of the database, the population of the supertype equals the union of the populations of the constrained subtypes. At the UML RTF meeting in March it was agreed that the UML notion of subtype completeness would be redefined as this set-theoretic notion, thus making it equivalent to ORM’s subtype totality constraint. With this understanding, the ORM and UML schemas in Figure 3 are equivalent.

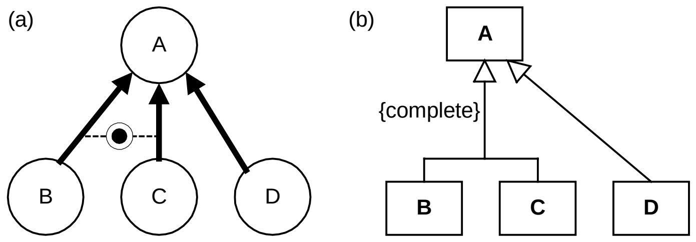

Figure 3: The subtype totality constraint A = B ∪ C expressed in (a) ORM and (b) UML.

Aggregation

In UML, the term “aggregation” is used to describe a whole/part relationship. For example, a team of people is an aggregate of its members, so this membership may be modeled as an aggregation association between Team and Person. Several different forms of aggregation might be distinguished in real world cases. For example, Jim Odell and Conrad Bock discuss the following six varieties of aggregation: component-integral; material-object; portion-object; place-area; member-bunch; and member-partnership [4, 5]. Currently, UML associations are classified into one of three kinds: ordinary association (no aggregation); shared (or simple) aggregation; composite (or strong) aggregation. Hence UML version 1.x recognizes only two varieties of aggregation: shared and composite. Although early planning for UML version 2.x foreshadows further kinds of aggregation being introduced, we confine our attention here to shared and composite aggregation. Some versions of ER include an aggregation symbol (typically only one kind). ORM, as well as many versions of ER, includes no special symbols for aggregation.

These different stances with respect to aggregation are somewhat reminiscent of the different modeling positions with respect to null values. Although over twenty kinds of null have been distinguished in the literature, the relational model recognizes only one kind of null, Codd's version 2 of the relational model proposes two kinds of null, and ORM argues that nulls have no place in base conceptual models (because all its base facts are elementary). But let's return to the topic at hand.

Shared aggregation is denoted in UML as a binary association, with a hollow diamond at the "whole" or "aggregate" end of the association. Composition (composite aggregation) is depicted with a filled diamond. For example, Figure 4 depicts a composition association from Club to Team, and a shared aggregation association from Team to Person.

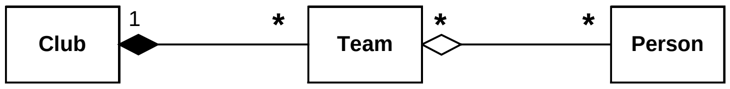

Figure 4: Composition (composite aggregation) and shared aggregation in UML

In ORM, this situation would be modeled as shown in Figure 5. As we see, ORM has no special notation for aggregation. Does Figure 4 convey any extra semantics, not captured in Figure 5? At the conceptual level, it is doubtful whether there is any additional useful semantics. At the implementation level however, there is additional semantics. Let's discuss this in more detail.

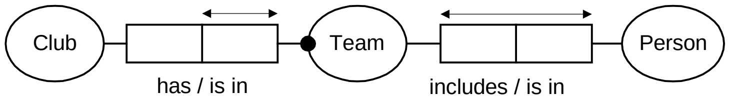

Figure 5: The Figure 4 example modeled in ORM

The UML standard declares that "both kinds of aggregation define a transitive … relationship" [6]. The use of "transitive" here is somewhat misleading, since it refers to indirect aggregation associations rather than base aggregation associations. For example, if Club is an aggregate of Team, and Team is an aggregate of Person, it follows that Club is an aggregate of Person. However if we wanted to discuss this result, it should be exposed as a derived association. In UML, derived associations are indicated by prefixing their names with "/". The derivation rule can be expressed as a constraint, either connected to the association by a dependency arrow, or simply placed beside the association as in Figure 6.

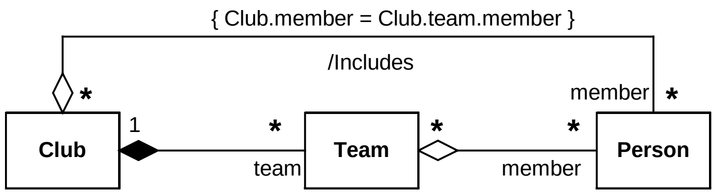

Figure 6: A derived aggregation in UML

In ORM, derived fact types may be diagrammed by marking them with an asterisk, and derivation rules may be specified in an ORM textual language such as ConQuer (see Figure 7). In many cases, derivation rules may also be diagrammed as a join-subset or join-equality constraint. As this example illustrates, the derived transitivity of aggregations can be captured in ORM without needing a special notation for aggregation.

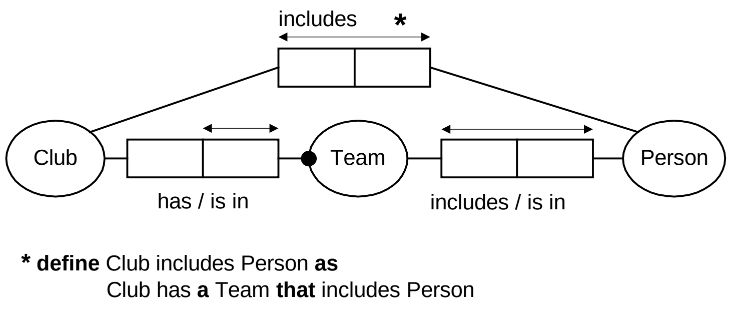
includes *
Club
has / is in
Team
includes / is in
Person
* **define** Club includes Person **as**
Club has **a** Team **that** includes Person

Figure 7: The derived aggregation of Figure 6 modeled in ORM

More fully, the UML standard declares that “both kinds of aggregation define a transitive, antisymmetric relationship (i.e. the instances from a directed, non-cyclic graph)” [6]. Recall that a relation *R* is antisymmetric if and only if, for all *x* and y, if x is not equal to *y* then *xRy* implies that *yRx*. It would have been better to simply state that paths of aggregations must be acyclic. At any rate, this rule is designed to stop errors such as that shown in Figure 8. If a person is part of a team, and a team is part of a club, it doesn’t make sense to say that a club is part of a person.

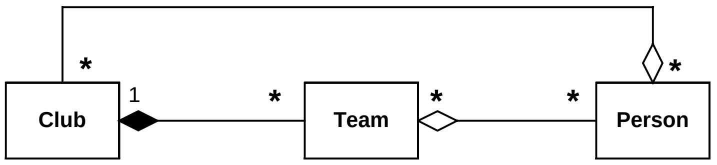

Figure 8: Illegal UML model. Aggregations should not form a cycle.

Since ORM does not specify whether an association is an aggregation, illegal diagrams like this can’t occur in ORM. Of course, it is possible for an ORM modeler to make a silly mistake by adding an association such as Club is part of Person, where “is part of” was informally understood in the aggregation sense, and this would not be formally detectable. But avoidance of such a bizarre occurrence doesn’t seem to be a compelling reason to add aggregation to ORM’s formal notation. There are plenty of associations between Club and Person that do make sense, and plenty that don’t. In some cases however, it is important to assert constraints such as acyclicity, and this is handled in ORM by ring constraints (see Part 7).

Composition does add some important semantics to shared aggregation. To begin with, it requires that each part belongs to at most one whole at a time. In ORM, this is captured by adding a uniqueness constraint to the role played by the part (e.g. see the role played by Team in Figure 5). In UML, the multiplicity at the whole end of the association must be 1 or 0..1. If the multiplicity is 1 (as in Figure 4), the role played by the part is both unique and mandatory (as in Figure 5). As an example where the multiplicity is 0..1 (i.e. where a part optionally belongs to a whole), consider the ring fact type of Figure 9: Package contains Package. Here “contains” is used in the sense of “directly contains”. The UML standard notes that “composition instances form a strict tree (or rather a forest)” [6]. This strengthening from directed acyclic graph to tree is an immediate consequence of the functional nature of the association (each part belongs to at most one whole), and hence ORM requires no additional notation for this. In this example, the ORM model explicitly includes an acyclic constraint. Note that this direct containment association is intransitive by implication (acyclicity implies irreflexivity, and any functional, irreflexive association is intransitive).

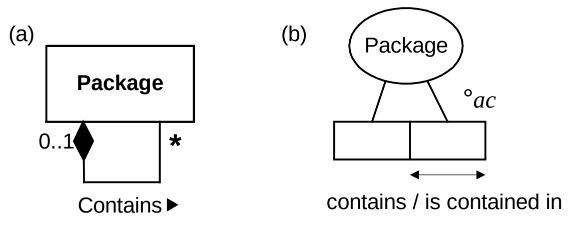

Figure 9: Direct containment modeled in (a) UML and (b) ORM

UML allows some alternative notations for aggregation. If a class is an aggregate of more than one class, the association lines may be shown joined to a single diamond (see Figure 10(a)). For composition, the part classes may be shown nested inside the whole by using role names, and multiplicities of components may be shown in the top right hand corners (see Figure 10(b)).

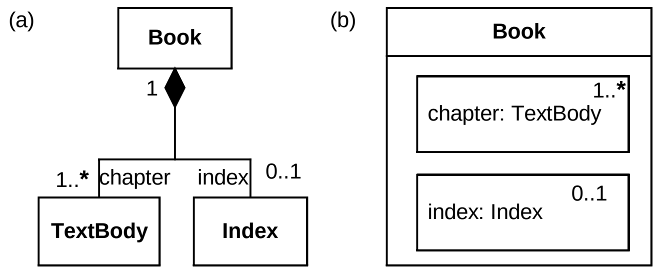

Figure 10: Alternative UML notations for aggregation

Some authors list kinds of association that are easily confused with aggregation but should not be modeled as such (e.g. topological inclusion, classification inclusion, attribution, attachment and ownership [4, 5]). For example, Finger belongs to Hand is an aggregation, but Ring belongs to Finger is not. There is some disagreement among authors about what should be included on this list. For example, attribution is treated by some as a special case of aggregation (a composition between a class and the classes of its attributes) [7]. My own viewpoint is that for conceptual modeling purposes, agonizing over such distinctions doesn't seem to be worth the trouble. This position seems to be taken by some other authors. For example, [7, p. 148] argues that "Aggregation conveys the thought that the aggregate is inherently the sum of its parts. In fact, the only real semantics that it adds to association is the constraint that chains of aggregate links may not form cycles … Some authors have distinguished several kinds of aggregation, but the distinctions are fairly subtle and probably unnecessary for general modeling".

Indeed there seems little justification for introducing the notion of aggregation at all as a separate concept at the conceptual level. There are plenty of other distinctions (apart from aggregation) we could make about associations, but we don't feel compelled to do so. At the implementation level however, composite aggregation does add important semantics. "A composite implies propagation semantics … For example, if the whole is copied or deleted, then so are the parts as well" [6]. Clearly this dynamic semantics has nothing to do with a conceptual view of the domain area, and it would be unreasonable to introduce this notion when validating the business model with the subject matter expert. However, once a decision is made to implement the conceptual model in an object-oriented system, it is important to capture this semantics. One way of doing this would be to convert a conceptual ORM model to a UML model, and then add aggregation at that stage.

Obviously there are different stances one could take about how, if at all, aggregation should be included in the conceptual modeling phase. My position is one of many. You can decide what's best for you.

Later issues

Later issues will discuss default values, changeability settings, derived data, derivation rules and queries in ORM and UML.

References

1. Booch, G., Rumbaugh, J. & Jacobson, I. 1999, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading MA, USA.
2. Blaha, M. & Premerlani, W. 1998, *Object-Oriented Modeling and Design for Database Applications,* Prentice Hall, New Jersey.
3. Halpin, T. 1995, *Conceptual Schema and Relational Database Design, 2nd edn (revised 1999)*, WytLytPub, Bellevue WA, USA.
4. Martin, J. & Odell, J. 1998, *Object-Oriented Methods: a Foundation, UML edn*, Prentice Hall, Upper Saddle River, New Jersey. { Ch. 18 discusses aggregation }
5. Odell, J. 1998, *Advanced Object-Oriented Analysis & Design using UML*, Cambridge University Press, & SIGS Books, New York. { Part V (pp. 137-65) discusses aggregation }
6. OMG-UML 1.3 draft, OMG UML Revision Task Force website, http://uml.systemhouse.mci.com/.
7. Rumbaugh, J., Jacobson, I. & Booch, G. 1999, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading MA, USA.