UML data models from an ORM perspective: Part 5

by Dr. Terry Halpin

Director of Database Strategy, Visio Corporation

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

This paper is the fifth 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 discusses ORM subset and equality constraints, and how these may be specified in UML.

Subset constraints

ORM allows a *subset constraint* to be graphically specified between any pair of compatible role-sequences by connecting them with a dashed arrow. This declares that the population of the source role sequence must always be a subset of the target role sequence (the one hit by the arrow-head). Each sequence may comprise one or more roles. These constraints have corresponding verbalizations. For example, in Figure 1 the subset constraint between single roles indicates that students have second names only if they have first names. The other subset constraint is between Student-Course role-pairs, and declares that students may pass tests in a course only if they have enrolled in that course. Since the role of having a surname is mandatory for Student, subset constraints to it from all the other student roles are implied (and hence not shown).

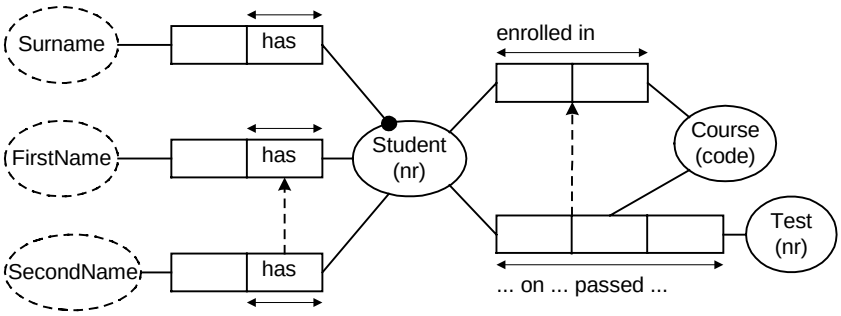

Figure 1: Subset constraints in ORM

As an extension mechanism, UML allows subset constraints to be specified between *whole associations* by attaching the constraint label "{subset}" next to a dashed arrow between the associations. For example, the subset constraint in Figure 2 indicates that any person who chairs a committee must be a member of that committee.

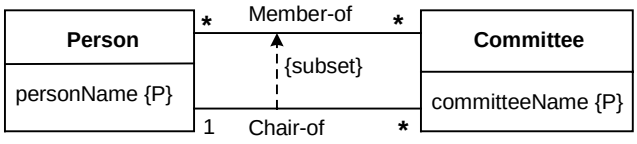

Figure 2: A subset constraint in UML

However UML does not provide a graphic notation for subset constraints between single roles or between parts of associations. Hence if a UML diagram depicted the relationship types in Figure 1 as associations, it would not be able to capture the subset constraints graphically. Of course, other options are available in UML. For instance, if we model surname, firstName and secondName as attributes of Person we can express the single-role subset constraint by attaching a comment including the following textual constraint (see Figure 3):

Student.firstName **is not null or** Student.secondName **is null**

Although this does capture the subset constraint, it is at a lower level than ORM's graphic or verbalized form, and is basically the same as the check clause generated by VisioModeler when mapping the constraint down to a relational implementation.

One way in UML to capture the pair-subset constraint from Figure 1 is to transform the ternary into a binary association with a subset constraint to the enrollment association, and with a binary association to Test. A better solution is to use ORM's overlap algorithm [2, p. 349] to objectify the enrollment association and associate this with Test. As discussed in Part 4, the equivalent UML action is to make a class out of enrollment (see Figure 3). Although in this situation an association class provides a good way to cater for a compound subset constraint, sometimes this nesting transformation leads to a very

unnatural view of the world. Ideally the modeler should be able to view the world naturally, while having optimization transformations that lessen the clarity of the conceptual schema performed under the covers.

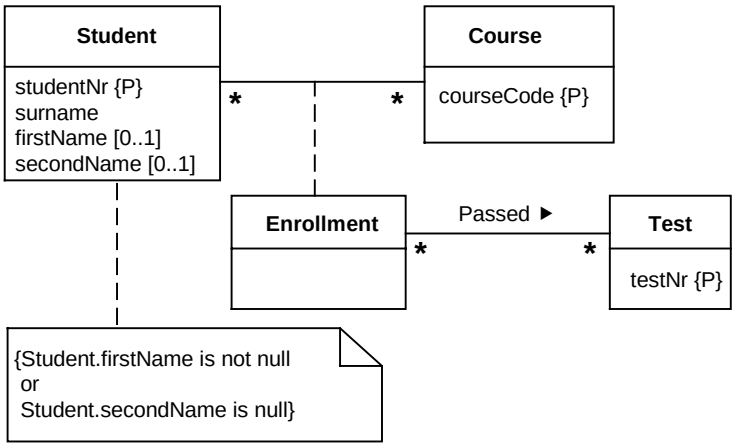

Figure 3: A UML version of the ORM schema in Figure 1

ORM has a mature formalization, including a rigorous theory of such topics as schema consistency, equivalence and implication. UML was only recently standardized, and is undergoing revisions. Since formal guidelines for working with UML are somewhat immature, extra care is needed to avoid logical problems. As a simple example, look back at Figure 2, which comes from the current draft of the UML 1.2 standard [3], with reference schemes added. Do you spot anything confusing about the constraints?

You probably noticed the problem. The multiplicity constraint of 1 on the chair association indicates that each committee must have at least one chair. The subset constraint tells us that a chair of a committee must also be a member of that committee. Taken together, these constraints imply that each committee must have a member. Hence one would expect to see a multiplicity constraint of "1..*" (one or more) on the Person end of the membership association. However we see a constraint of "*" (zero or more) instead, which at best is very misleading.

An ORM schema equivalent to Figure 2 is shown in Figure 4(a). The implied mandatory role constraint (each Committee includes at least one Person) is added explicitly in Figure 4(b). Which representation do you prefer?

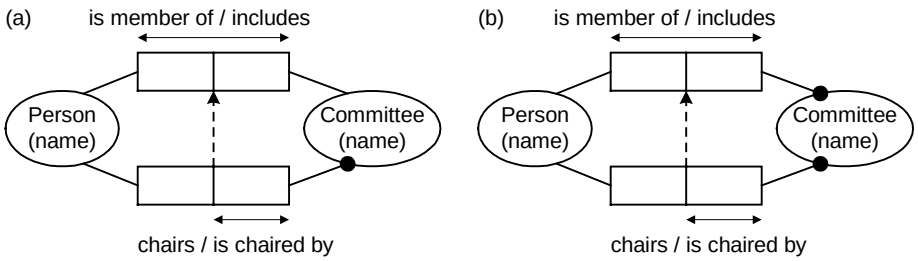

Figure 4: Schema (a) has an implied mandatory role constraint, shown explicitly in (b)

Although display options for implied constraints may sometimes be a matter of taste, practical experience has shown that in cases like this is better to show implied constraints explicitly rather than expect modelers or domain experts participating in the modeling process to figure them out for themselves. If you enter the schema of Figure 4(a) in VisioModeler, and attempt to build the logical dictionary, the tool will detect the misleading nature of the constraint pattern and ask you to resolve the problem. Human interaction is the best policy here, since there is more than one possible mistake (e.g. is the subset constraint correct or is the optional role correct?). Clicking on the error message throws you back into the conceptual schema with a red arrow highlighting the problem for you to fix (e.g. add the mandatory role constraint).

Note that if the schema of Figure 4(b) is mapped to a relational database it generates a referential cycle, since the mandatory fact types for Committee map to different tables (so each committee must appear in both tables). Referential cycles can be messy to work with, so VisioModeler warns you about this, but still generates the code to cope with it. The relational schema diagram generated by VisioModeler is shown in Figure 5 (the arrows show the foreign key references, one simple and one composite, that correspond to the subset constraints).

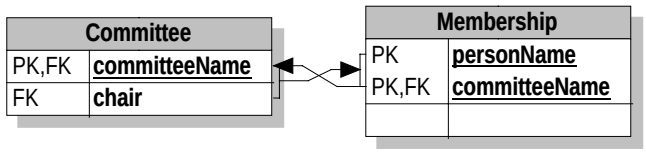

Figure 5: The relational schema mapped from the schema of Figure 4(b)

As another constraint example in UML, consider Figure 6, which is the UML version of an OMT diagram used in [1, p. 68] to illustrate a subset constraint between associations. See if you can spot any problems with the constraints.

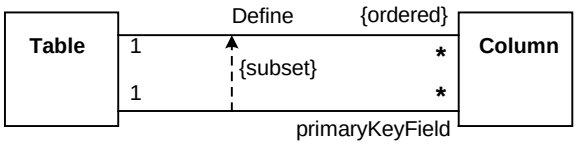

Figure 6: Spot anything wrong?

There are some fairly obvious problems with the multiplicity constraints. For example, the “1” on the primary key association should be “0..1” (not all columns belong to primary keys), and the “*” on the define association should presumably be “1..*” (unless we allow tables to have no columns). Assuming that tables and columns are identified by oids or artificial identifiers, the subset constraint makes sense, but the model is arguably sub-optimal since the PK association and subset constraint could be replaced by a boolean isaPKfield attribute on Column.

From an ORM perspective, heuristics lead us to initially model the situation using natural reference schemes as shown in Figure 7. Here ColName denotes the local name of the column in the table, and we have simplified reality by assuming tables may be identified just by their name. As seen by the external uniqueness constraints, two natural reference schemes for Column suggest themselves (name plus table, or position plus table). We can choose one of these as primary, or instead introduce an artificial identifier. The unary predicate indicates whether a column is, or is part of, a primary key. If desired, we could derive the association "Column is a primary key field of Table" from the path: "Column is in Table **and** Column isaPKcol" (the subset constraint from the previous model is then implied).

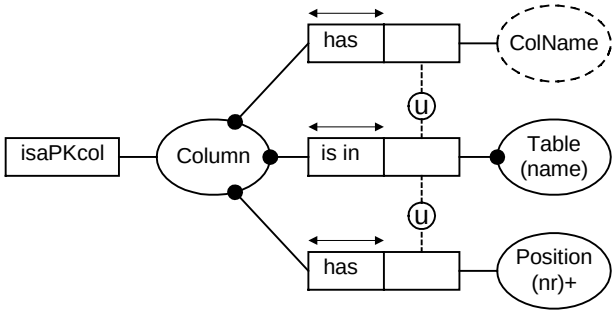

Figure 7

What is interesting about this example is not that the authors of the earlier model may have made some trivial errors with constraints (I've made slips of the pen like that in some of my book examples too), but rather the difference in modeling approaches. Most OMT and UML modelers seem to assume that oids will be used as identifiers in their initial modeling, whereas ORM modelers like to expose natural reference schemes right from the start, and populate their fact types accordingly. These different approaches often lead to different solutions. The main thing is to first come up with a solution that is natural and understandable to the domain expert, because here is where the most critical phase of model validation should take place. Once a correct model has been determined, optimization guidelines can be used to enhance it.

One other feature of the example is worth mentioning. The UML solution in Figure 6 uses the annotation "{ordered}" to indicate that a table is comprised of an *ordered set* (i.e. a sequence with no duplicates) of columns. In the ORM community, a debate has been going on for several years on the best way to deal with constructors (e.g. set, bag, sequence, unique sequence) at the conceptual level. My view (and that of several other ORM researchers) is that such constructors should not appear in the base conceptual model. Hence the use of Position in Figure 7 to convey column order (the uniqueness of the order is conveyed by the uniqueness constraint on Column-has-Position). Keeping fact types elementary has so many advantages (e.g. validation, constraint expression, flexibility and simplicity) that it seems best to relegate constructors to derived views. I may have more to say about this in a later article.

Equality constraints

In ORM, an *equality constraint* between two compatible role sequences is shorthand for two subset constraints (one in either direction), and is shown as a double-headed arrow. Such a constraint indicates that the populations of the role-sequences must always be equal. If two roles played by an object type are mandatory, then an equality constraint between them is implied (and hence not shown).

As a simple example of an equality constraint, consider Figure 8. Here the equality constraint indicates that if a patient's systolic blood pressure is measured, so is his/her diastolic blood pressure (and vice versa). In other words, either both measurements are taken, or neither. This kind of constraint is fairly common. Less common are equality constraints between sequences of two or more roles.

UML has no graphic notation for equality constraints. For whole associations we could use two separate subset constraints, but this would be very messy. We could add a new notation, using "{equality}" besides a dashed arrow between the associations, but this notation would be unintuitive, since the direction of the arrow would have no significance (unlike the subset case).

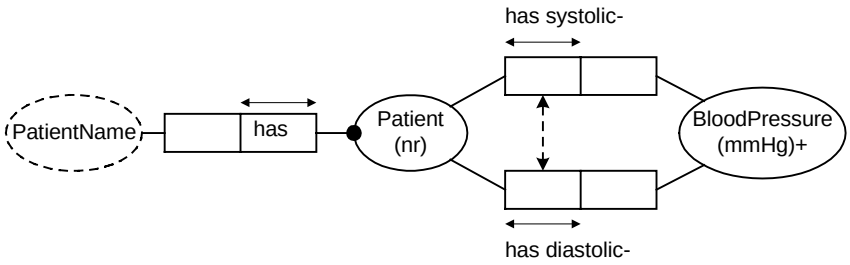

Figure 8: A simple equality constraint

In general, equality constraints in UML would normally be specified as textual constraints (in braced comments). For our current example, the two blood pressure readings would normally be modeled as attributes of patient, and hence a textual constraint is attached to the Patient class as shown in Figure 9. Like UML textual subset constraints, this is awkward compared to the corresponding ORM constraint (graphic or verbalized).

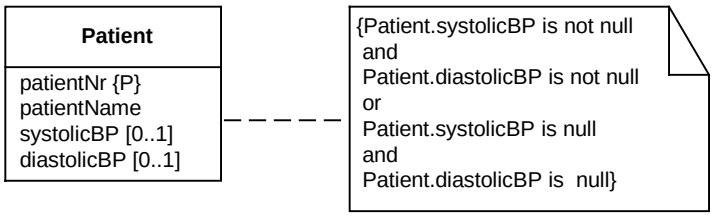

Figure 9: A simple subset constraint in UML

Subset and equality constraints enable various classes of schema transformations to be stated in their most general form, and ORM's more general support for these constraints allows more transformations to be easily visualized. For example, Figure 10 depicts equivalence PSG2 [2, p. 331].

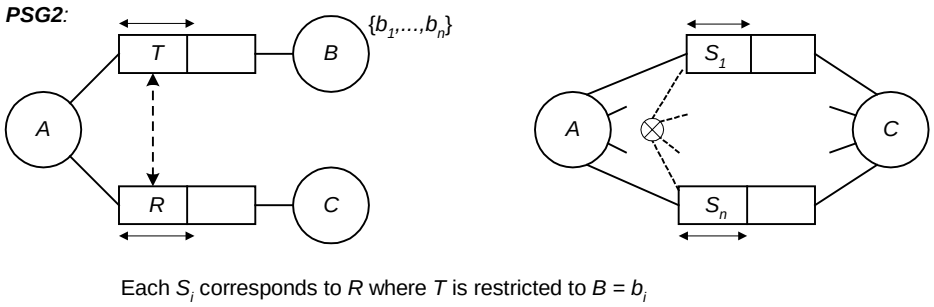

Figure 10: A basic schema equivalence in ORM

Later issues

Later issues will discuss other advanced graphic constraints in ORM and UML (join, ring, aggregation), 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. Halpin, T. 1995, *Conceptual Schema and Relational Database Design, 2nd edn*, Prentice Hall Australia.
3. OMG-UML v1.2, OMG UML Revision Task Force website, http://uml.systemhouse.mci.com/.