UML data models from an ORM perspective: Part 7

by Dr. Terry Halpin

Director of Database Strategy, Visio Corporation

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

This paper is the seventh 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 ORM subset and equality constraints, and how to specify these in UML. Part 6 discussed subtyping. Part 7 discusses some other graphic constraints (value, ring and join constraints).

Value constraints

An ORM *value constraint* restricts the population of a value type to a finite set of values specified either in full (*enumeration*), by start and end values (*range*), or some combination of both (*mixture*). The values themselves are primitive data values, typically character strings or numbers. The constraint is shown by declaring the possible values in braces besides either the value type, or an entity type with a reference mode. In the latter case, the constraint is understood to apply to the implicit value type. For example, in Figure 1 the constraints apply to Sexcode, RatingNr and SQLchar.

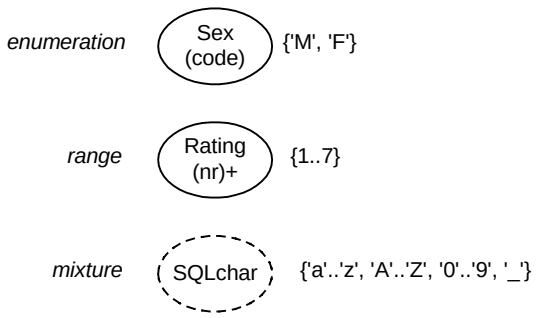

Figure 1: Value constraints in ORM

In UML, enumeration types may be modeled as classes, stereotyped as enumerations, with their values listed (somewhat unintuitively) as attributes. Ranges and mixtures may be specified by declaring a textual constraint in braces, using any formal or informal language. For example, see Figure 2.

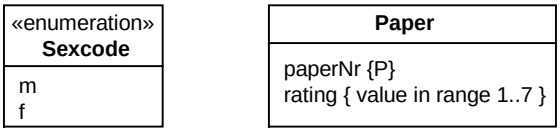

Figure 2: Data value restrictions declared as enumerations or textual constraints in UML

Value constraints other than enumeration, range and mixture may be declared in either ORM or UML as textual constraints, e.g. {committeeSize must be an odd number}. For further UML examples, see [5, pp. 236, 268].

Ring constraints

A ring fact type has at least two roles played by the same object type (either directly, or indirectly via a supertype). ORM allows various *ring constraints* to be applied to such role-pairs. For example, in Figure 3, the isParentOf association is declared to be acyclic (^{O}ac) and intransitive (^{O}it). Here "acyclic" means nobody can be one of his/her own descendants. A satisfying population is shown in the fact table below the schema. In the population graph shown at the right of the figure, people are denoted by circular nodes containing the first letter of their name, and the directed arrows denote the "is parent of" relationship. The acyclic constraint means there can't be any cycles or loops in the graph.

In this example, "intransitive" means nobody is a parent of any of his/her grandchildren. In terms of the graph, it means we can't add any arrows that jump over one node to provide an alternate path to the target node. By default, ORM constraints are "*hard*", meaning no violation is permitted. If we did accept that incest might occur in the UoD, and wanted to record any cases of it, this intransitive constraint should be down-graded to a "*soft* constraint", where violations are accepted but other action is taken to minimize their occurrence (e.g. send message to police).

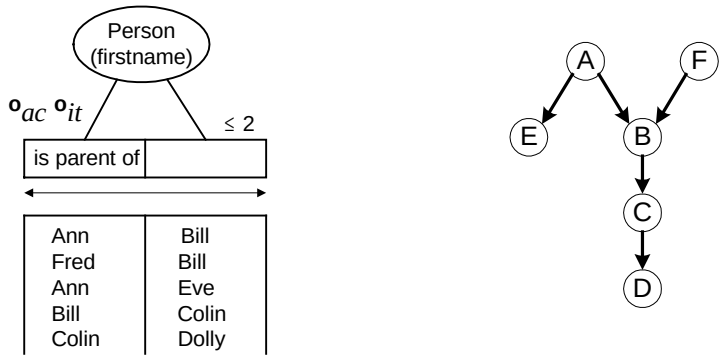

Figure 3: Some ring constraints in ORM

ORM provides six built-in ring constraints: antisymmetric (Oans), asymmetric (Oas), acyclic (Oac), irreflexive (Oir), intransitive (Oit), and symmetric (Osym). Because of their underlying logic, various implications exist between the constraints, and some combinations are impossible. To save you having to worry about these complexities, I designed the ring-constraint interface for VisioModeler so that you can't enter a ring constraint that is implied by, or incompatible with, one that you have already chosen (see Figure 4).

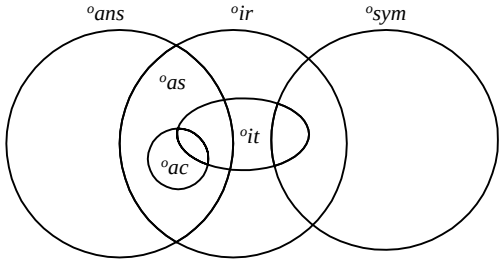

Figure 4: Ring constraint interface in VisioModeler

If you are mapping your model to a relational database, some ring constraints are very efficiently enforced. For example, irreflexivity typically maps to a simple check clause like "**check** (parent <> child)". On the other hand, some ring constraints can be very expensive (e.g. acyclicity). In this case, a conscious decision needs to be taken as to whether to have the constraint enforced at all by the system (e.g. in batch mode overnight) or to have users instructed that they are responsible for enforcing the constraint.

UML does not provide ring constraints built-in, so the modeler needs to specify these as a textual constraint in some chosen language. In UML, if a textual constraint applies to just one model element (e.g. an association path), it may be added in braces beside that element. For example, the ORM parenthood schema might be recast in UML as shown in Figure 5(a). It is the responsibility of the software tool (used to work with the diagram) to ensure the constraint is linked internally to the relevant model element, and to interpret any textual constraint expressions. If the tool cannot interpret the constraint, it should be placed inside a note, without braces, showing that it is merely a comment, and explicitly linked to the relevant model element, as shown in Figure 5(b).

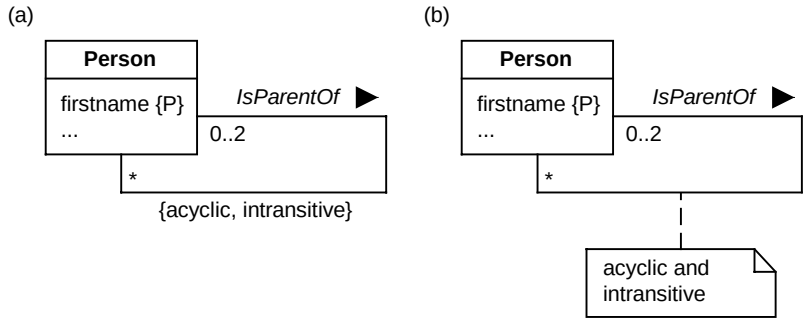

Figure 5: Ring constraints expressed in UML as (a) textual constraints and (b) comments

Join constraints

In ORM, a *join* constraint applies to one or more role sequences, at least one of which is projected from a *path* from one predicate through an object type to another predicate. The act of passing from one role through an object type to another role invokes a *conceptual join*, since the same object instance is asserted to play both the roles. The external uniqueness constraint (discussed in a earlier article) is actually a very simple case of this, in which there is just one argument. For example, in Figure 6 suppose we start at Employee, then follow the path to Date. This gives us the path: Employee was issued a ParkPermit that was issued on a Date. The "that" in this path expression asserts that the parking permit issued to the employee is the *same* one issued on the date. This identity claim is a conceptual join—like an equi-join in relational theory, except that it is over objects, not attribute values. In a later issue, we briefly discuss how such path expressions are used in ConQuer, an ORM conceptual query language. Now that the path is known, we project on the first and last roles (those played by Employee and Date) and assert uniqueness over this combination. In other words, a given employee on a given date can be issued at most one parking permit. This is the most fundamental way to understand external uniqueness constraints.

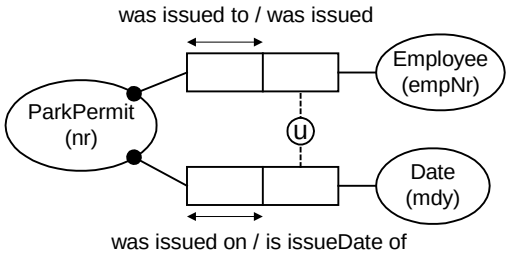

Figure 6: An external uniqueness constraint is a simple join constraint over one path

Role sequences featuring as arguments in set comparison constraints (subset, equality, exclusion) may also arise from projections over a join path. For example, in Figure 7, the subset constraint runs from the (Room, Facility) role-pair projected from the path: Room at a Time is used for an Activity that requires a Facility. This path includes a conceptual join on Activity. Since the subset constraint involves at least one join path, it is called a *join-subset constraint*. The constraint may be verbalized as: **if a** Room at **a** Time is used for **an** Activity **that** requires **a** Facility **then that** Room provides **that** Facility.

This example is based on a room scheduling application at a university with built-in facilities in various lecture and tutorial rooms (PA = Personal Address system, DP = Data Projection facility, INT = Internet access). Figure 7 includes a satisfying population for the three fact types.

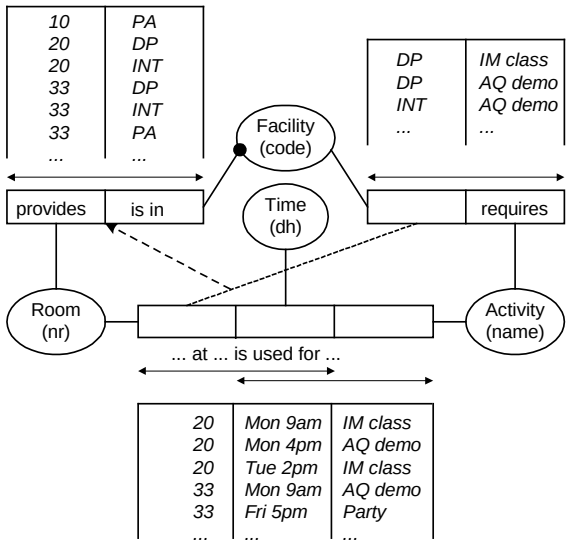

Figure 7: A join-subset constraint in ORM

Although join constraints arise frequently in real applications, UML has no graphic symbol for them. Nevertheless, they may be declared on UML diagrams by writing a textual constraint or comment in a note (dog-eared rectangle), attached by a dashed line to the model elements involved (here, three associations). Figure 8 uses a comment.

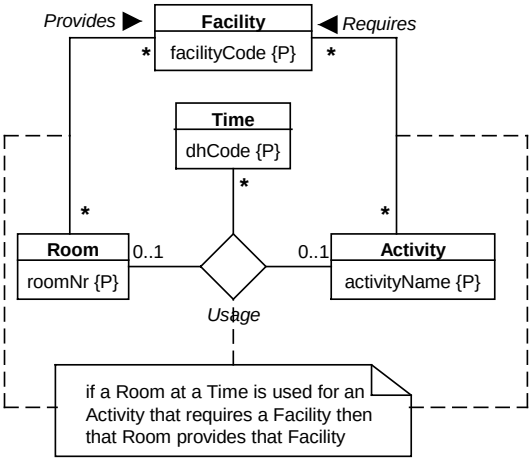

Figure 8: Join constraint specified as a comment in UML

Figure 7 again illustrates how ORM facilitates validation constraints via sample populations. The UML associations in Figure 8 are not so easily populated on the diagram. When attributes are used, the situation worsens considerably. As another example, consider the UML Employee class shown in Figure 9. This is nice and compact, but it makes it hard to express the common business rule that certain titles apply to only one sex (e.g. Lady applies only to females). In ORM this can be captured by a populated join-subset constraint as shown in the right hand side of the figure. In ConQuer, this constraint verbalizes as: if Person1 has a Title that applies only to Sex1 then Person1 is of Sex1. Step 5b of ORM's conceptual schema design procedure prompts the modeler to add the extra association between Title and Sex, and in this case the population becomes part of the rule. It is unclear as to how to approach this problem in UML, other than by converting title and sex to classes and writing down a population somewhere in a note.

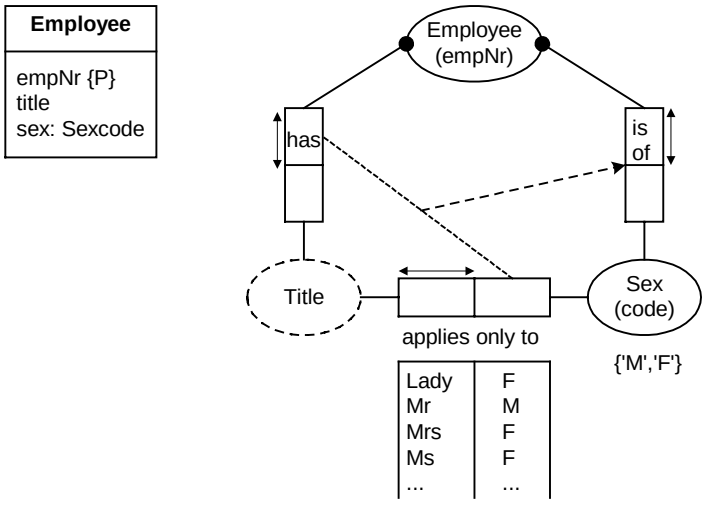

Figure 9: ORM makes it easy to capture the constraint between title and sex

As an example of a join-exclusion constraint, consider the following rule from a conference paper review application: no Person who works at an Institute that employs a Person who wrote a Paper may review that paper. As discussed in a later article, subset and equality constraints also provide one way of specifying derivation rules. In the absence of further marks on the schema diagram, ORM join constraint paths may sometimes be ambiguous. This problem may easily be resolved by having the modeler indicate the path in some way (e.g. by shift-clicking the predicates on the path) and then displaying this path in some way when the constraint is inspected (e.g. by shading).

Later issues

Later issues will discuss aggregation, initial value declarations, derivation rules and changeability settings 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. OMG-UML v1.2, OMG UML Revision Task Force website, http://uml.systemhouse.mci.com/.
5. Rumbaugh, J., Jacobson, I. & Booch, G. 1999, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading MA, USA.