UML data models from an ORM perspective: Part 10

by Dr. Terry Halpin

Director of Database Strategy, Visio Corporation

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

This paper is the tenth 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, language design criteria, 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, co-referencing and exclusion constraints with UML association classes, qualified associations, and xor-constraints respectively. Part 5 discussed subset and equality constraints. Part 6 discussed subtyping. Part 7 discussed value, ring and join constraints. Part 8 listed some recent updates to the UML standard, then discussed aggregation. Part 9 examined initial values and derived data in ORM and UML. Part 10 discusses changeability and collection types in UML and ORM.

Changeability properties

In UML, restrictions may be placed on the *changeability* of attributes, as well as the roles (ends) of binary associations. It is unclear whether changeability may be applied to the ends of n-ary associations, but my guess is that this is currently forbidden. The following three values for changeability are recognized, only one of which can apply at a given time:

- changeable
- frozen
- addOnly

The value "changeable" was previously called "none". Although the new term "changeable" was approved for UML 1.3 [0], some instances of "none" still occur in the standard; this oversight should be remedied in a later version. The default changeability is "changeable" (any change is permitted). Although the UML standard [0, p. 2-25] and

some authors [0, p. 166] indicate that "changeable" is a value, the standard also says "there is no symbol for whether an attribute is changeable", so it appears that this default cannot be explicitly declared. However it makes sense to allow explicit declaration of this default, and it would not be surprising to see the standard revised to permit it. The other settings (frozen and addOnly) may be explicitly declared in braces. For an attribute, the braces are placed at the end of the attribute declaration. For an association, the braces are placed at the opposite end of the association from the object instance to which the constraint applies.

Recall that a "link" is an instance of an association. The term "frozen" means that once an attribute value or link has been inserted, it cannot be updated or deleted, and no additional values/links may be added to the attribute/association (for the constrained object instance). The term "addOnly" means that although the original value/link cannot be deleted or updated, others values/links may be added to the attribute/association (for the constrained object instance). Clearly, addOnly is only meaningful if the maximum multiplicity of the attribute/association-role exceeds its minimum multiplicity.

As a simple if unrealistic example, see Figure 1. Here empNr, birthDate and country of birth are frozen for Employee, so they cannot be changed from their original value. For instance, if we assign an employee the empNr 007, and enter his/her birthdate as 02/15/1946 and birth country as 'Australia', then we can never make any changes or additions to that.

Notice also that for a given employee, the set of languages and the set of countries visited are addOnly. Suppose that when facts about employee 007 are initially entered, we set his/her languages to {Latin, Japanese} and countries visited to {Japan}. So long as employee 007 is referenced in the database, these facts may never be deleted. However we may add to these (e.g. later we might add the facts that employee 007 speaks German and visited India).

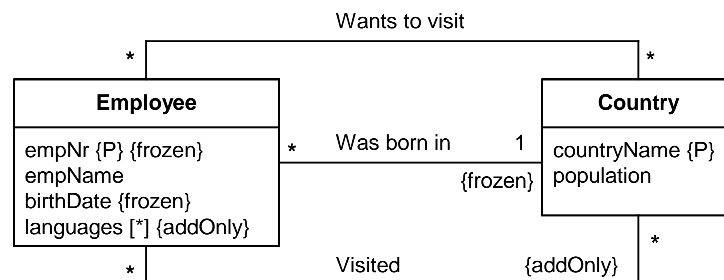

Figure 1 Changeability of attributes and association roles may be specified in UML

By default, the other properties are changeable. For example, employee 007 might change his name by deed poll from 'Terry Hagar' to 'Hari Seldon', and the set of countries he wants to visit might change, after some traveling, from {Ireland, Italy, USA} to {Greece, Ireland,}.

Some traditional data modeling approaches also note some restrictions on changeability. For example, Oracle's ER notation includes a diamond to mark a relationship as non-transferable (once an instance of an entity type plays a role with an object, it cannot ever play this role with another object). Although changeability restrictions may at first appear very useful, in practice their application in database settings is limited. One reason for this is that we almost always want to allow facts entered into a database to be changed. With snapshot data, this is the norm, but even with historical data, changes can occur. The most common occurrence of this is to allow for corrections of mistakes, which might be because we were told the wrong information originally or because we carelessly made a misspelling or typo when entering the data.

In exceptional cases, we might require that mistakes of a certain kind be retained in the database (e.g. for auditing purposes) but be corrected by entering later facts to compensate for the error. This kind of approach makes sense for bank transactions (see Figure 2). For example, if a deposit transaction for $100 was mistakenly entered as $1000, the record of this error is kept, but once the error is detected it can be compensated for by a bank withdrawal of $900. As a minor point, the balance is both derived and stored, and its frozen status is typically implied by the frozen settings on the base attributes, together with a rule for deriving balance.

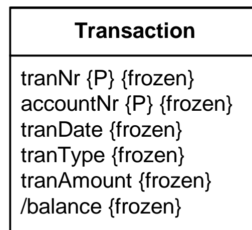

Figure 2 All attributes of Transaction are frozen

Although not stated in UML 1.3, some authors allow changeability to be specified for a class, as an abbreviation for declaring this for all its attributes and opposite association ends [0, p. 184]. For instance, all the {frozen} constraints in Figure 2 might be replaced by a single {frozen} constraint below the name "Transaction". While this notation is neater, it would be rarely used. Even in this example, we would probably want to allow for the possibility of adding non-frozen information later (e.g. a transaction might be audited by zero or more auditors).

Changeability settings may have more use in the design of program code than in conceptual modeling (e.g. {frozen} corresponds to const in C++). Although changeability settings are not supported in ORM, which focuses on static constraints, such features could easily be added as role properties if desired. In the wider picture, being able to completely model security issues (e.g. who has the authority to change what) would provide greater value. This view is nicely captured by the following comment of John Harris, in a recent thread on the InConcept website: "Rather than talk of "immutable" data I think it is better to talk of a privilege requirement. For instance, you can't change your recorded salary but your boss can, whether it's because you've had a pay rise or because

there's been a typing error. Privileges can be as complicated or as simple as they need to be, whereas "immutable" can only be on or off. Also, privileges can be applied to the insertion of new data and removal of old data, not just to updates”.

Collection types

Though collection types (e.g. sets, bags, sequences and arrays) are commonly used in programming, their use as record components in database schemas largely disappeared with the widespread acceptance of relational databases, where each table column is based on an atomic domain. However, the recent emergence of object-relational and object databases has once again allowed collection types to be embedded as database fields. Although a number of collection types were slated for inclusion in the object-relational database standard SQL3, the only one that made it was array (a one dimensional array with a maximum number of elements). It is anticipated that three further collection types will be added in SQL4: set (unordered collection with no duplicates); multiset (bag, i.e. unordered collection that allows duplicates); and list (sequence, i.e. an ordered bag). Some commercial systems already support these. Experience with these systems indicates that little performance gain is actually achieved by use of collection types; but this may change as the technology matures. Array, set, bag and list are also included as collection types in the object database standard ODMG 2.0 [0].

UML includes none of these as standard notations, but does include the {ordered} constraint to indicate mapping to an ordered set (i.e. a sequence with no duplicates); and its associated textual language OCL (Object Constraint Language) includes set, bag and sequence types as well as collection as their abstract supertype [0, pp. 38-49]. While UML allows collection types to be specified as stereotypes of classes, and realized as implementation classes [0, pp. 485-6], this usage seems geared toward code design so will not be elaborated here.

Different approaches have arisen as to how collection types should be specified within the conceptual analysis and logical design of data. Some proposals use collections directly within the conceptual schema, some introduce them only at the logical schema level, while some specify them as annotations to the conceptual schema to guide the mapping to the logical level. As a simple example, consider Figure 3. The ORM schema (a) and UML schema (b) depict driving as a many-to-many association. The employee name information is modeled as a functional fact type in ORM and as an attribute in UML. If this is mapped to a relational database system, then by default the m:n association maps to a separate table, resulting in a 2-table schema (c).

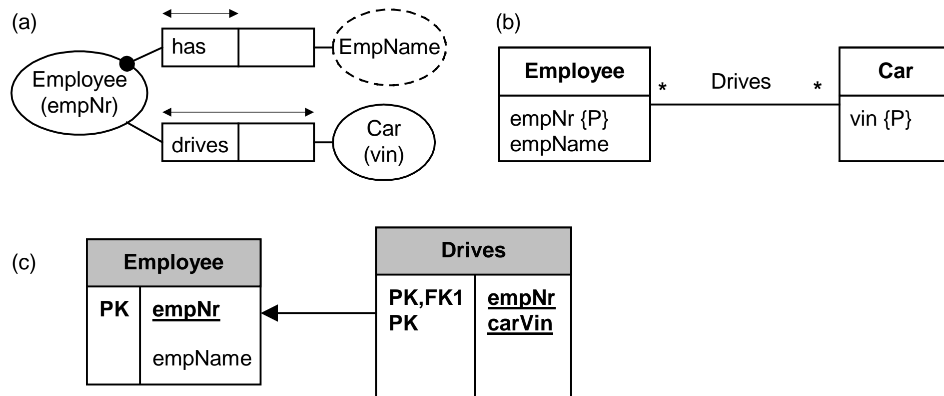

Figure 3 ORM schema (a) and UML schema (b) map by default to relational schema (c)

Now suppose that for some reason we wish to map both fact types into the same table, as shown in Figure 4(d). Some object-relational databases support this option. Clearly this mapping decision is an implementation, not a conceptual, issue, but how do we specify it? Visio Enterprise 5 lets you do this at the logical level (d), and VisioModeler lets you specify it either at the logical level or as an annotation to the ORM schema. The annotation shown in Figure 4(a) differs from that of VisioModeler (which uses a box between the role and its object type), but the idea is the same (indicating that this role maps to a set field of the co-role's table). The display of such annotations should be hidden during conceptual analysis, and toggled on only when we wish to discuss overrides to the default logical mapping. In UML we could invent a similar annotation, as in Figure 4(b), or instead use a multi-valued attribute, as in Figure 4(c), with this display being used only for discussing the logical mapping. As discussed in an earlier article, multi-valued attributes should never be used in conceptual analysis.

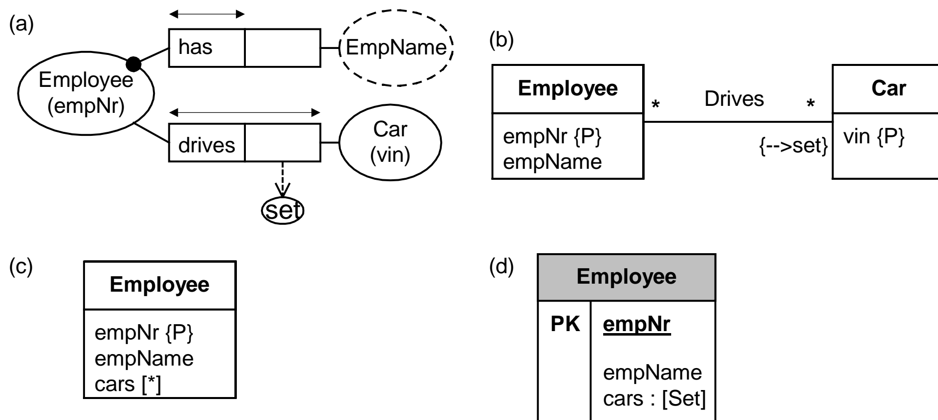

Figure 4 Some possible ways of indicating that driving should map to a set-valued column

Some extensions of ORM (e.g. PSM [0]) allow collection types (e.g. set, bag, sequence and schema) to be modeled as first class object types, using constructors often shown as a shape around the member object type. A sequence is an ordered bag, and in extended ORM its collection type may be marked "seq". If the sequence cannot have duplicates, it is a "unique sequence" (or ordered set) and is marked "seq". As an example of the unique sequence (or ordered set) constructor, see Figure 5(a). Here an author list is a sequence of authors, each of whom may appear at most once on the list. This may be modeled in flat ORM by introducing a Position object type to store the sequential position of any author on the list, as shown in Figure 5(b).

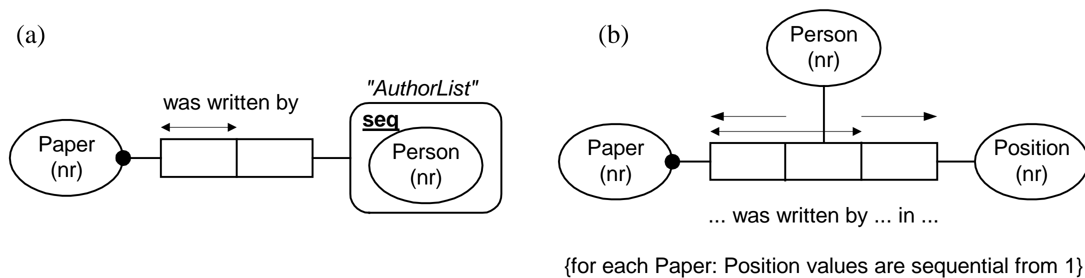

Figure 5 Unique sequence modeled in ORM with a constructor (a) or by introducing Position (b)

The uniqueness constraint on the first two roles declares that for each paper an author occupies at most one position; the constraint covering the first and third roles indicates that for any paper, each position is occupied by at most one author. The textual constraint below the graphic indicates that the positions in any list are numbered sequentially from 1. Although this ternary representation may appear awkward, it is easy to populate and it facilitates any discussion involving position (e.g. who is the second author for paper 21?). From an implementation perspective, a sequence structure could still be chosen: this can simplify updates by localizing their impact. However the update overhead of the positional structure is not onerous anyway, given set-at-a-time processing (e.g. to delete author n, simply set position to position–1 for position > n).

Though not shown here, the ternary solution can also be modeled in UML. If the ternary model is chosen as the base model, it would be useful to support the annotated binary shown in Figure 6(a) or Figure 6(b) as a view of the base model. In ORM we have shown a unique sequence annotation connected to the relevant role. This representation is equivalent to the {ordered} constraint in UML, as shown in Figure 6(b), indicating that the authors are to be stored as a unique sequence. The unique sequence annotation is not yet supported by Visio. UML does include "{ordered}" as a standard notation, but it does not yet include notations for other collections, although obvious ones suggest themselves (e.g. {sequence}).

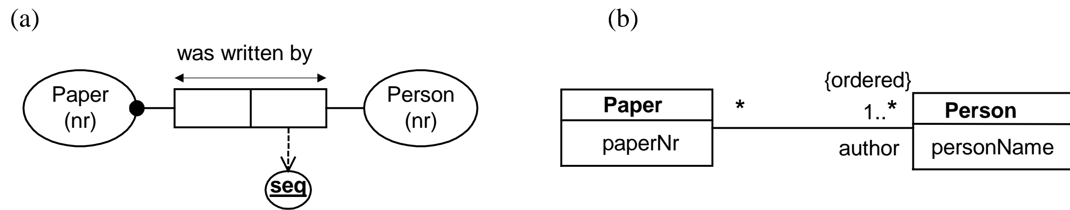

Figure 6 Unique sequence modeled with an annotation in ORM (a) and UML (b)

Flat models (no constructors) substantially simplify the declaration of constraints (which typically apply to members, not collections), derivation rules (and hence queries), and avoid arbitrary or non-conceptual decisions about how to store (and possibly duplicate) fact types and constraints. For example, in Figure 7 the ORM exclusion constraint may be verbalized: no Person wrote and reviewed the same Book. Although one could use collection types here (e.g. sets of books for an author, or sets of authors of a book) this would be extremely unwise, since it would complicate verbalization, validation, fact expression (possibly duplicated) and constraint expression (possibly duplicated). In conceptual modeling, we should not have to concern ourselves about how individual fact types might be stored in structures, or where the constraint code will reside. Such concerns are implementation details, and should be delayed until a clear conceptual picture of the world is obtained.

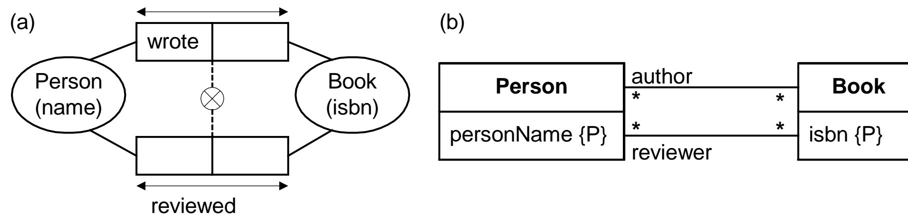

Figure 7 Pair-exclusion constraint in ORM (a) needs to be captured textually in UML

Since UML does not provide a graphical notation for such an exclusion constraint, it should be specified either informally as a note, or formally using a language of choice. Since OCL includes collection types with predefined operations, and the population of the association roles author and reviewer are sets, this constraint can be expressed in OCL as follows:

```
Book
self.author -> intersection(self.reviewer) -> isEmpty
```

Although this constraint expression is clear enough to somebody with a formal background, it is of little use for validating the rule with the subject matter expert

(typically a business person with little formal training). For such purposes, ORM's ConQuer language is far more suitable.

Next issue

The ten articles in this series have covered UML data modeling issues from an ORM perspective. My next couple of articles will consider other data modeling notations (flavors of ER, as well as IDEF1X) from an ORM viewpoint. Later on, I may return to UML to discuss its behavioral side.

References

Booch, G., Rumbaugh, J. & Jacobson, I. 1999, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading MA, USA.

Cattell, R.G.G. (ed.) 1997, *The Object Database Standard: ODMG 2.0*, Morgan Kaufmann Publishers, San Francisco.

Halpin, T. 1995, *Conceptual Schema and Relational Database Design, 2nd edn (revised 1999)*, WytLytPub, Bellevue WA, USA.

ter Hofstede, A.H.M., Proper, H.A. & Weide, th.P. van der 1993, 'Formal definition of a conceptual language for the description and manipulation of information models', *Information Systems*, vol. 18, no. 7, pp. 489-523.

Martin, J. & Odell, J. 1998, *Object-Oriented Methods: a Foundation, UML edn*, Prentice Hall, Upper Saddle River, New Jersey.

OMG, *UML Specification v. 1.3 final draft*, OMG UML Revision Task Force website, http://uml.systemhouse.mci.com/artifacts.htm.

OMG, *UML 1.3 Revisions and Recommendations*, Appendix A, issues 35-6, document ad/99-06-11, http://uml.systemhouse.mci.com/artifacts.htm.

Rumbaugh, J., Jacobson, I. & Booch, G. 1999, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading MA, USA.

Warmer, J. & Kleppe, A. 1999, *The Object Constraint Language: precise modeling with UML*, Addison-Wesley, Reading MA, USA.

This paper is made available by Dr. Terry Halpin and is downloadable from www.orm.net.