In workshop proceedings: UML: a critical evaluation and suggested future, HICCS-34 conference (Maui, January 2001), © 2000 IEEE.

Augmenting UML with Fact-orientation

Terry Halpin
Microsoft Corporation, USA
TerryHa@microsoft.com

Abstract

The Unified Modeling Language (UML) is more useful for object-oriented code design than conceptual information analysis. Its process-centric use-cases provide an inadequate basis for specifying class diagrams, and its graphical language is incomplete, inconsistent and unnecessarily complex. For example, multiplicity constraints on n-ary associations are problematic, the constraint primitives are weak and unorthogonal, and the graphical language impedes verbalization and multiple instantiation for model validation. This paper shows how to compensate for these defects by augmenting UML with concepts and techniques from the Object Role Modeling (ORM) approach. It exploits "data use cases" to seed the data model, using verbalization of facts and rules with positive and negative examples to facilitate validation of business rules, and compares rule visualizations in UML and ORM. Three possible approaches are suggested: use ORM for conceptual analysis then map to UML; supplement UML with population diagrams and user-defined constraints; enhance the UML metamodel.

1. Introduction

The Unified Modeling Language (UML) was adopted in 1997 by the Object Management Group (OMG) as a language for object oriented (OO) analysis and design. This paper is concerned with UML version 1.3, the latest approved version at the time of writing. A minor revision (1.4) should be approved around December 2000, and a major revision (2.0) should be completed a few years later. Though not yet a standard, UML has been proposed for standardization by the International Standards Organization, with approval likely around 2001 [28].

The UML notation includes the following kinds of diagram for modeling different perspectives of an application: use case diagrams, class diagrams, object diagrams, statecharts, activity diagrams, sequence diagrams, collaboration diagrams, component diagrams and deployment diagrams. This paper focuses on conceptual data modeling, so considers only the static structure (class and object) diagrams. Class diagrams are used for the data model, and object diagrams for data populations. Although not yet widely used for designing database applications, UML class diagrams effectively provide an extended Entity-Relationship (ER) notation that can be annotated with database constructs (e.g. key declarations). Background on UML may be found in its specification [31], a simple introduction [13] or a detailed treatment [6, 32]. In-depth discussions of UML for database design may be found in [30] and (with a slightly different notation) [3].

UML has become popular for designing OO program code. It is well suited for this purpose, covering data, behavior, and OO-implementation details (e.g. attribute visibility and directional navigation across associations). However, UML is less suitable for developing and validating a conceptual data model with domain experts. Its use-cases are process-centric, and in practice the move from use cases to class diagrams is often little more than a black art. Moreover, the UML notation prevents many common business rules from being diagrammed.

We believe these defects are best avoided by using fact-oriented modeling as a precursor to object-oriented modeling in UML. Object-Role Modeling (ORM) is the main exemplar of the fact-oriented approach, and is supported by CASE tools such as Microsoft Visio Enterprise [34]. For data modeling, ORM's graphical notation is more expressive and orthogonal than UML's, its models and queries are semantically stabler, and its design procedures fully exploit data examples using both verbalization and multiple instantiation to help capture and validate business rules with domain experts.

This paper identifies several weaknesses in the UML graphical language and discusses how fact-orientation can augment the object-oriented approach of UML. It shows how verbalization of facts and rules, with positive and negative examples, facilitates validation of business rules, and compares rule visualizations in UML and ORM on the basis of specified modeling language criteria. The following three approaches are suggested as possible ways to exploit the benefits of fact-orientation: (1) use ORM for conceptual information analysis and map the ORM model to UML; (2) use UML in its current form, supplemented by informal population diagrams and user-defined constraints; (3) correct and extend the UML metamodel to better support business rules.

The rest of this paper is structured as follows. Section 2 provides a brief comparative overview of UML and ORM, based on linguistic design criteria. Section 3 discusses verbalization issues related to multiplicity constraints on binary associations. Section 4 illustrates how "data use cases" help guide the data modeling process as a joint activity between modeler and domain expert. It also exposes problems with UML multiplicity constraints on n-ary associations, and highlights the need for a richer graphical constraint notation. Section 5 summarizes how the lessons learned from fact-orientation can be used to augment UML, identifies areas of future research, and lists references for further reading.

2. ORM, UML and language criteria

Object-Role Modeling is a conceptual modeling method that views the world as a set of objects (entities or values) that play roles (parts in relationships). For example, you are now playing the role of breathing (a unary relationship involving just you), and also the role of reading this paper (a binary relationship between you and this paper). An entity in ORM corresponds to a UML object, and a value to a UML data value. A role in ORM corresponds to an association-end in UML, except that ORM also allows unaries. The main structural difference between ORM and UML is that ORM excludes attributes as a base construct, treating them instead as a derived concept. For example, Person.birthdate is modeled in ORM as the fact type: Person was born on Date. Overviews of ORM may be found in [15, 16] and a detailed treatment in [14]. The ORM notation uses only a handful of symbols, readily mastered by UML modelers. Although various ORM-based proposals for process/behavioral modeling exist [e.g. 24], they are ignored here.

The ORM language was designed from the ground up to meet the following criteria: expressibility; clarity; learnability (hence orthogonality, parsimony and convenience); semantic stability (minimize the impact of change); semantic relevance (scope views to just the currently relevant task); validation mechanisms; abstraction mechanisms; and formal foundation. Background on these principles may be found in [1, 4, 25, 26]. Practical trade-offs between design criteria can arise, e.g. expressibility-tractability [29] and parsimony-convenience [18]. In this paper our focus is on validation mechanisms, expressibility and orthogonality.

The most debatable feature of ORM is its avoidance of attributes in the base model. This omission was originally made to avoid fuzzy and unstable distinctions about whether a feature should be modeled as an attribute or association [12]. Although this advantage is enjoyed by some other semantic modeling approaches, such as OSM [10], a disadvantage is that attribute-free diagrams often take up more space. A detailed argument that this price is worth paying can be found in [19]. The main advantages are that all facts and rules can be easily verbalized as sentences, all data structures can be easily populated with multiple instances, the metamodel is simplified, and models and queries are stabler since they are immune to changes that reshape attributes as associations. Finally the compactness of attribute-based models can still be achieved by deriving them as views (this is automatable).

Table 1 summarizes the main correspondences between conceptual data constructs in ORM and UML. Some examples are given later, and complementary discussions can be found in the references [14, 18, 19, 20, 21]. An uncommented "—" indicates no predefined support for the corresponding concept, and "†" indicates incomplete support. Clearly, ORM's built-in symbols provide greater expressibility for conceptual constraints on data.

Table 1 Conceptual data constructs in ORM and UML

ORM	*UML*
Data structures:	**Data structures**:
object type: entity type;	object class
value type	data type
— { use association }	*attribute*
unary association	— { use Boolean attribute }
2^+-ary association	2^+-ary association
objectified association	association class
co-reference	qualified association †
Predefined Constraints:	**Predefined Constraints**:
internal uniqueness	multiplicity of ..1 *
external uniqueness	— {use qualified assoc. } †
simple mandatory role	multiplicity of 1^+.. †
disjunctive mandatory role	—
frequency: internal; external	multiplicity †; —
value	enumeration, and textual
subset and equality	subset †
exclusion	xor †
subtype link and definition	subclass discriminator etc. †
ring constraints	—
join constraints	—
object cardinality	class multiplicity
—{use unique and ring}†	aggregation/composition
Textual constraints	**Textual constraints**

Because of its orthogonality and avoidance of attributes, ORM achieves this greater expressibility without adding complexity. For example, ORM includes a disjunctive mandatory role (inclusive-or) constraint to constrain instances of an object type to play at least one of a set of roles (e.g. each Applicant must have a Qualification or a JobReference or both). ORM also includes an exclusion constraint that may apply between compatible role sequences (e.g. no Person who writes a Paper may referee that Paper). In ORM an exclusion constraint between single roles may be orthogonally combined with an inclusive-or constraint to form an exclusive-or constraint (e.g. no Person may get a BusPass and a ParkingPermit). In contrast, UML supports an exclusive-or constraint as a primitive, but no inclusive-or and no general exclusion constraint.

Unlike UML, ORM allows constraints to be applied wherever they makes sense. For example, subset constraints may apply between compatible role sequences, not just associations (e.g. if a Person drives a Car then that Person has a DriverLicence). Ring constraints are logical constraints on ring associations (e.g. “no Person reports to himself/herself” is an irreflexive ring constraint). Join constraints apply to roles from connected predicates, e.g. each Employee who works in a Country also speaks a Language that is spoken in that Country).

Although the additional constraints in ORM often arise in practice, UML models often omit them unless the modeler is very experienced. Both UML and ORM allow the user to add constraints and derivation rules in a textual language of their choice. UML suggests OCL (Object Constraint Language) [33] for this purpose, but does not mandate its use. ORM’s conceptual query language, ConQuer [4, 5, 21], provides a formal but higher level alternative to OCL. Although textual languages are needed for completeness, it is easier for a modeler to think of a rule if it is part of his/her graphical rule language.

3. Binary associations

Since the domain expert is the person who understands the universe of discourse (UoD) or application domain, it is critical to promote good communication between the modeler and the domain expert in the conceptual analysis phase. Subject matter experts are often not technically skilled in modeling notations, so any business rules should be verbalized in their natural language for model validation. This section discusses verbalization of binary associations and their associated multiplicity constraints.

Consider a UoD in which employees must occupy a room, possibly shared with another employee, and some rooms may be unoccupied. For a given state of the database, the *population* of a type is the set of instances of that type that are present in the database. For this UoD, each population of the occupancy association is a total function (mandatory n:1 relation) from the population of Employee to the population of Room. A significant sample population is included in the instance diagram at the top of Figure 1.

Figure 1(a) depicts this binary association in UML. Classes are denoted by named rectangles, and *binary associations* by connecting lines. The association ends correspond to roles in ORM, and may be given a role name (e.g. “office”). The association itself may be given a name (e.g. “Occupies”) as well as a marker “ ▸ ” to indicate the direction in which the association should be read. So long as an association name is supplied, the association can be verbalized as a sentence type (e.g. Employee occupies Room).

The association roles (ends) may be adorned with *multiplicity constraints* that specify the possible multiplicities. For example, “1..*” means one or more (at least one), “0..1” means zero or one (at most one), “1” abbreviates “1..1” (exactly one) and “*” abbreviates “0..*” (zero or more). Like ORM, UML allows multiplicities to include combinations of numbers and number ranges (e.g. “2, 4, 6, 10..20”), even if these would be rarely used.

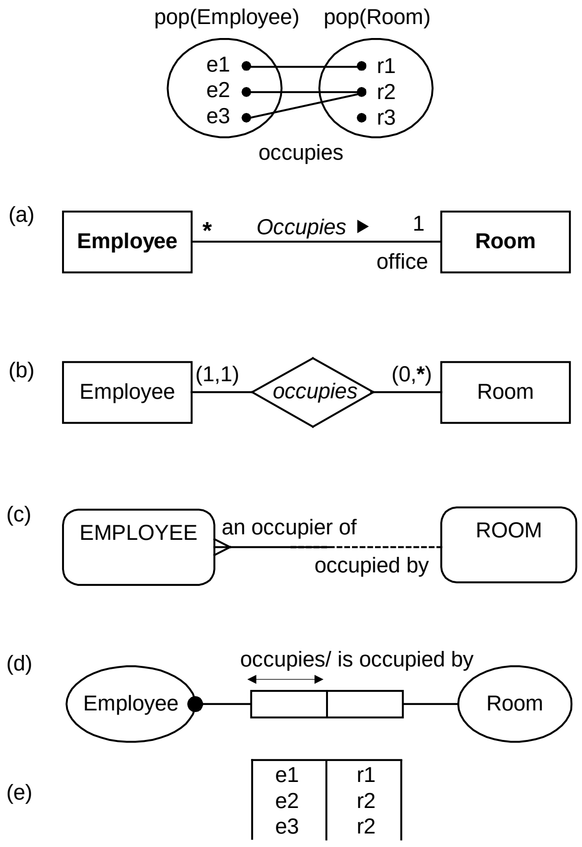

Figure 1 Mandatory n:1 association in (a) UML (b) DSB-ER (c) Barker-ER (d) ORM

UML places each multiplicity constraint on the “*far role*”, in the direction in which the association is read. Hence the multiplicity constraint on the Room role may be *verbalized* thus: **each** Employee occupies **exactly one** Room. The “*” constraint on the Employee role may be verbalized: **it is possible that more than one** Employee occupies **the same** Room. The “*” (zero or more) is the default multiplicity for a role, and may be regarded as the absence of a constraint rather than a constraint. Hence we could omit its verbalization, but it is normally safer to provide it to clarify its impact.

These verbalizations, which we developed for use in ORM, rely on singular terms being used for class names (e.g. “Employee” not “Employees”) for natural phrasing. Words shown in bold type have formal meaning, allowing an ORM tool to automatically generate an ORM diagram from the textual formulation of the association and its constraints. Although UML does not have any formal verbalization, a request for proposal has been issued by the UML committee for a “Human Readable Textual Notation”, so something like this could eventually be added to UML. ORM’s verbalization patterns could provide a good basis for extending UML in this way.

Figure 1(b) shows the same association in an ER notation recently proposed by Dey, Storey and Barron for work with binary and n-ary (n > 2) associations [9]. Let’s call this DSB-ER notation after its proponents. Here, entity types are depicted as named rectangles and binary relationships are depicted as named diamonds, as in Chen’s original ER [8]. The constraints are called *participation constraints*. The association and its constraints may be verbalized as before. As with some other versions of ER, this notation places the constraint on the “*near role*”, to indicate the minimum and maximum number of times each instance of the role player must participate in that role. Hence the “(1, 1)” and “(0,*)” on the left and right roles correspond to UML’s “1” and “*” placed on the right and left roles respectively (the opposite).

Figure 1(c) shows the same example in the Barker-ER notation popularized by Richard Barker [1] and Oracle Corporation. Unlike UML and DSB-ER, but like ORM, the Barker notation supports *forward and inverse readings* of binary relationships. This is useful practice facilitates navigation in different directions around a schema, and often leads to improved verbalization of rules. Some UML users have added their own notations in this regard, such as appending reverse readings in parentheses to the association name [11]. However the UML specification has no formal support for this. We recommend that UML be extended by adding a slot in its metamodel to store reverse readings, and provide a standard syntax for their display.

Unlike the two previous notations, Barker-ER uses *separate notations for minimum and maximum cardinalities*. Minimum cardinalities of 0 (optional) or at least 1 (mandatory) are specified as optional and mandatory roles. A role that is *optional* for its entity type is designated by a dashed line-half, and a role that is *mandatory* is depicted by a solid line-half: these are specified on the near role. A maximum cardinality of 1 is the default (no explicit mark), and a maximum cardinality of many is depicted as a crows-foot: these are shown on the far role as in UML.

Barker [1] suggests a relationship naming scheme that, while awkward for verbalizing relationship types or instances, does allow a structured means of verbalizing the cardinality constraints. Let *A R B* denote an infix relationship *R* from entity type *A* to entity type *B*. Name *R* in such a way that each of the following four patterns results in an English sentence: **each** A (**must** | **may) be** *R* (**one and only one** *B* | **one or more** *B-plural-form*). Use “must” or “may” when the first role is mandatory or optional respectively. Use “one and only one” or “one or more” when the cardinality on the second role is one or many respectively. For example, the constraints in Figure 1(c) verbalize as: **each** Employee **must be** an occupier of **one and only one** Room; **each** Room **may be** occupied by **one or more** Employees. This verbalization convention is good for basic multiplicity constraints on infix binaries. However it is less general than ORM’s approach, which applies to instances as well as types, for predicates of any arity, with no need for pluralization.

Figure 1(d) shows the same association in ORM. Entity types are depicted as named, solid ellipses, and relationships as named sequences of one or more roles, with each role depicted as a box connected by a line to its object type. A relationship is called a *fact type* unless it is used simply to provide a primary reference scheme. For binary associations, forward and inverse readings may be provided, separated by a slash. As in UML, each role may also be named, although ORM tools typically store role names on property sheets rather than display them on the diagram.

A black dot “•” on a role connector indicates the role is *mandatory* (must be played by each instance in the population of the object type). By default, a role is optional (no black dot). ORM constraints were designed to facilitate validation using sample populations. An arrow-tipped bar over one or more roles is a *uniqueness constraint* declaring that each entry in the population of that role sequence is unique (occurs there exactly once). Any relationship may be populated with a table where each column corresponds to the role in that position. So the constraint over the left role of Figure 1(d) indicates that entries in the left column of Figure 1(e) must be unique, unlike the right column. If the association were instead many-to-many, the constraint would span both roles and only the entry-pairs making up the table rows must be unique.

Of the four notations, only UML depicts a mandatory role by a minimum multiplicity > 0 on the far role. As we’ll see in the next section, this leads to problems for n-ary associations. As it turns out, of all the notations discussed, only the ORM notation generalizes properly for *n*-ary associations.

4. Data use cases and n-ary associations

Use cases in UML illustrate ways in which the required information system may be used, so they are useful in requirements analysis. However because they focus on *behavioral* modeling, they can only go so far in helping the modeler arrive at a *data* model. They should be supplemented by examples of information that the system is expected to manage. In ORM these examples have traditionally been referred to as “information samples familiar to the domain expert”. By analogy with the UML term, we call them *data use cases*. They can be output reports or input screens, and since they exist at the external level they can present information in many different ways (e.g. tables, forms, graphs, diagrams).

Whatever the appearance of a data use case, a subject matter expert should be able to verbalize its information in The modeler then transforms that informal verbalization into a formal yet natural verbalization that is clearly understood by the domain expert. These two verbalizations, one by the domain expert transformed into one by the modeler, comprise step 1 of ORM’s conceptual analysis procedure. Here we use verbalization of populations to arrive at the fact instances that are then abstracted to fact types. Constraints and derivation rules are meta-facts (facts about the object facts), which are then added and themselves validated by verbalization and population. This approach is very effective in practice, and we believe it is an ideal precursor to the specification of the data model in UML or any other language.

Suppose that our system is required to output reports like that shown in Figure 2. We ask the domain expert to read off the information contained in the tables and then rephrase this in formal English. For example, the subject matter expert might read off the facts on the top row of the first table as follows: Archery is new (it’s the first year it’s been included in the rankings); the US ranks first in archery, and scored 10 points for that. As modelers, we note that Rank functionally determines Points in the population, so ask: Does the Rank (e.g. 1) determine the Score (e.g. 10)? The domain expert replies in the affirmative (if he/she gets this wrong, ORM’s arity-check can detect it later [14]).

We now rephrase the information into elementary sentences: the Sport named ‘Archery’ is new; the Country coded ‘US’ has the Rank numbered 1 in the Sport named ‘Archery’; the Rank numbered 1 earns the Score 10 points. Similarly, the top row of the second table may be verbalized as: the Country coded ‘AD’ has the CountryName ‘Andorra’. If reference schemes are agreed to up front, these long-winded verbalizations can be abbreviated. Once the domain expert agrees with the verbalization, we proceed to abstract from the fact instances to the fact types.

(a) ** new*

Sport	*Rank*	*Country*	*Points*
Archery *****	1	US	10
Baseball	1	US	10
	2	JP	5
Cricket	1	AU	10
	1	GB	10
...	...	...	...

(b)

Country	
Code	*Name*
AD	Andorra
AE	United Arab Emirates
...	...
ZW	Zimbabwe

Figure 2 Two sample output reports for a data use case

We may now draw the conceptual schema and populate it with sample facts. For discussion purposes, we consider the ORM solution (Figure 3) before the UML solution. Simple reference schemes may be abbreviated in parenthesis (e.g. “Country(code)” abbreviates the injective association Country has Countrycode). Value types need no reference scheme, and appear as named, dashed ellipses (e.g. CountryName). Here we have one unary fact type, Sport is new, two binary associations Country has CountryName, Ranks earns Score, and one ternary association Country has Rank in Sport.

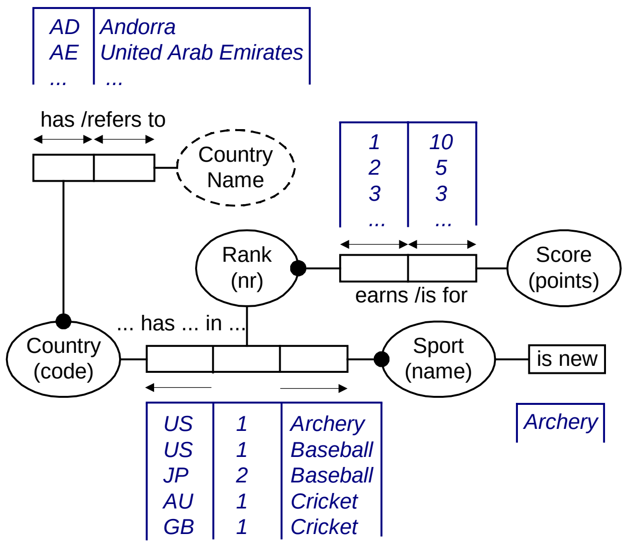

Figure 3 ORM schema for Figure 2, with sample data

Unlike other approaches, ORM allows mixfix predicates, which are sentences with object holes (denoted by "…") that may appear anywhere in the sentence. In this example, the ternary predicate is "… has … in …". This allows verbalization of sentences of any arity in any natural language, along with their associated constraints and derivation rules. Other approaches use a simple name for the verb phrase or assume binary infix predicates, that support only SVO (Subject Verb Object) languages, not SOV languages (e.g. Japanese) or VSO languages (e.g. Tongan). In principle, mixfix predicates could be used in UML, by extending its metamodel with positional information to provide a role order for predicate readings.

For each fact type in Figure 3, a sample fact table has been added to help validate the constraints. ORM schemas can be represented in diagrammatic or textual form, and tools such as Visio Enterprise can automatically transform between the two representations. Models are *validated* with domain experts in two ways: *verbalization*; and *population*. For example, the uniqueness constraints on the rank association in Figure 3 verbalize as: **each** Rank earns **exactly one** Score; **each** Score refers to **at most one** Rank. The 1:1 nature of this association is illustrated by the population, where each column is unique. A sample row for rank 3 has been added to illustrate the mandatory and optional nature of the roles played by Rank (a rank's score must be recorded even if no country achieves this rank).

The uniqueness constraint on the first and last roles of the ternary has a *positive verbalization* of: **each** Country has **at most one** Rank in **each** Sport. This is illustrated by the population, where the Country-Sport value pairs are unique. To double check a constraint in ORM, a *negative verbalization* of the constraint may be given, as well as a *counter-example* to test whether the constraint may be violated. For example, the uniqueness constraint on the ternary may also be verbalized thus: **it is impossible that the same** Country has **more than one** Rank in **the same** Sport. Adding the counter-row (US, 2, Archery) to the sample population of the ternary gives the US two ranks in archery, and hence violates the uniqueness constraint. Concrete examples like this make it easier for domain experts to see whether the constraint being tested really is a rule.

Because all fact types are elementary, and no attributes are used, populations never contain null values. Although closed or open world semantics may be chosen, the default semantics is closed world. For example Baseball appears in the population of Sport but does not play the role "is new", so we know it is not new. This is less confusing to the domain expert than assigning False to a boolean attribute, as in UML For this reason, and to support natural verbalization, we suggest that UML be extended to allow unaries. A trivial change to the metamodel would allow this (change the multiplicity on Association-end from "2..*" to "1..*"). However, pragmatism may require an inelegant alternative that is easier for vendors to support.

Unlike other approaches, ORM allows *n* readings for any *n*-ary predicate (*n* > 0), one starting at each role. This facilitates constraint declaration, and navigation through the information model from any starting position using natural sentences [4, 5]. In principle, the UML metamodel could be extended to support this.

Figure 4 shows a UML schema for the same UoD. All the ORM binary fact types are modeled here as attributes. In the absence of a standard UML syntax for primary identification or uniqueness constraints on attributes, we use our own notations "{P}" and "{U1}" respectively. Such notations are needed if UML is to be used to completely model even simple database applications.

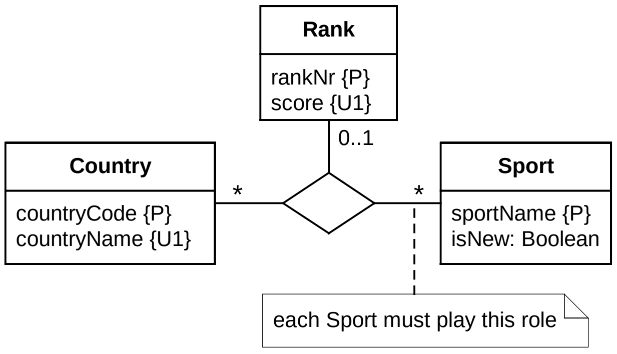

Figure 4 UML schema for Figure 2

The uniqueness constraint from the ORM ternary is modeled in UML using the 0..1 multiplicity constraint on the role played by Rank. The "*" multiplicities indicate the absence of any other uniqueness constraint. If an *n*-ary fact type is elementary, any internal uniqueness constraint must span either *n*-1 or *n* roles. The UML notation for multiplicity constraints can express these cases, but cannot express uniqueness or frequency constraints on fewer than *n*-1 roles. Hence unlike ORM it cannot be used to specify compound fact types that may be required for derivation or denormalization. The DSB-ER notation was developed to cater for cardinalities on *n*-aries, but is even worse than UML in this regard since it cannot express composite uniqueness and frequency constraints. The Barker-ER notation has the same problem if extended to *n*-aries.

Note that the *simple mandatory role constraint on Sport cannot be expressed by a multiplicity constrain*t in UML. It might be thought that this constraint can be expressed by changing the multiplicity on the Country role to 1..*. But this would mean that each Sport-Rank pair formed from the populations of Sport and Rank must be associated with at least one country. But this is not true, since the role played by Rank is optional. For example, the pairs Archery-2 and Archery-3 have no associated country in the sample population. As discussed later, any attempt to redefine the semantics of multiplicity constraints in terms other than the populations of its object types leads to other problems.

This exposes a fundamental problem with the scaleability of UML's multiplicity notation. Although it caters adequately for binaries, it cannot express a simple mandatory constraint on at least 1 and at most *n*-2 roles within an *n*-ary association. If we are to use an *n*-ary in UML, the only thing we can do in such cases is to add a textual description of the constraint in a note, as in Figure 4. *This problem is a direct consequence of choosing to attach minimum multiplicity to a far role instead of the near role*. The DSB-ER and ORM notations can express mandatory constraints on roles of *n*-aries, and the Barker-ER notation could be extended to do so, since each attaches minimum multiplicity on the near role.

Sometimes, we can overcome this problem with UML by binarizing the *n*-ary. For example, Figure 5 expresses the fact type Country is ranked in Sport as a binary association, that is objectified as the class Ranking. The mandatory role for Sport is now catered for by the 1..* constraint on the role for Country. However, this approach has problems. To begin with, it is often too unnatural. If the domain expert thinks in terms of a ternary, why force him/her to rethink the model in terms of binaries? More importantly, this solution does not always work in UML. For example, suppose we have the additional constraint that no ties are allowed for sport ranks. There is no symbol in UML to express this rule on the binarized solution (although it can be expressed on the ternary).

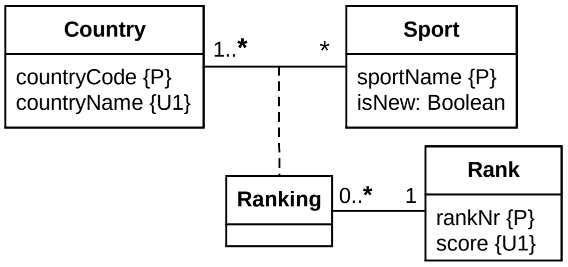

Figure 5 Alternative UML schema for Figure 2

The only way to express the no-ties rule with the binarized model would be to extend UML with the additional notion of an external multiplicity constraint that can span model elements from different associations. ORM already includes such a constraint. For example, Figure 6 shows the binarized solution in ORM with an external uniqueness constraint (circled "u") to indicate that each Sport-Rank pair is associated with at most one Country in the overall association. ORM shows an objectified association by enclosing the association in an envelope. Although this works, it is more awkward to think about than the ternary solution for this case.

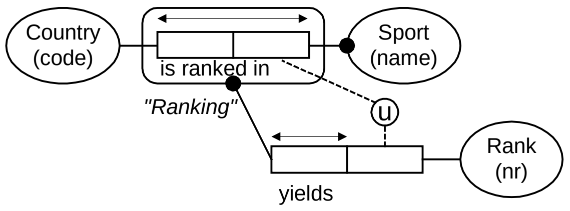

Figure 6 A nested ORM model that forbids ties

The next example illustrates the no-ties rule on the ternary, as well as another defect of multiplicities in UML. Consider the report shown in Table 2. In this UoD, no ties are allowed, and we are interested only in the first two ranks. Moreover, we may list a sport before any other details (e.g. ranking) are known for it. If a sport is ranked, we must know both its first and second place getters.

Table 2 A data use case for a somewhat different UoD

Sport	*Rank*	*Country*	*Points*
Aikido	?	?	?
Archery	1	US	10
	2	GB	5
Baseball	1	US	10
	2	JP	5
Basketball	?	?	?
Cricket	1	AU	10
	2	GB	5
...	...	...	...

An ORM schema for this situation is shown in Figure 7, together with a sample population for the ternary association and for the object type Sport. Here the "!" on Sport indicates it is an independent object type (instances of it can exist without playing any fact role). A meta-rule in ORM implies that any population object must play in some fact unless it is declared independent. There is no space here to extol the virtues of this rule, but its practical utility is such that we believe it should be added to UML.

Notice the uniqueness constraint over the roles played by Rank and Sport in the ternary. This enforces the no-ties rule. In ORM the positive verbalization of this constraint is: **given any** Rank **and** Sport, **at most one** Country has **that** Rank in **that** Sport. This is supported by the sample population. The negative verbalization of the constraint is: **it is impossible that more than one** Country has **the same** Rank in **the same** Sport. The negative verbalization is especially useful in using *counter-examples* to check the constraint. For instance, if we gave both Australia and Great Britain the rank 1 in cricket (as in Figure 3), this would violate the constraint. Such concrete counter-examples make it easy for domain experts to validate doubtful constraints.

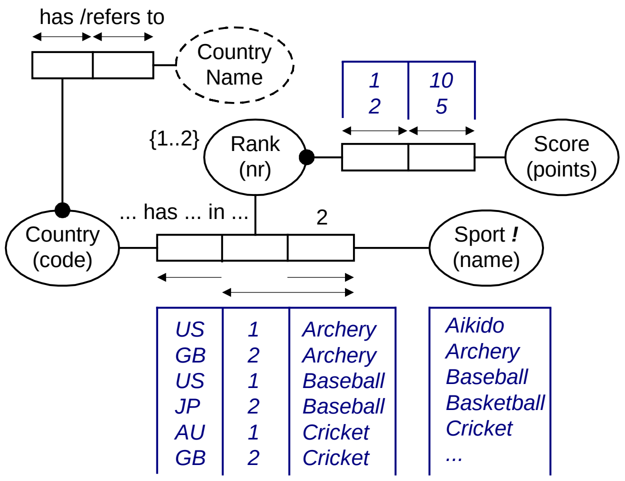

Figure 7 ORM schema for Table 2 with sample data

The frequency constraint of 2 on the Sport role means any sport that plays that role does so exactly twice. In the context of the uniqueness constraints and the value constraint of {1..2} on Rank, this ensures that both ranks are recorded for any ranked sport. Again, the population clarifies the constraint. Notice that some sports (e.g. Aikido) have not yet been ranked. Figure 8 shows the UML solution. There is no way of specifying the frequency constraint via a multiplicity constraint, so it has been added informally in a note.

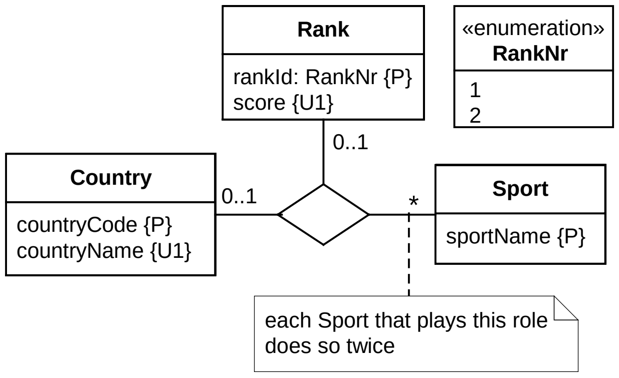

Figure 8 UML schema for Table 2

The two uniqueness constraints are expressed using 0..1 multiplicity constraints. This is possible because uniqueness is a case of maximum multiplicity. The frequency constraint of 2 cannot be expressed on a ternary in UML because it involves a minimum occurrence frequency of 2. An occurrence frequency of *n* means: if an instance plays the role, it does so *n* times. *Given any n-ary association, UML multiplicity constraints cannot express a minimum occurrence frequency above 1 for any role* (or combination of fewer than *n*-1 roles).

ORM allows mandatory and frequency constraints over a set of roles (possibly from different associations). Uniqueness constraints are just frequency constraints of 1 with a special notation because of their importance and ubiquity. These constraints are orthogonal, and apply to associations of any arity. UML's multiplicity constraints can express simple mandatory and frequency constraints for binary associations, but cannot express mandatory role constraints or minimum occurrence frequencies above 1 for roles in *n*-ary associations. So UML's multiplicity notation is far weaker than expected.

In fact, the whole notion of a *minimum multiplicity above 0 is problematic for n-aries in UML.* The UML 1.3 specification offers only the following description for the semantics of multiplicities in n-ary associations: "The multiplicity of a role represents the potential number of instance tuples in the association when the other n-1 values are fixed" [31, p. 3-73]. Consider a ternary association *R(A, B, C).* Let pop(*A*), pop(*B*) and pop(*C*)be the populations of *A*, *B* and *C* (in the database, not necessarily just in *R*), and pop(*rA*), pop(*rB*) and pop(*rC*) be the populations of the roles in R. Let *R* have multiplicities *, *, 2..* on the roles of *A*, *B*, *C* respectively. What does the 2 mean? For consistency with the meaning of multiplicities for binary associations, we should define it thus: each pair (*a*, *b*), where *a* is in pop(*A*) and *b* is in pop(*B*), is associated in *R* with at least 2 instances from pop(*C*). But such a constraint is in practice virtually useless, since it far too strong to apply except in pathological cases. To base the constraint on the types rather than populations would be even worse in this regard.

What we really need is a way to define the constraint in terms of *R*'s population. For example, each pair (*a*, *b*) that occurs in the projection pop(*R*)[*a*, *b*] is associated in *R* with at least 2 instances from pop(*C*). This corresponds to an ORM minimum frequency constraint of 2 on (*rA*, *rB*). Although useful and desirable, this definition is inconsistent with the whole approach to multiplicity constraints in UML. For example, if accepted it would mean that minimum multiplicities of 0 could never occur.

Internal frequency (and uniqueness) constraints in ORM can be efficiently implemented and validated because they apply just to the local population of their predicate. Mandatory constraints refer to the population of an object type, so are ontologically distinct as well as harder to enforce. Because of their global impact, mandatory constraints need to be considered more carefully. For such reasons, the separation of mandatory and frequency constraints is highly desirable.

To address the problems with UML multiplicities on *n*-aries, there are a number of possible solutions. Ideally, multiplicity constraints for associations should be replaced by ORM's mandatory and frequency/uniqueness constraints, at least for *n*-ary associations. However this is unlikely to ever happen, and would cause backward compatibility headaches. We could try adding extra constraints for mandatory and frequency for *n*-ary associations. This would achieve the required expressibility but would make UML even more unnecessarily complex than it is now (e.g. the concept of mandatory role would be dealt with by a multiplicity constraint on binaries but by a mandatory constraint on n-aries). A third solution is to use ORM for the original analysis where the constraints can be easily declared and validated, then map the ORM model to UML where the constraints would appear in notes. Since the ORM notation is easily mastered, and requires no change to the UML notation, the third solution seems attractive, and could certainly be automated.

As a final note on the no-ties example, we might try to overcome the problem of expressing the frequency constraint in UML by transforming the ternary into two binary associations: Country is first in Sport; Country is second in Sport, as shown in Figure 9. However apart from the fact that this transformation doesn't scale (e.g. large numbers of ranks), there are now two constraints that get lost.

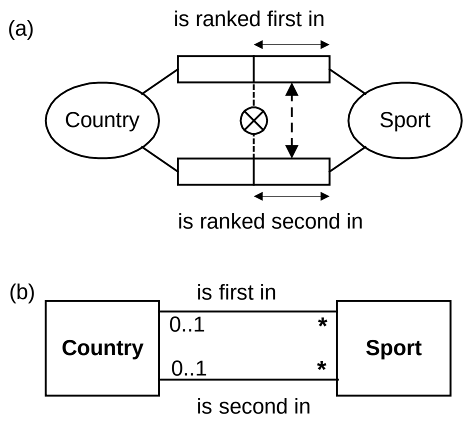

Figure 9 The exclusion and equality constraints in (a) are lost in the UML model (b)

The ORM model in Figure 9(a) shows the missing constraints. The pair-exclusion constraint denoted by a circled "X" enforces the no-ties rule that no country can be ranked first and second in the same sport. The equality constraint shown as a dashed line with arrowheads indicates that if a sport has a winner it also has a runner-up, and vice versa. Although these constraints can be added informally in notes to the UML diagram, it would be better to extend the UML metamodel to support them. Currently UML is very unorthogonal and restrictive with regard to constraints. It supports an exclusive-or (xor) constraint but no exclusive constraint and no inclusive-or constraint. Although UML's xor constraint is described as applying between associations, it actually applies between roles. UML supports a subset constraint between full associations but not between parts of associations (e.g. roles). The UML specification also contains a number of inconsistencies in its handling of these constraints. For a formal discussion of such inconsistencies and a means of extending the UML metamodel to adequately capture such constraints, see [19, section 5].

5. Conclusion

Fact-orientation, as exemplified by ORM, provides many advantages for conceptual data analysis, including expressibility, validation by verbalization and population at both fact and constraint levels, and semantic stability (e.g. avoiding changes caused by attributes evolving into associations). ORM also has a mature formal foundation that may be used to refine the semantics of UML.

Object-orientation, as exemplified by UML, provides several advantages such as compactness, and the ability to drill down to detailed implementation levels for object-oriented code. If UML is to be used for conceptual analysis of data, some ORM features can be adapted for use in UML either as heuristic procedures or as reasonably straightforward extensions to the UML metamodel and syntax. These include mixfix verbalizations of associations and constraints for associations, and exploitation of data use cases by populating associations with tables of sample data using role names for the column headers.

However there are some fundamental aspects that need drastic surgery to the semantics and syntax of UML if it is ever to cater adequately for non-binary associations and some commonly encountered business rules. This paper revealed some serious problems with multiplicity constraints on *n*-ary associations, especially concerning non-zero minimum multiplicities. For example, they cannot be used in general to capture mandatory and minimum occurrence frequency constraints on even single roles within *n*-aries, much less role combinations. Moreover, UML's treatment of set-comparison constraints is defective. Although it is possible to fix these problems by changing UML's metamodel to be closer to ORM's, such a drastic change to the metamodel may well be ruled out for pragmatic reasons (e.g. maintaining backward compatibility and getting the changes approved).

In contrast to UML, ORM has only a small set of orthogonal concepts that are easily mastered. UML modelers willing to learn ORM can get the best of both approaches by using ORM as a front-end to their data analysis and then mapping the ORM models to UML, where the additional constraints can be captured in notes or textual constraints. Automatic transformation between ORM and UML is feasible, and is currently being researched.

References

1. Barker, R. 1990, *CASE*Method: Tasks and Deliverables*, Addison-Wesley, Wokingham, England.
2. Bentley, J. 1988, 'Little languages', *More Programming Pearls*, Addison-Wesley, Reading MA, USA.
3. Blaha, M. & Premerlani, W. 1998, *Object-Oriented Modeling and Design for Database Applications*, Prentice Hall, New Jersey.
4. Bloesch, A. & Halpin, T. 1996, 'ConQuer: a conceptual query language', *Proc. 15th International Conference on Conceptual Modeling ER'96* (Cottbus, Germany), B. Thalheim ed., Springer LNCS 1157 (Oct.) 121-133.
5. Bloesch, A. & Halpin, T. 1997, 'Conceptual queries using ConQuer-II', *Proc. 16th Int. Conf. on Conceptual Modeling ER'97* (Los Angeles), D. Embley, R. Goldstein eds, Springer LNCS 1331 (Nov.) 113-126.
6. Booch, G., Rumbaugh, J. & Jacobson, I. 1999, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading MA, USA.
7. Campbell, L., Halpin, T. & Proper, H. 1996, 'Conceptual schemas with abstractions: making flat conceptual schemas more comprehensible', *Data & Knowledge Engineering*, 20, 1, 39-85.
8. Chen, P.P. 1976, 'The entity-relationship model—towards a unified view of data', *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9–36.
9. Dey, D., Storey, V.C. & Barron, T.M. 1999, 'Improving database design through the analysis of relationships', *ACM Transactions on Database Systems*, vol. 24, no. 4, pp. 453-486.
10. Embley, D. 1998, *Object Database Management*, Addison-Wesley.
11. Eriksson, H. & Penker, M. 2000, *Business Modeling with UML – Business Patterns at Work*, John Wiley.
12. Falkenberg, E. 1976, 'Concepts for modelling information', *Modelling in Data Base Management Systems*, G. Nijssen ed., North-Holland, Amsterdam, pp. 95-109 (see esp. p. 104, where "properties" means "attributes").
13. Fowler, M. with Scott, K. 1997, *UML Distilled*, Addison-Wesley.
14. Halpin, T. 1995, *Conceptual Schema and Relational Database Design, 2nd edn* (revised 1999), WytLytPub, Bellevue WA, USA.
15. Halpin, T. 1998, 'Object Role Modeling (ORM/NIAM)', *Handbook on Architectures of Information Systems*, P. Bernus, K. Mertins & G. Schmidt eds, Springer-Verlag, Berlin, pp. 81-101.
16. Halpin, T. 1998, 'Object Role Modeling: an overview', available online at http://www.orm.net/overview.html.
17. Halpin, T.A. 1998-9, 'UML data models from an ORM perspective: Parts 1-10', *Journal of Conceptual Modeling*, InConcept, Minneapolis USA, available online from www.orm.net/uml_orm.html.
18. Halpin, T.A. 1999, 'Data modeling in UML and ORM revisited', *Proc. EMMSAD'99: 4th IFIP WG8.1 Int. Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, Heidelberg, Germany (June).
19. Halpin, T.A. 2000, 'Integrating fact-oriented modeling with object-oriented modeling', *Information Modeling in the New Millenium*, eds M. Rossi & K. Siau, Idea Group Publishing Company, Hershey, USA.
20. Halpin, T.A. & Bloesch, A.C. 1998, 'A comparison of UML and ORM for data modeling', *Proc. EMMSAD'98: 3rd IFIP WG8.1 Int. Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, Pisa, Italy (June).
21. Halpin, T.A. & Bloesch, A.C. 1999, 'Data modeling in UML and ORM: a comparison', *Journal of Database Management*, vol. 10, no. 4, Idea group Publishing Company, Hershey, USA, pp. 4-13.
22. Halpin, T. & Proper, H. 1995, 'Subtyping and polymorphism in object-role modelling', *Data & Knowledge Engineering* 15, 3 (June), 251-281.
23. Halpin, T. & Proper, H. 1995, 'Database schema transformation and optimization', *OOER'95: Object-Oriented and Entity-Relationship Modeling*, Springer LNCS, 1021 (Dec.) 191-203.
24. ter Hofstede, A.1993, *Information Modelling in Data Intensive Domains*, PhD thesis, University of Nijmegen.
25. ter Hofstede, A., Proper, H. & van der Weide, T. 1993, 'Formal definition of a conceptual language for the description and manipulation of information models', *Information Systems* 18, 7 (Oct.), 489-523.
26. ISO 1982, *Concepts and Terminology for the Conceptual Schema and the Information Base*, J. van Griethuysen ed., ISO/TC97/SC5/WG3-N695 Report, ANSI, New York.
27. Jacobson, I., Booch, G. & Rumbaugh, J. 1999, *The Unified Software Development Process*, Addison-Wesley, Reading MA, USA.
28. Kobryn, C. 1999, 'UML 2001: a standardization odyssey', *Communications of the ACM*, vol. 42, no. 10, pp. 29-37.
29. Levesque, H. 1984, 'A fundamental trade-off in knowledge representation and reasoning', *Proc. CSCSI-84*, London, Ontario, 141-52.
30. Muller, R.J. 1999, *Database Design for Smarties*, Morgan Kaufmann, San Francisco, CA.
31. OMG UML Revision Task Force, *OMG Unified Modeling Language Specification*, version 1.3, available online from http://omg.org/uml/.
32. Rumbaugh, J., Jacobson, I. & Booch, G. 1999, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading MA, USA.
33. Warmer, J. & Kleppe, A. 1999, *The Object Constraint Language: precise modeling with UML*, Addison-Wesley, Reading MA, USA.
34. www.microsoft.com (online details about Visio Enterprise).