Verbalizing Business Rules: Part 13

Terry Halpin
Neumont University

Business rules should be validated by business domain experts, and hence specified in a language easily understood by business people. This is the thirteenth in a series of articles on expressing business rules formally in a high-level, textual language. The first article [3] discussed criteria for a business rules language, and verbalization of simple uniqueness and mandatory constraints on binary associations. Article two [4] examined hyphen-binding, and verbalization of internal uniqueness constraints that span a whole association, or that apply to *n*-ary associations. Article three [5] covered verbalization of basic external uniqueness constraints. Article four [6] considered relational-style verbalization of external uniqueness constraints involving nesting or long join paths, as well as attribute-style verbalization of uniqueness constraints and simple mandatory constraints. Article five [7] discussed verbalization of mandatory constraints on roles of *n*-ary associations, and disjunctive mandatory constraints (also known as inclusive-or constraints) over sets of roles. Article six [8] considered verbalization of value constraints. Article seven [9] examined verbalization of subset constraints. Article eight [10] discussed verbalization of equality constraints. Article nine [11] covered verbalization of exclusion constraints. Article ten [12] dealt with verbalization of internal frequency constraints on single roles. Article eleven [13] considered verbalization of multi-role, and external, frequency constraints. Article twelve [14] discussed verbalization of ring constraints. This article covers verbalization of basic subtype constraints.

Basic Subtype Constraints

The act of classifying an object type into one or more specific types is known as *specialization*. In Object-Role Modeling (ORM) [2] and Entity-Relationship (ER) modeling, the specialized types are said to be *subtypes* of the more general type (the *supertype*). For example, InPatient and OutPatient are subtypes of the supertype Patient. In the Unified Modeling Language (UML) [16, 17], “class” is used instead of “type”, so the terms “subclass” and “superclass” are used instead. The inverse act of introducing a more general type is known as *generalization*. Regardless of whether the inclusion of a subtype within a supertype is arrived at by means of specialization or generalization, the resulting model is the same. The inclusion relationship between a subtype and its supertype is known as a *subtyping* or *inheritance* relationship. Figure 1 depicts the subtyping relationship between InPatient and Patient in the graphical notations of (a) ORM, (b) UML, and (c) Barker ER [1].

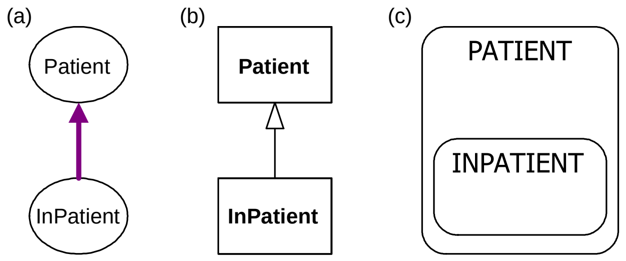

Figure 1 Declaring InPatient as a subtype of Patient in (a) ORM, (b) UML, and (c) Barker ER.

Both ORM and UML display the subtyping relationship as an *arrow* from the subtype to the supertype. With this approach, a pattern of subtype-supertype relationships forms a *directed acyclic graph*. The graph is acyclic (no loops), because only *proper subtypes* are allowed (i.e. a subtype is never identical to any of its supertypes). The Barker ER notation depicts subtyping by means of *Euler diagrams*, visually enclosing the subtype within the supertype. The subtype-to-supertype relationship is between types, not instances, so is a meta-relationship. This type inclusion relationship is a *constraint*, declaring that each instance of the subtype is also an instance of the supertype. The subtype connection in Figure 1 may be verbalized as follows:

Each InPatient **is a** Patient.

This verbalization is just a sugared version of the predicate calculus formula: $\forall x(\text{InPatient } x \rightarrow \text{Patient } x)$. In set-theory notation, this corresponds to the subset relationship InPatient $\subseteq$ Patient, although to ensure proper subtypehood we need to add the restriction InPatient $\neq$ Patient. In practice, the constraint requires that for each state of the business domain, the population of InPatient is a subset of the population of Patient.

In several versions of ER, including Barker ER, subtype classification schemes must always form a *partition* of the supertype (i.e. the subtypes are mutually exclusive, and they collectively equal the supertype). For example, InPatient and OutPatient form a partition of Patient. The mutual exclusion between the subtypes may be expressed in set-theory by declaring their intersection to be the null set (i.e. InPatient $\cap$ OutPatient = { }). The claim that the subtypes are collectively exhaustive of their supertype means that their union equals their supertype (InPatient $\cup$ OutPatient = Patient). In practice, these constraints apply to the populations of the types, for each state of the business domain.

Figure 2(a) depicts this partition in the ORM notation. Here a circled cross attached to the subtyping arrows indicates that the subtypes are exclusive, and a circled dot indicates that the subtypes are exhaustive. The two constraints are displayed together, one superimposed on the other, forming a "lifebuoy" symbol. This symbol may be considered as an xor (exclusive-or) constraint on the upper roles of the meta-fact instances corresponding to the subtyping arrows.

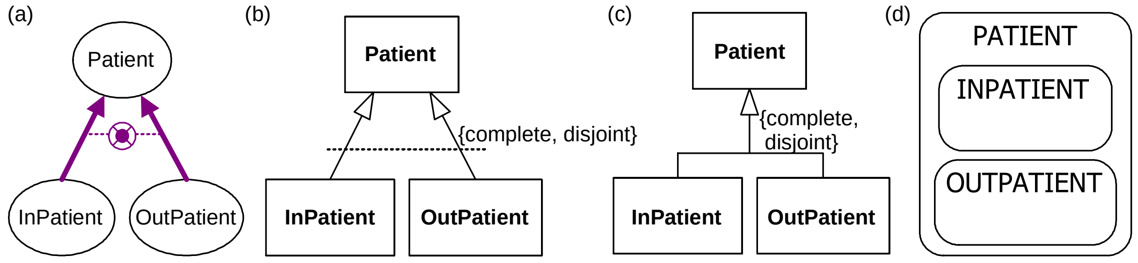

Figure 2 A Patient partition in (a) ORM, (b) UML direct style, (c) UML tree style, (d) Barker ER.

Figure 2(b) depicts the partition in UML in direct style (where each inheritance relationship appears as a separate arrow). The relevant constraints appear in braces besides a dashed line connecting the subtyping arrows: "complete" indicates that the Patient class is the union of the subclasses; "disjoint" indicates the subclasses are mutually exclusive. Figure 2(c) depicts the partition in UML in tree style (connecting the subtypes to the supertype via a single arrow head). In this case, the constraints are placed in braces besides the arrow-head.

Figure 2(d) depicts the partition in Barker ER notation. As with normal Euler diagrams, the exclusion between the subtypes is depicted by their separation (no visual overlap). The Barker ER notation always assumes that the subtypes are exhaustive (if needed, an additional "Other" subtype may be added to ensure this).

Regardless of the graphic notation used, the two constraints conveying the total (exhaustive) and exclusive nature of the subtypes in Figure 2 may be verbalized separately as follows:

Each Patient **is an** InPatient **or an** OutPatient. -- totality
No Patient **is an** InPatient **and an** OutPatient. -- exclusion

Here the use of "**an**" assumes the ability to determine that the following object type name begins with a vowel sound. If the object type name begins with a consonant sound, then "**a**" is used instead of "**an**". If such support is not provided, then "**a**" is always used. The two constraints may also be combined in a compact, single verbalization as follows:

Each Patient **is an** InPatient **or an** OutPatient **but not both**. -- partition

If more than two subtypes occur in the partition, the totality constraint may be verbalized by a simple extension to the previous pattern. For example, assuming the only possible marital states are single, married, divorced, and widowed, we have:

Each Person **is a** SinglePerson **or a** MarriedPerson **or a** WidowedPerson **or a** DivorcedPerson. -- totality

Alternatively, this constraint may be verbalized:

Each Person **is an instance of at least one of the following:**
SinglePerson; MarriedPerson; WidowedPerson; DivorcedPerson. -- totality

An exclusion constraint over more than two subtypes means pairwise-exclusion (i.e. no instance may belong to any pair of subtypes). So the earlier verbalization pattern for exclusion between two subtypes has no simple extension for this case. Instead we use the alternative pattern just discussed for totality, but replace "**least**" by "**most**". For example, assuming the four marital states just mentioned are mutually exclusive, we have:

Each Person **is an instance of at most one of the following:**
SinglePerson; MarriedPerson; WidowedPerson; DivorcedPerson. -- exclusion

The two constraints may be combined into a single verbalization as follows:

Each Person **is an instance of exactly one of the following:**
SinglePerson; MarriedPerson; WidowedPerson; DivorcedPerson. -- partition

These patterns for the case of more than two subtypes may also be used for the simpler case of two subtypes.

In ORM, the absence of an explicit or implicit exclusion constraint between subtypes means the subtypes may overlap (i.e. have some instances in common). In UML, this situation is depicted by the annotation "{overlapping}". In ORM, the absence of an explicit or implicit totality (exhaustion, i.e. inclusive-or) constraint between subtypes means the union of the subtype populations need not equal the supertype population. In UML, this situation is depicted by the annotation "{incomplete}". Figure 3 illustrates this practice using equivalent examples in ORM and UML taken from [2].

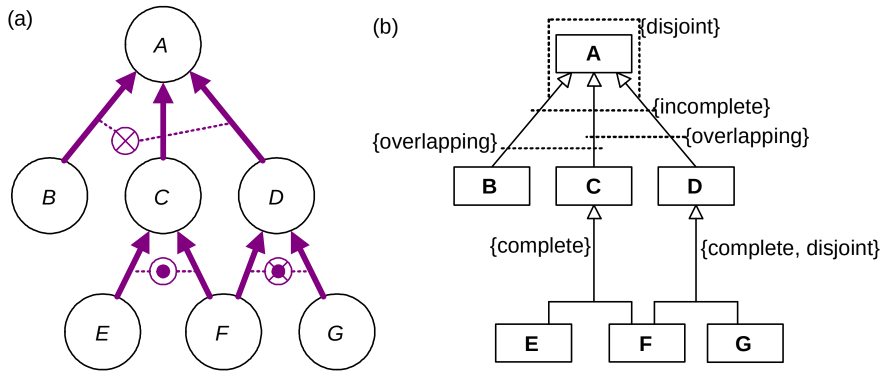

Figure 3 Equivalent subtyping patterns in (a) ORM and (b) UML.

Barker ER does not cater for cases like this. As a simple example, consider a case of *multiple inheritance* as shown in Figure 4. Here FemaleInPatient is a subtype of the overlapping FemalePatient and InPatient types. While such cases are supported directly in ORM and UML, Barker ER requires partitions for all subtype classifications, so the only possible approach it could adopt would be to use two separate diagrams for the two partitions; but even then it is unable to include FemaleInPatient as subtypes of both FemalePatient and InPatient. This is the main problem with using Euler diagrams for subtyping, especially the restricted version of Euler diagrams supported in Barker ER: for simple partitions they are more visually intuitive than directed acyclic graphs, but for more complex cases they are inadequate or unwieldy. For a related example showing problems with unrestricted Euler diagrams, see [2 , p. 246].

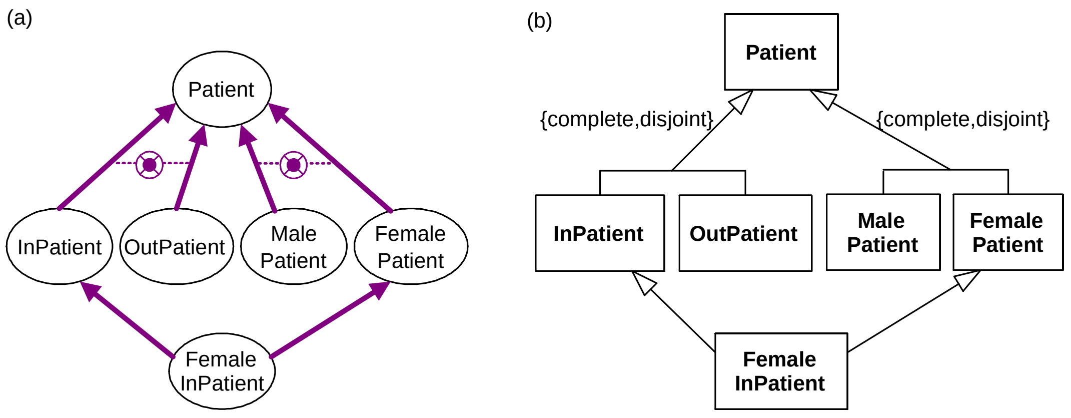

Figure 4 A case of multiple inheritance in (a) ORM and (b) UML.

Returning to UML's "{overlapping}" and "{incomplete}" annotations, these are not really constraints at all (they instead indicate lack of constraints). So they have no positive verbalization. If desired however, a default verbalization could be provided to spell out the implications of the absence of a subtype totality or exclusion constraint. For example, suppose that for some reason we wished to include Australian and MalePerson as subtypes of Person, as shown in Figure 5. These subtypes overlap, since it is possible to have male Australians; and the subtypes are incomplete since there may be instances of Person that belong to neither subtype (e.g. an American woman).

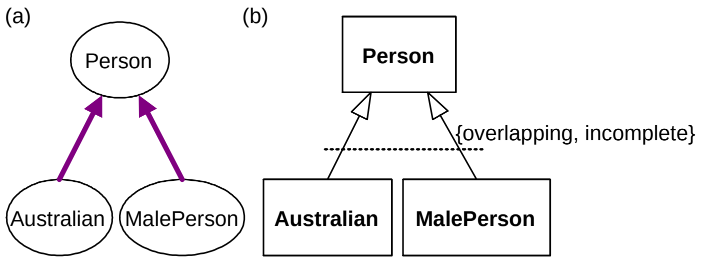

Figure 5 Overlapping, incomplete subtypes in (a) ORM, and (b) UML.

For this example, the absence of completeness and exclusion constraints may be explicitly declared by the following default verbalizations.

It is possible that some Person **is an instance of more than one of the following:**
Australian; MalePerson. -- overlap

It is possible that some Person **is an instance of none of the following:**
Australian; MalePerson. -- incomplete

That completes our coverage of basic subtype constraints. It turns out that these constraints provide only an incomplete coverage of the constraints that may apply in realistic subtyping situations. To handle more complex cases, it is necessary to provide formal support for subtype definitions. Verbalization of subtype definitions is addressed in the next article.

References

1. Barker, R. 1990, *CASE*Method: Tasks and Deliverables*, Addison-Wesley, Wokingham, England.
2. Halpin, T. A. 2001, *Information Modeling and Relational Databases*, Morgan Kaufmann, San Francisco.
3. Halpin, T. A. 2003, ‘Verbalizing Business Rules: Part 1’, *Business Rules Journal*, Vol. 4, No. 4 (April 2003), URL: http://www.BRCommunity.com/a2003/b138.html.
4. Halpin, T. A. 2003, ‘Verbalizing Business Rules: Part 2’, *Business Rules Journal*, Vol. 4, No. 6 (June 2003), URL: http://www.BRCommunity.com/a2003/b152.html.
5. Halpin, T. A. 2003, ‘Verbalizing Business Rules: Part 3’, *Business Rules Journal*, Vol. 4, No. 8 (August 2003), URL: http://www.BRCommunity.com/a2003/b163.html.
6. Halpin, T. A. 2003, ‘Verbalizing Business Rules: Part 4’, *Business Rules Journal*, Vol. 4, No. 10 (October 2003), URL: http://www.BRCommunity.com/a2003/b172.html.
7. Halpin, T. A. 2004, ‘Verbalizing Business Rules: Part 5’, *Business Rules Journal*, Vol. 5, No. 2 (February 2004), URL: http://www.BRCommunity.com/a2004/b179.html.
8. Halpin, T. A. 2004, ‘Verbalizing Business Rules: Part 6’, *Business Rules Journal*, Vol. 5, No. 4 (April 2004), URL: http://www.BRCommunity.com/a2004/b183.html.
9. Halpin, T. A. 2004, ‘Verbalizing Business Rules: Part 7’, *Business Rules Journal*, Vol. 5, No. 7 (July, 2004), URL: http://www.BRCommunity.com/a2004/b198.html.
10. Halpin, T. A. 2004, ‘Verbalizing Business Rules: Part 8’, *Business Rules Journal*, Vol. 5, No. 9 (September, 2004), URL: http://www.BRCommunity.com/a2004/b205.html.
11. Halpin, T. A. 2004, ‘Verbalizing Business Rules: Part 9’, *Business Rules Journal*, Vol. 5, No. 12 (December, 2004), URL: http://www.BRCommunity.com/a2004/b215.html.
12. Halpin, T. A. 2005, ‘Verbalizing Business Rules: Part 10’, *Business Rules Journal*, Vol. 6, No. 4 (April, 2005), URL: http://www.BRCommunity.com/a2005/b229.html.
13. Halpin, T. A. 2005, ‘Verbalizing Business Rules: Part 11’, *Business Rules Journal*, Vol. 6, No. 6 (June 2005), URL: http://www.BRCommunity.com/a2005/b238.html.
14. Halpin, T. A. 2005, ‘Verbalizing Business Rules: Part 12’, *Business Rules Journal*, Vol. 6, No. 10 (October 2005), URL: http://www.BRCommunity.com/a2005/b252.html.
15. Halpin, T., Evans, K., Hallock, P. & MacLean, B. 2003, *Database Modeling with Microsoft Visio for Enterprise Architects*, Morgan Kaufmann, San Francisco.
16. Object Management Group 2003, *UML 2.0 Infrastructure*, URL: http://www.omg.org/uml.
17. Object Management Group 2003, *UML 2.0 Object Constraint Language*, URL: http://www.omg.org/uml.