Business Rule Verbalization

Terry Halpin

Northface University
Salt Lake City, USA
terry.halpin@northface.edu

Abstract: Business rules should first be specified at the conceptual level, using concepts and languages easily understood by the business domain experts who are best qualified to validate the rules. This paper focuses on the verbalization of static business rules (i.e. constraints and derivation rules that apply to individual states of the business), ignoring dynamic rules relating to state transitions and/or workflows. We propose desirable language criteria for rule verbalization, and orthogonal dimensions by which rule verbalizations may be classified, and illustrate these general qualities with a wide variety of business rule examples, using model fragments depicted graphically in Entity-Relationship, Object-Role Modeling and Unified Modeling Language notations.

1 Introduction

In practical modeling of information systems, one often encounters business rules that go beyond simple rules like multiplicity constraints. Although business rules may be implemented in many ways, they should first be specified at the conceptual level, using concepts and language easily understood by the business domain experts who are best qualified to validate the rules. These subject matter experts understand the business domain even if they are unfamiliar with technical notations used by professional analysts.

Business rules come in many varieties, and may be specified using graphical and/or textual languages. We regard a *static business rule* to be a constraint or derivation rule that applies to each individual state of the business, taken one state at a time. This paper focuses on the *verbalization* of static business rules, ignoring dynamic rules relating to state transitions or workflows. The rule verbalization techniques and patterns are illustrated using practical examples. While the main verbalization approach discussed in this paper is based on extensions to Object-Role Modeling (ORM), many of the techniques may be adapted to other popular modeling notations, including Entity Relationship (ER), and Unified Modeling Language (UML) notations. For detailed background on ORM, see [Ha01]. For detailed coverage of UML, see [RJB99, Om03a, Om03b, Om03c]. Comparative reviews of ORM, ER, and UML are found in [Ha02a, Ha02b].

Section 2 overviews language criteria and dimensions for business rule verbalization. Section 3 illustrates these principles by verbalizing multiplicity constraints on single binary associations. Section 4 considers verbalization of related constraints on n-ary

associations and multiple associations. Section 5 discusses verbalization of set-comparison, ring, and other constraints. Section 6 considers verbalization of derivation rules. Section 7 reviews the main contributions, and suggests areas for future research.

2 Criteria for Business Rule Verbalization

We begin with some terminology. A *fact instance* is a proposition taken to be true, and is expressed by a declarative sentence (e.g. "The Island named 'Honshū' is part of the Country named 'Japan'."), consisting of a *predicate* symbol (e.g. … is part of …) applied to a sequence of one or more object terms (e.g. "The Island named 'Honshū'", "The Country named 'Japan'"). An *object type* is a kind of object, e.g. Island, Country. A *fact type* includes predicate and object type(s) but not instances, e.g. Island is part of Country. A *fact-role* is a part played by an object in a predicate. In the context of a predicate, a fact-role is simply called a *role*. The number of roles in a predicate is called its *arity*.

Static business rules are best applied to a fact model that identifies the fact types of interest to the business. Table 1 shows some fact types with arities from 1 to 4. Each role corresponds to an object placeholder (depicted here as an ellipsis "…") in the predicate. Here predicates are displayed in *mixfix* notation, allowing object terms to be placed in a sentence at any position. Higher order predicates (quinary etc.) are also possible.

Table 1 Examples of fact types of different arity

Fact Type	*Predicate*	*Arity*
Person smokes	… smokes	1 (Unary)
Person was born in Country	… was born in …	2 (Binary)
Person played Sport for Country	… played … for …	3 (Ternary)
Person introduced Person to Person on Date	… introduced … to … on …	4 (Quaternary)

Business rules may be specified graphically on a business model diagram, and/or textually using a language for rule verbalization. In [Ha01, sec. 3.1], we suggested the following criteria for evaluating the suitability of a conceptual modeling language: expressibility (what can be expressed?); clarity (ease of understanding); simplicity and orthogonality (avoid unneeded concepts, minimize inter-concept dependencies, allow expressions wherever their values are legal); semantic stability (minimize the impact of change); semantic relevance (ignore implementation details); validation mechanisms (support ways for domain experts to validate the model); abstraction mechanisms (provide ways of hiding detail); and formal foundation (logical grounding). For a textual language used to express business rules, the following criteria seem to be the most essential. The flexibility and localizability criteria are implied by the clarity criterion.

- *expressibility* the language is able to express a wide range of business rules
- *clarity* the rules are understandable by non-technical domain experts
- *flexibility* the language directly supports predicates of any arity
- *localizability* the language constructs are expressible in different native languages
- *formality* the rules are unambiguous, and should ideally be executable

Supporting predicates of any arity allows domain experts to verbalize the rules directly in terms of the way they think. This makes it much easier for them to understand and validate the rules. It is always possible to reformulate unary, ternary, and higher arity predicates in terms of binary predicates, but this can make it harder for the subject matter experts to comprehend the rules. Some rules are hard enough to get a grip on in the first place. There is no excuse for imposing on the domain experts a binary straightjacket that forces them to recast a fact type they would normally verbalize as a quaternary in terms of a binarized equivalent (e.g. a binary nested within a binary nested with a binary). Similarly, we should not force domain experts to recast unary facts such as Country 'US' is large in terms of Boolean attribute assignments (e.g. US : Country.isLarge = True).

Once we accept that predicates of any arity must be supported, using natural verbalizations, it follows that *mixfix* notation must be supported. In Table 1, the "... smokes" predicate is prefix, and the "... was born in ..." predicate is infix, but the "... played ... for ..." and " ... introduced ... to ... on ..." predicates require mixfix because they have more than two object placeholders. Although prefix or postfix notation could be used, such as PlayedFor(X, Y, Z), this would be unnatural for non-technical people.

Localizability provides yet another reason for mixfix notation, since even for binary predicates, the verb-phrase need not occur in the infix position. For example, verbs are usually placed last in Japanese, and are usually placed first in Tongan. Some years ago, we specified a verbalizer component that is currently implemented in a database modeling tool [Ha03]. The verbalizer automatically translates ORM graphical constraints into English, as well as Japanese, German, and French in the localized versions. In this paper, we extend and refine this original work in several directions, focusing on business rule textual languages that are based on English.

Table 2 provides an overview of *classification schemes for rule verbalization* that we devised based on our industrial modeling experience, where these dimensions proved to be pragmatically useful in characterizing ways in which business rules may be textually formulated for validation with domain experts.

An informal verbalization of business rules may be useful, so long as it is unambig-uous to the domain expert who has to validate the rules. The RuleSpeak sentence templates [RL01] provide one way to do this. To exploit the benefits of model-driven development however, business rules should be expressed in a *formal* language, so that they can be automatically transformed into executable code. The OSA model [EKW92, Em98] supports high level, informal rules as well as formal rules in a predicate calculus notation. Our approach instead uses a single language that is both formal and conceptual, so that it can serve for communication and validation with domain experts, as well as being executable. The rest of the paper focuses on formal, high level languages, and discusses other dimensions (form, modality, style, and context) with the aid of examples.

Table 2 Classification schemes for rule verbalization

Dimension	*Choices*
Form	Positive Negative Default
Modality	Alethic Deontic
Style	Relational Attribute
Context	Local Global
Formality	Informal Semi-formal Formal

3 Multiplicity Constraints on Binary Associations

Figure 1 graphically depicts a binary relationship (orbiting) between two entity types (Moon and Planet) using (a) the ER notation popularized by Richard Barker [Ba90], (b) the UML notation, (c) the ORM notation, and (d) a simple Fact-model notation. Here the fact model notation shows the binary fact type without constraints (these are specified separately). The other notations display both the fact type and its associated constraints. The constraints indicate that each moon orbits exactly one (at least one and at most one) planet, and that a planet is orbited by zero or more moons.

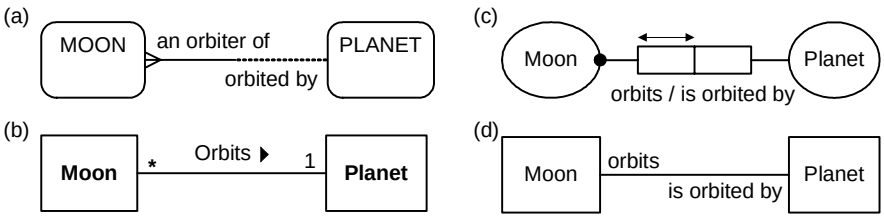

Fig. 1 An n:1 association in (a) Barker ER, (b) UML, (c) ORM, and (d) Fact-model notation

To specify that each moon orbits at least one planet, both Barker ER and ORM use a *mandatory role constraint*. Barker ER depicts a role as one half of a relationship line, whereas ORM uses a role box. To indicate that a role is mandatory or optional, Barker ER uses a solid or dotted line respectively. In ORM, a role is optional unless it has been assigned a mandatory constraint (depicted graphically as a solid dot).

To indicate the n:1 cardinality of the Moon orbits Planet relationship, Barker ER uses a single line end for "one" and crows-foot for "many". ORM instead uses a *uniqueness constraint* depicted as an arrow-tipped bar over the role(s) to which it applies. As shown later, ORM allows populated fact tables to be associated with fact types, with each role corresponding to a table column. A uniqueness constraint over a role (or role set)

indicates that each instance in the population of that role (or role set) is unique. The lack of a uniqueness constraint indicates that an instance may occur there more than once.

Unlike Barker ER and ORM, the UML approach combines the separate concepts of mandatory and cardinality/uniqueness constraints into a single multiplicity constraint (e.g. '0..1' for at most one, '1' or '1..1' for exactly one, and '0..*' or '*' for zero or more).

Of the four approaches, only Barker ER and ORM provide standard support for constraint verbalization. To enable association optionality and cardinality settings to be verbalized, Barker recommends the following naming discipline for relationships [Ba90, p. 3-5]. 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 optionality/cardinality settings in Figure 1(a) verbalize as:

Each Moon must be an orbiter of one and only one Planet.
Each Planet may be orbited by one or more Moons.

This verbalization convention is good for basic constraints on infix binaries. If the user follows the naming discipline, and pluralization can be automated (e.g. House to/from Houses, Mouse to/from Mice), these rules may be regarded as formal and executable. Overall however, the Barker ER approach is too restrictive, because it handles only binary associations, and while it does include a few other graphical constraints, it provides no rules for verbalizing these in a textual language. In contrast, ORM applies to predicates of any arity, and covers many more kinds of constraint both graphically and textually, with no need for pluralization.

Let's see how an ORM textual language might cater for these constraints. The orbital fact type is reproduced in Figure 2, along with another fact type about planets. Reference schemes for each object type are shown in parenthesis, and sample fact tables are also provided.

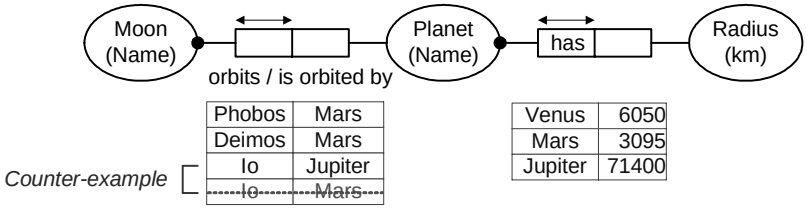

Fig. 2 Fact types with satisfying and counter populations

The first three rows of the orbital fact table provide a significant fact population with respect to uniqueness constraints: its first column values are unique, and its second column contains duplicates, clearly showing the n:1 nature of the association. The planetary radius table is only partly significant with respect to uniqueness constraints, because its column values are unique for both roles, suggesting a 1:1 association. A domain expert however can confirm that it is possible for two planets to have the same radius.

The ORM approach encourages rule *validation by verbalization and population*. A verbalization in *positive form* stresses what is required to *satisfy* the rule, so it can be exemplified by satisfying fact population (a set of fact instances consistent with the rule). A verbalization in *negative form* stresses what is required to *violate* the rule, so it can tested using *counter-examples* that illustrate the rule being violated. The uniqueness constraint on the orbital fact type may be verbalized in positive and negative forms thus:

Each Moon orbits at most one Planet. -- positive form
It is impossible that the same Moon orbits more than one Planet. -- negative form

The positive form of the constraint is illustrated by the satisfying population on rows 1-3 of the orbital fact table. The textual rule introduces the *quantifiers* each ($\forall$), and at most one ($\exists^{0..1}$), but leaves the fact type verbalization unaltered.

The negative form of the constraint is illustrated by the counter-example population on rows 3-4, where the same moon (Io) orbits two planets (Jupiter and Mars). This duplicate value in the first column violates the uniqueness constraint on that column. Use of counter-examples is a powerful way to check whether a constraint really does apply.

The *mandatory constraint* on the first role of Moon orbits Planet may be verbalized thus:

Each Moon orbits some Planet. -- positive form
Each Moon orbits at least one Planet. -- positive form (alternative)
It is impossible that some Moon orbits no Planet -- negative form (long version)
No Moon orbits no Planet. -- negative form (short version)

A verbalization in *default form* states what is allowed in the absence of an explicit assertion of a rule. For example, the absence of a uniqueness constraint on the second role of Moon orbits Planet may be verbalized explicitly thus:

It is possible that the same Planet is orbited by more than one Moon. -- default

The population illustrates this possibility (Mars has two moons). The absence of a mandatory constraint on the second role of Moon orbits Planet may be stated explicitly as below. The combined population of the two tables in Figure 2 illustrates this possibility (Venus has no moons).

It is possible that some Planet is orbited by no Moon -- default

If a role has both a simple uniqueness and simple mandatory constraint, both constraints may be combined in a single verbalization by using "exactly one" to abbreviate "at least one and at most one". For example, the constraints on the orbital fact type may be succinctly stated thus: Each Moon orbits exactly one Planet.

Constraints on 1:1 binaries are verbalized using combinations of constraint verbalizations already discussed. A many-to-many association is shown in Figure 3. In Barker ER, this is verbalized as: Each Person may be a visitor of one or more Countries, and each Country may be visited by one or more Persons. In ORM, this is depicted as a uniqueness constraint that spans both the roles, as reflected by the sample population (only person-country pairs are unique).

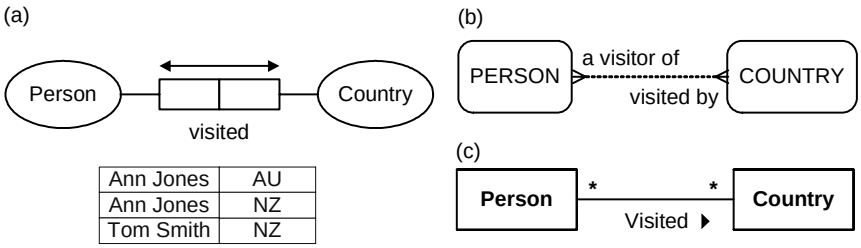

Fig. 3 A many-to-many association in (a) ORM, (b) Barker ER, and (c) UML

The standard verbalization in ORM for this spanning uniqueness constraint is shown below. This reflects the absence of a single role uniqueness constraint on the first role, and the absence of a single role uniqueness constraint on the second role.

> It is possible that the same Person visited more than one Country
> and that the same Country was visited by more than one Person.

While the above verbalizations clarify the many-to-many nature of the association, they do not explicitly capture the further intention that person-country pairs must be unique. In other words, the population of the association must be a *set* of person-country pairs, not a bag (multi-set). For example, the spanning uniqueness constraint would be violated if we added the pair "Tom Smith, NZ" as a fourth row of the fact table.

This further aspect of the constraint is usually left implicit, because it is assumed that the populations of *all* primitive (non-derived) associations must be sets, not bags. If we wish to make this assumption explicit for any given association, we may add a verbalization to that effect. For our current example, this may be verbalized as follows.

> Each instance of Person visited Country occurs only once.

As good modeling practice, *any spanning uniqueness constraint should be checked with the domain expert to confirm that a bag is not required*. For example, a person may visit a country more than once. If we wish to record whether (but not how often) a person visited a country, then we may use the fact type Person visited Country without further qualification. If however we wish to know how often a person visited a country, we must modify the model. For example, we might objectify the association Person visited Country as Visit, and then add a many-to-many, temporal fact type such as Visit began on Date.

4 Constraints on N-ary and Multiple Associations

While industrial ER is restricted to binary associations, both ORM and UML support *n*-ary associations ($n > 2$). For example, Figure 4 depicts the ternary fact type Room at HourSlot is booked for Course. The ORM model includes two uniqueness constraints, each spanning two roles, as well as a satisfying population. The UML model includes equivalent multiplicity constraints.

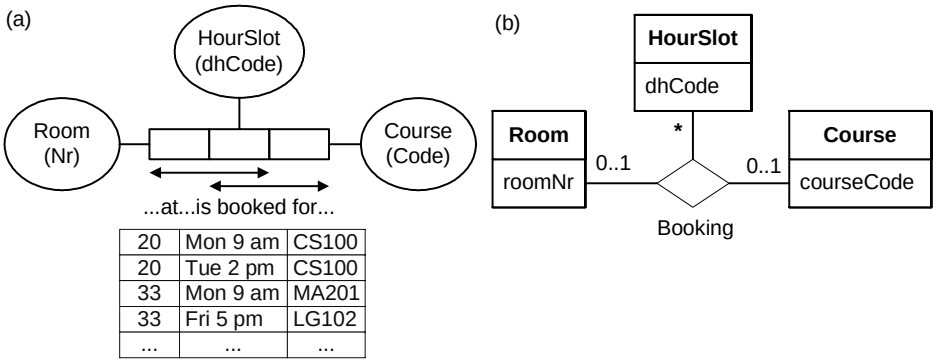

Fig. 4 Uniqueness constraints on a ternary association in (a) ORM and (b) UML

The ORM uniqueness constraint spanning the first two roles of the association may be expressed in positive form thus:

Given any Room and HourSlot
    that Room at that HourSlot is booked for at most one Course

Every constraint has an associated *modality*, determined by the logical modal operator that functions explicitly or implicitly as its main operator. For a simple introduction to modal logic, see [Gi00]. In practice, the modality is typically either *alethic* or *deontic* (see Table 3). Logical negation may be used to obtain the usual equivalences (e.g. not necessary ≡ possible, not obligatory ≡ permitted, not permitted ≡ forbidden).

Table 3 Alethic and deontic modal operators

Alethic	*Deontic*
It is necessary that	It is obligatory that
It is possible that	It is permitted that
It is impossible that	It is forbidden that

In positive verbalizations, the modality is often assumed, but may be explicitly prepended. For the constraint just considered, we have:

It is necessary that -- alethic
given any Room and HourSlot
    that Room at that HourSlot is booked for at most one Course

It is obligatory that -- deontic

given any Room and HourSlot
    that Room at that HourSlot is booked for at most one Course

The deontic version indicates that the rule *ought* to be obeyed, while recognizing that the constraint *might* be violated. Most business constraints are deontic in nature. Negative verbalizations typically make the modality explicit. For example, the same constraint may be verbalized thus:

It is impossible that -- alethic
the same Room at the same HourSlot
    is booked for more than one Course

It is forbidden that -- deontic
the same Room at the same HourSlot
    is booked for more than one Course

In Figure 4, two role pairs have a uniqueness constraint. The absence of a uniqueness constraint on the other role pair (room, course) may be verbalized in default form thus:

It is possible that -- alethic
the same Room at more than one HourSlot
    is booked for the same Course

It is permitted that -- deontic
the same Room at more than one HourSlot
    is booked for the same Course

Suppose that we modify the schema fragment in Figure 4 by requiring that each course has a room-hourslot booked for it. In ORM, this additional constraint may be specified graphically by applying a mandatory role dot to the role played by Course, and verbalized as follows ("some", "a", and "an" are alternatives to "at least one").

For each Course
some Room at some HourSlot is booked for that Course

This constraint cannot be specified graphically in UML, because its multiplicity constraint does not scale properly for n-ary associations. See [Ha02a] for a detailed discussion of this and other problems with UML.

If the same object type plays more than one role in an *n*-ary association, the use of the pronoun "that" in verbalizing a constraint may lead to ambiguity. For example, Figure 5(a) shows the quaternary association Person judged Person to have SkillLevel in Language. In UML, this would typically be modeled using a ternary association class with a skill level attribute, as shown in Figure 5(b).

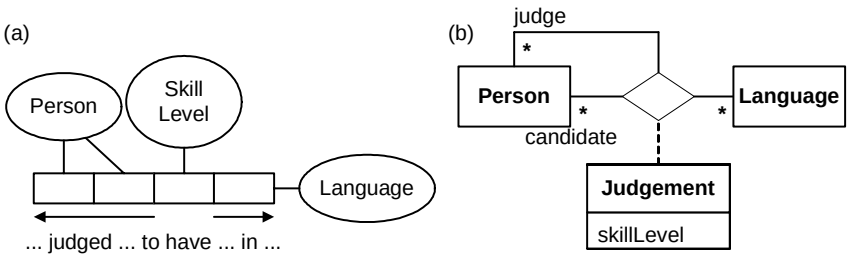

Fig. 5 An association with more than one role played by the same object type

Applying the earlier verbalization pattern leads to ambiguous references for "that Person". In such cases, it is convenient to append numbered *subscripts* to distinguish the object variables [BH97]. The constraint may now be verbalized clearly, as follows.

Given any $Person_1$, $Person_2$ and Language
$Person_1$ judged $Person_2$ to have at most one SkillLevel in that Language.

We now turn to *constraints that apply to multiple fact-roles*. Figure 6 shows a model fragment depicted in (a) Barker ER, and (b) ORM notations. Here, Room has a composite identification scheme. Each room is identified by combining its local room number with the building in which it resides. In Barker ER, this is captured by flagging the room number attribute (with a "#") and the room containment role (with a "|") as identifier components. In ORM, this is captured by the *external uniqueness constraint* (circled "u") over the relevant roles played by Building and RoomNr. UML has no standard support for value-based identification (simple or composite), so is ignored for this example. The external uniqueness constraint may be verbalized in positive form thus:

For each Building and RoomNr
there is at most one Room that is in that Building and has that RoomNr.

In negative form, with deontic modality, the same constraint verbalizes thus:

It is forbidden that
more than one Room is in the same Building and has the same RoomNr.

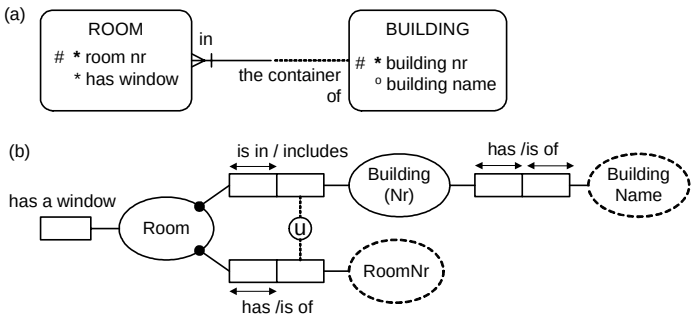

Fig. 6 Room has a composite identification scheme

Both UML and ORM allow associations to be *objectified* as object types. In Figure 7, the association Country plays Sport is objectified as Play. This is called an association class in UML. The external uniqueness constraint in Figure 7(a) captures the no-ties rule expressed informally as a note in Figure 7(b).

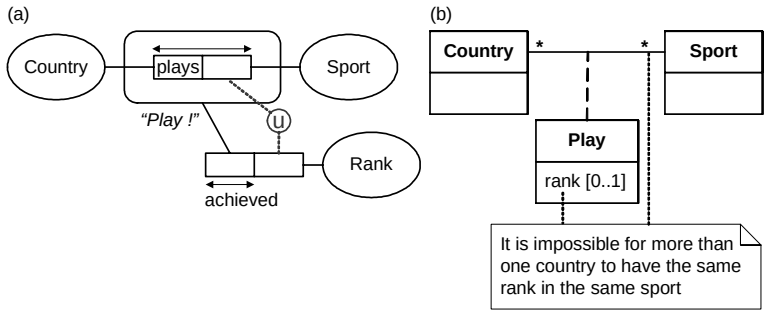

Fig. 7 A no-ties constraint in (a) ORM and (b) UML

The constraint may be verbalized using earlier patterns (e.g. For each Sport and Rank, at most one Country that plays that Sport achieved that Rank). However this pattern does not generalize to all possible cases of objectification. In this and many other cases it is convenient to use a more general pattern based on *contextual* specification that pre-declares the *context* in which the constraint is to be interpreted. For example:

Context: Country plays Sport is objectified as Play.
Play achieved Rank.

Constraint: For each Sport, Rank combination
there is at most one Country.

5 Further Constraints

The ORM graphical notation covers many more kinds of static constraint than does ER or UML. This section briefly considers verbalization of a few of these. Consider the ORM model diagrammed in Figure 8. Here the "^{O}ir" depicts an *irreflexive constraint*, which may be verbalized as: No Movie is based on itself. The circled "⊆" denotes a *subset constraint*: Each Movie that was reviewed by a Person also was directed by a Person. The circled "×" depicts an *exclusion constraint* between the associations: No Person directed and reviewed the same Movie.

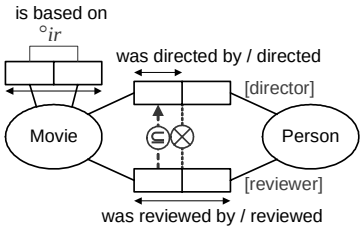

Fig. 8 Irreflexive, subset, and exclusion constraints

While ORM requires at least one reading for each fact type, it also allows fact-roles to be named. In Figure 8, two role names are displayed in square brackets besides their role. Up to now, all of our rules have been verbalized in *relational style*, using a reading for the fact type (relationship type). Role names may be used to provide alternative verbalizations in *attribute style*. Attribute style is especially useful for notations like ER and UML, which express some fact types as attributes. The exclusion constraint above may be verbalized in attribute style thus:

For each Movie
    no director is a reviewer.

The subset constraint in Figure 9 runs from the set of advisor-country pairs instantiating the serves-in association to the set of advisor-country pairs projected from a schema path that *conceptually joins* the Language roles. This constraint may be verbalized thus: Each Advisor who serves in a Country also speaks a Language that is used by that Country.

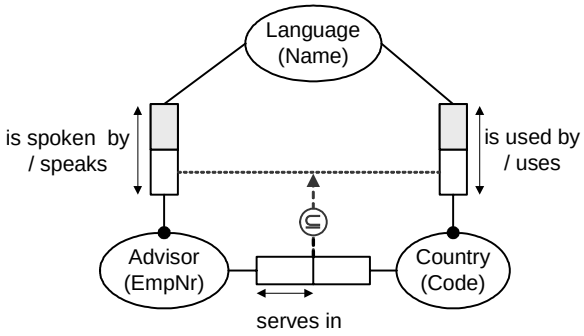

Fig. 9 A join-subset constraint

6 Derivation Rules

Derivation rules may be specified to derive an object type or a fact type from other types. For example, given the fact types City is in Country and City has Population we may define the *subtype* LargeUSCity thus: Each LargeUSCity is a City that is in Country 'US' and has Population >= 1000000. Figure 10 provides a simple example of a *derived fact type* in UML and ORM. The uncle-of association is derivable, as indicated by a slash in UML and an asterisk in ORM. The UML model specifies an attribute style derivation rule for this association in the Object Constraint Language (OCL), which for once is readable. The ORM model verbalizes the derivation rule in relational style. Using role names, the ORM model may also verbalize the rule in attribute style thus: For each Person, uncle = brother of parent.

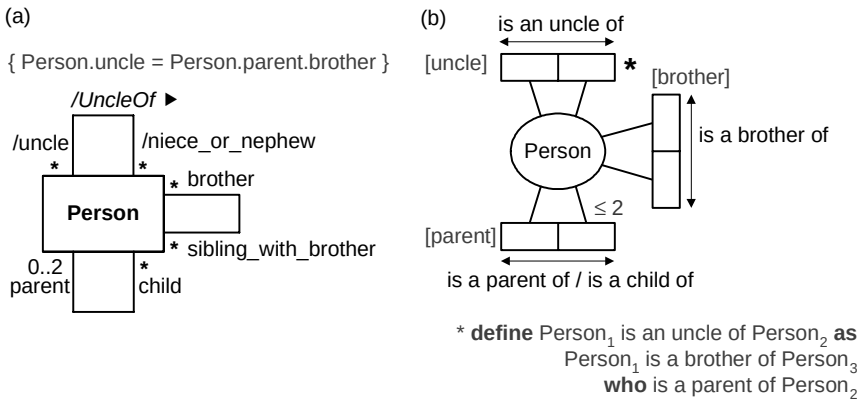

Fig. 10 A derived fact type in (a) UML, and (b) ORM.

Attribute style derivation rules are especially useful for arithmetic derivation, as shown in Figure 11.

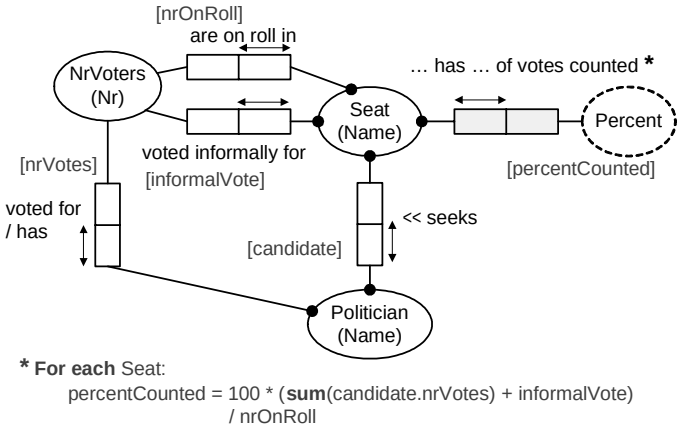

Fig. 11 Attribute style is usually best for arithmetic derivation rules

7 Conclusion

In this paper we proposed several criteria for verbalizing business rules, and illustrated the these principles using verbalization patterns that enable a wide range of static constraints and derivation rules to be specified in a natural, yet formal way, serving the dual purpose of rule validation with domain experts while allowing constraint enforcement code to be automatically generated from the high level verbalization.

In practice, far more complex business rules sometimes apply, in both static and dynamic arenas. Space limitations preclude a detailed coverage of our research on complex rules (e.g. advanced aspects of schema paths over n-ary and nested predicates), but some of our recent work on complex rules may be found in [HW03]. A challenging

research objective is to extend the executable verbalization patterns to cover almost all business rules that occur in typical business domains. In conjunction with Bloesch, we developed a conceptual query language capable of expressing arbitrary first order queries, with automatic code generation [BH97]. Query languages may also be used to specify constraints and derivation rules. One future research objective is to unify aspects of conceptual query technology with our work on rule verbalization, to help realize the dream of very high level executable specifications in the business rules area.

Bibliography

[Ba90] Barker, R.: CASE*Method: Tasks and Deliverables. Addison-Wesley, Wokingham, England, 1990.

[BH97] Bloesch, A; Halpin, T.: Conceptual queries using ConQuer-II. In (Embley, D.; Goldstein, R. Eds.): Proc. 16th Int. Conf. on Conceptual Modeling ER'97, Los Angeles, Springer LNCS 1331, 1997; pp. 113-126. Online at www.orm.net/pdf/ER97-final.pdf.

[EKW92] Embley, D.; Kurtz, B.; Woodfield, S.: Object-Oriented Systems Analysis: A Model-Driven Approach. Prentice Hall, Englewood Cliffs, NJ, 1992.

[Em98] Embley, D.: Object Database Management. Addison-Wesley, Reading, MA, 1998.

[Gi00] Girle, R.: Modal Logics and Philosophy. McGill-Queen's University Press, Montreal & Kingston. 2000.

[Ha01] Halpin, T.: Information Modeling and Relational Databases. Morgan Kaufmann, San Francisco, 2001.

[Ha02a] Halpin, T.: Information Analysis in UML and ORM: a Comparison. In (Siau, K Ed.): Advanced Topics in Database Research, vol. 1, Idea Publishing Group, Hershey PA, USA, 2002; pp 307-323.

[Ha02b] Halpin, T.: Metaschemas for ER, ORM and UML Data Models: A Comparison. In (Siau, K Ed.): Journal of Database Management, vol. 13, no. 2, 2002; pp. 20-29, Idea Publishing Group.

[Ha03] Halpin, T.; Evans, K.; Hallock, P.; MacLean, W.: Database Modeling with Microsoft® Visio for Enterprise Architects. Morgan Kaufmann, San Francisco, 2003.

[Ha04] Halpin, T.: Constraints on Conceptual Join Paths. In (Krogstie, J.; Halpin, T.; Siau, K. Eds.): Information Modeling Methods and Methodologies, Idea Publishing Group, Hershey, 2004.

[HW03] Halpin, T.; Wagner, G.: Modeling Reactive Behavior in ORM. In: Conceptual Modeling – ER2003, Proc. 22nd ER Conference, Chicago, October 2003, Springer LNCS.

[Om03a] Object Management Group: UML 2.0 Infrastructure Specification, 2003. Online at: www.omg.org/uml.

[Om03b] Object Management Group: UML 2.0 Object Constraint Language, 2003. Online at: www.omg.org/uml.

[Om03c] Object Management Group: UML 2.0 Superstructure Specification, 2003. Online at: www.omg.org/uml.

[RL01] Ross, R.; Lam, G.: RuleSpeak Sentence Templates: Developing Rules Statements Using Sentence Patterns.. Business Rule Solutions, Online at www.BRCommunity.com, 2001.

[RJB99] Rumbaugh, J.; Jacobson, I.; Booch, G.: The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

[WK03] Warmer, J.; Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA, Second Edition. Addison-Wesley, 2003.