Temporal Modeling: Part 4

Terry Halpin
Neumont University

This is the fourth in a series of articles on the impact of *time* on the conceptual modeling of business domains. The first article [11] discussed the temporal data types *instant* (point in time), *interval* (duration of time), and *period* (anchored duration of time), classified *temporal object types* into *once-only* (e.g., Date) and *repeatable* (e.g., WeekDay) object types, and discussed *four kinds of fact type*: *definitional* (truth of instances is a matter of definition), *once-only* (instances correspond to a single event), *repeatable* (instances may correspond to multiple events) and *time-deictic* (the meaning of instances depends on the time of utterance/inscription). It then showed how to model temporal details about point events or period events underlying instances of once-only fact types that are unchangeable. The second article [12] examined the modeling of temporal information about events underlying *changeable fact types* (their nonnull fact populations may change over time, by replacing, adding, or deleting facts) that are initially *functional* (*n*:1 or 1:1 associations). The third article [13] discussed how to maintain history of *changeable fact types* that are *nonfunctional* (e.g. *m:n* binaries, or higher arity fact types).

This fourth article provides yet another way in UML 2 to maintain history of nonfunctional, changeable fact types, and then discusses *rigid subtypes and role subtypes,* and related dynamic constraints. Three graphical notations are used for examples: second generation Object-Role Modeling (ORM 2) [8, 9] as supported by the open source (Neumont ORM Architect) NORMA tool [4, 15]; the Unified Modeling Language (UML) [16]; and the Barker notation [1] for Entity-Relationship Modeling (ER) [3].

Another Way to Maintain History of Nonfunctional, Changeable Fact Types in UML 2

The previous article [13] discussed three ways to maintain history of nonfunctional, changeable fact types, such as Employee visited Country: include a distinguishing temporal role as part of the identifier; introduce a simple, visible identifier; or introduce an ordinal number as part of the identifier. As noted by Artur Kasprzyk in a recent e-mail to me, a fourth option available in UML 2 is to tag the association ends of the repeatable fact type with {nonunique}, thereby making it a bag association.

To understand how this works, let's first consider the usual situation where a fact type can be populated only by a set of facts rather than a bag (multiset) of facts. In the ORM schema in Figure 2(a), the uniqueness constraint on the Employee visited Country fact type is depicted as a bar spanning both roles of the fact type. This constraint indicates that the association is many to many, and unique. So when we populate the association with sample facts, we allow the same employee to visit many countries, and the same country to be visited by many employees, but each (employee, country) pair in any given population appears only once in that population. This constraint is satisfied in the sample fact table shown.

Figure 2(b) depicts the same situation in a UML class diagram. The {P} notation for preferred identifier and the sample fact table are nonstandard extensions to UML. The "*" multiplicity means "0 or more", so an employee may visit zero or more countries and a country may be visited by zero or more employees. By default, UML assumes that association ends are assigned the {unique} property but not the {ordered} property, ensuring that collection instances are sets (no repetition is allowed, and order is irrelevant). Also by default, role names are lower case versions of the relevant class name. So Employee.country returns a set of countries, and Country.employee returns a set of employees.

Figure 2(c) depicts the same situation in the Barker ER notation. Since this notation does not support objectified associations, or allow associations to have attributes, Visit is modeled as an entity type, and no attempt is made to populate this with visit facts.

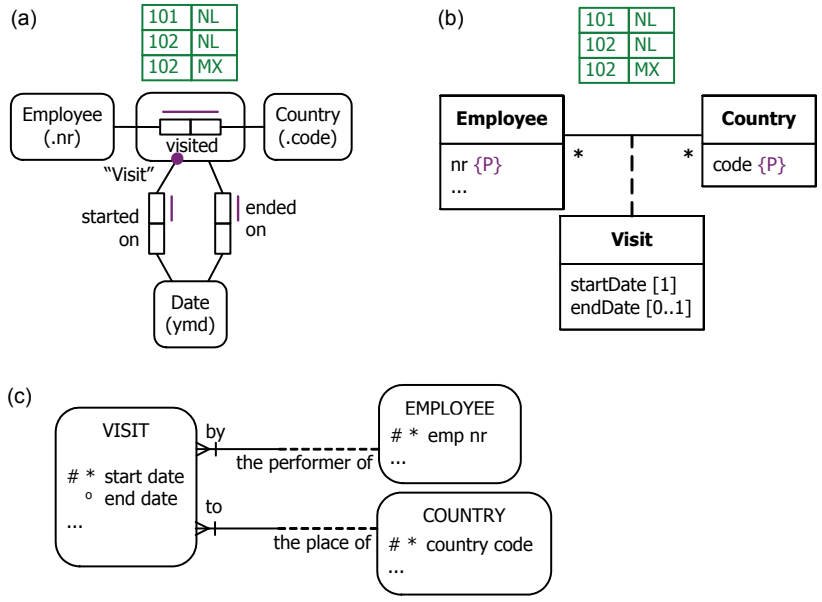

Figure 1 The Visit association populated by a set of facts in (a) ORM and (b) UML.

UML 2 allows association ends to be assigned the {nonunique} property. In this case, if the {ordered} property is not assigned, role instances now return a bag rather than a set. If {ordered} were also assigned, the collection would instead be a sequence (an ordered bag). For example, in Figure 2 both ends of the visit association are assigned the [nonunique} property, so an employee may be associated with a bag of countries, and a country may be associated with a bag of employees. The sample population is expanded with an extra (102, NL) row, to indicate that employee 102 visited The Netherlands twice. The population of the visit association is now a bag of (employee, country) pairs, not a set of (employee, country) pairs.

In this situation, UML is able to distinguish between multiple visits of the same employee to the same country because it identifies visits by internal, object identifiers. While this works, I personally don't favor this approach because the visit identifiers are not part of human communication. Rather than hiding identifiers, I recommend addressing the problem of repeatable facts that give rise to bag situations by remodeling in terms of sets, using human-oriented visible identifiers, as discussed in the previous article. But I can appreciate that not everyone agrees with this recommendation.

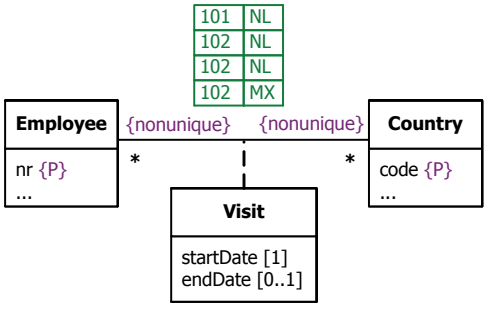

Figure 2 The {nonunique} property may be used in UML 2 to supports bag associations.

Rigid Subtypes vs Role Subtypes

All popular data modeling approaches provide at least basic support for *subtyping*, in which instances of one type (e.g. MalePerson) are necessarily instances of another more comprehensive type (e.g. Person). In such cases, the specialized type is said to be a subtype of the more general type. For a basic discussion of subtyping, see [10].

Recent proposals from the ontology engineering community employ type metaproperties to ensure that subtyping schemes are well formed from an ontological perspective. Guarino and Welty [5] argue that every property in an ontology should be labeled as rigid, nonrigid, or antirigid. *Rigid* properties (e.g. being a person) necessarily apply to all their instances for their entire existence. *Nonrigid* properties (e.g. being hard) necessarily apply to some but not all their instances. *Antirigid* properties (e.g. being an employee) apply contingently to all their instances. One may then apply a metaconstraint (e.g. antirigid properties cannot subsume rigid properties) to impose restrictions on subtyping (e.g. Employee cannot be a supertype of Person).

Later Guizzardi, Wagner, Guarino, and van Sinderen [6] proposed a UML profile that stereotyped classes into kinds, subkinds, phases, roles, categories, roleMixins, and mixins, together with a set of meta-constraints, to help ensure that UML class models are ontologically well-formed. Guizzardi used this modeling profile in his doctoral thesis on ontological foundations for conceptual information models [7].

While the above research provides valuable contributions to ontology engineering, I have some reservations about its use in industrial information systems modeling, because I believe that the 7-stereotype scheme may be overly burdensome to the majority of industrial data modelers. To be fair, I've also had pushback on the expressive detail of ORM, to which I've replied "Well, the world you are modeling is that complex—do you want to get it right or not?" Perhaps the same response could be made in defense of the 7-stereotypes.

At any rate, a simpler alternative currently being considered for ORM 2 *classifies each subtype as either a rigid subtype or role subtype*. A *type is rigid* if and only if each instance of that type must belong to that type for its whole lifetime (in the business domain being modeled). Examples include Person, Cat, Animal, and Book. In contrast, any object that may at one time be an instance of a *role type* might not be an instance of that type at another time during its lifetime (in the business domain). Here I use "role" liberally to include a role played by an object (e.g. Manager, Student, Employee—assuming these are changeable in the business domain) as well as a phase or state of the object (e.g. Child, Adult, FaultyProduct—assuming changeability).

Although this rigid/role classification scheme applies to any type, to reduce the classification burden for modelers this distinction is typically needed only for subtypes (in order to control subtype migration). As a simple example, Figure 3 shows how Dog and Cat might be depicted as rigid subtypes in ORM and UML. The circled "X" and "{disjoint}" constraints indicate that the subtypes are mutually exclusive (i.e. in each state of the business domain, no object instance is recorded as being simultaneously both a dog and a cat). The rigidity notation tentatively being considered for ORM is square bracketing of the subtype name (violet for alethic; blue with "○" for deontic, e.g. changing from male to female might be possible but forbidden). For UML I've chosen a rigid stereotype. Our next example identifies a case where the rigidity of a root type (here Animal) should also be declared.

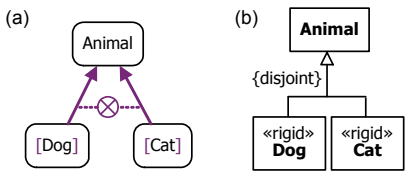

Figure 3. Rigid subtypes depicted in (a) ORM and (b) UML

Rigidity is a *dynamic constraint* rather than a static constraint, since it restricts state changes (e.g. no dog may change into a cat, and no cat may change into a dog). Currently, ORM is being extended to cater for a variety of dynamic constraints using a formal textual language to supplement the ORM graphical language [2], and it is possible that rigidity might end up being captured textually in ORM rather than graphically as shown here.

In the above example, the subtypes are asserted. If instead they are derived, the relevant fact type/attribute used in their definition may be constrained by an appropriate changeability setting with impact on subtype rigidity. In Figure 4(a) the fact type Animal is of AnimalKind is made *unchangeable* (an animal can't change its kind), as indicated by the square brackets (this notation is tentative). In Figure 4(b) the defining animal kind attribute is constrained to be readOnly (prior to UML 2, this was called "frozen").

In either case, the unchangeability of animal kind combined with the rigidity of Animal implies that the subtypes are rigid. If we were instead to assert the subtypes (rather than deriving them from a subtype definition) and derive animal kind from subtype membership, the changeability/rigidity settings would still need to be kept in sync.

UML 2 [16] recognizes four changeability settings: unrestricted, readOnly, addOnly, and removeOnly. ORM 2 is currently being extended to enable declaration of fact type changeability (updateability and deleteability). Barker ER uses a diamond to indicate nontransferable relationships, but this may not be used for attributes.

To avoid explicitly declaring role subtypes as such in ORM 2, subtypes may be assumed to be role subtypes by default (since rigidity is a constraint, this default is consistent with the ORM practice of not assuming a constraint unless it is explicitly declared). Unlike rigid subtypes, *migration between role subtypes is often permitted*. As a simple example, a person may play the role of child, teenager, and adult at different times in his/her lifetime (see Figure 5). The next article in this series discusses various ways to maintain history of objects as they migrate between role subtypes.

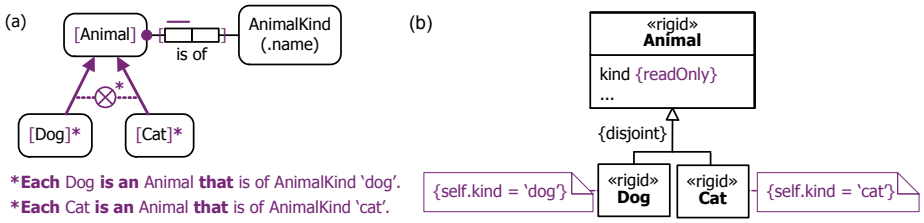

Figure 4. Rigidity of subtypes is now derived (given that Animal is rigid)

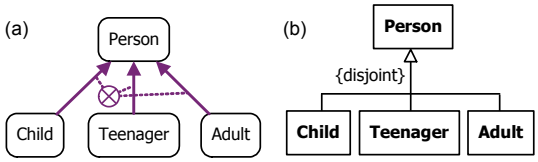

Figure 5. Migration between role subtypes is allowed

Now consider Figure 6, which shows the well known data model pattern for party in ORM, UML, and Barker ER notations. Here Party is partitioned into Person and Organization. Ontologically, Person and Organization are substance sortals (they carry their own natural, intrinsic principle of identity). If "Party" simply means "Person or Organization" (a disjunction of sortals), then Party is a mixin type and there is no problem with this portrayal.

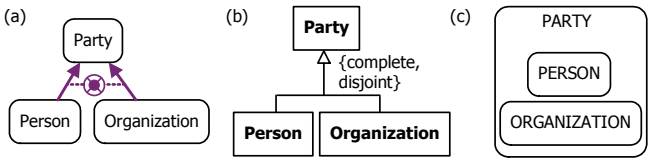

Figure 6. The Party pattern

But what if "Party" has the sense of a role type (e.g. Customer)? If we replace "Party" by "Customer" in Figure 6, then Guizzardi [7, p. 281] would claim that the schema is not well-formed because a rigid universal (e.g. Person) cannot be a subtype of an antirigid one (e.g. Customer). For information modeling purposes however, if each person or organization in the business domain must be a customer, then it's acceptable to specialize Customer into Person and Organization, even though ontologically this is incorrect (in the real world of which the business domain is just a part, not all persons are customers).

The definition of rigid type given earlier is relative to the business domain. In the case just described, Customer is a rigid type in this sense, even though it is not rigid in the ontological sense. *Information models of business domains can be well formed even though they are not proper ontologies*.

If however our business domain includes (now or possibly later) some people or organizations that are not customers, then we do need to remodel, since Customer is no longer rigid even in the sense used here. One of many possible solutions using Party as a mixin type is shown in Figure 7. This solution differs from that of Guizzardi [7, p. 282], where Person and Customer have no common supertype. My original formalization of ORM, which made top level entity types mutually exclusive by default, requires the introduction of a supertype such as Party. This can be pragmatically useful (e.g. by allowing a simple global identification scheme for all parties).

However to avoid unnatural introduction of supertypes, Erik Proper and I long ago allowed the mutual exclusion assumption to be overridden by explicitly declaring an overlap possibility (depicted by overlapping "O"s) between top level types [14]. To reduce notational clutter, and as a relaxation for ORM 2, I now allow overlap possibility between top level types to be implicitly deduced from the presence of a common subtype. With this understanding, the Party supertype could be removed from Figure 7.

Such a relaxation, however, should be used with care. For example, a UML class diagram produced recently by a UML expert depicted the classes Cashier and Customer. When I asked the expert whether it was possible for a customer to be a cashier, he said "Maybe". However nothing on the class diagram indicated this possibility, just as it did not reveal whether nonsense such as a customer being a cashier transaction (transactions were also modeled on the diagram) was possible. The class diagram was little more than a cartoon with informal semantics.

It is sometimes useful in the modeling process to delay decisions about whether some service will be performed in an automated, semiautomated, or manual manner. For example, we can decide later whether cashiers will be ATMs and/or humans. Until we make that decision however, it is safer to allow for all possibilities (e.g. by explicitly declaring an overlap possibility between Cashier and Customer). Otherwise, we should explicitly indicate if our current model is to be interpreted informally.

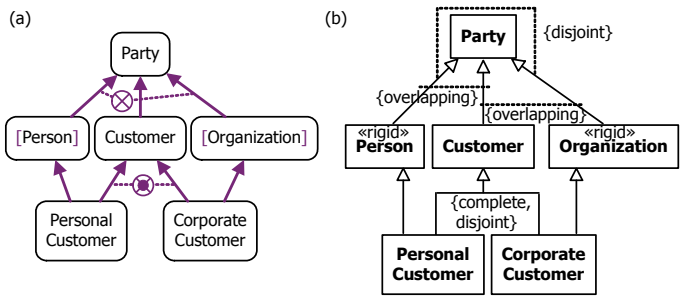

Figure 7. Remodeling is needed when Customer is a role type

Conclusion

This article began by discussing a fourth way to maintain history of nonfunctional, changeable fact types in UML using the {nonunique} property to declare bag associations. Various issues relating to the difference between rigid subtypes and role subtypes were then examined, mainly to lay the groundwork for the next article in this series. Unlike a rigid subtype, a role subtype allows objects over time to join or leave the subtype, thus allowing subtype migration. In many business domains, we need to retain historical details about objects as they move from one subtype to another. The next article considers a variety of data model patterns to allow such histories to be recorded.

References

1. Barker, R. 1990, *CASE*Method: Tasks and Deliverables*, Addison-Wesley, Wokingham, England.
2. Balsters, H., Carver, A., Halpin, T. & Morgan, T. 2006, 'Modeling Dynamic Rules in ORM', *On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, eds. R. Meersman, Z. Tari, P. Herrero et al., Montpellier. Springer LNCS 4278, pp. 1201-10.
3. Chen, P. P. 1976, 'The entity-relationship model—towards a unified view of data'. *ACM Transactions on Database Systems*, 1(1), pp. 9–36.
4. Curland, M. & Halpin, T. 2007, 'Model Driven Development with NORMA', *Proc. 40th Int. Conf. on System Sciences (HICSS-40)*, 10 pages, CD-ROM, IEEE Computer Society, January 2007.
5. Guarino, N. & Welty, C. 2002, 'Evaluating Ontological Decisions with OntoClean', *Communications of the ACM*, vol. 45, no. 2, pp. 61-65.
6. Guizzardi, G., Wagner, G., Guarino, N. & van Sinderen, N. 2004, 'An Ontologically Well-Founded Profile for UML Conceptual Models, *Proc. 16th Int. Conf. on Advanced Information Systems Engineering, CAiSE2004*, eds. A, Persson & J. Stirna. Springer LNCS 3084, pp. 112-126.
7. Guizzardi, G. 2005, Ontological Foundations for Structural Conceptual Models, CTIT PhD Thesis Series, No. 05-74, Enschede, The Netherlands.
8. Halpin, T. & Morgan, T. 2008, *Information Modeling and Relational Databases*, *2nd edition*, Morgan Kaufmann, San Francisco.
9. Halpin, T. 2005, 'ORM 2', *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, eds. R. Meersman, Z. Tari, P. Herrero et al., Cyprus. Springer LNCS 3762, pp 676-87.
10. Halpin, T. 2006, 'Verbalizing Business Rules: Part 14', *Business Rules Journal*, Vol. 7, No. 4 (April 2006), URL: http://www.BRCommunity.com/a2006/b283.html.
11. Halpin T. 2007, 'Temporal Modeling (Part 1)', *Business Rules Journal*, Vol. 8, No. 2 (Feb. 2007), URL: http://www.BRCommunity.com/a2007/b332.html.
12. Halpin, T. 2007, 'Temporal Modeling (Part 2)', *Business Rules Journal*, Vol. 8, No. 6 (June 2007), URL: http://www.BRCommunity.com/a2007/b351.html.
13. Halpin, T. 2007, 'Temporal Modeling (Part 3)', *Business Rules Journal*, Vol. 8, No. 11 (Nov. 2007), URL: http://www.BRCommunity.com/a2007/b374.html.
14. Halpin, T. & Proper, H. 1995, 'Subtyping and polymorphism in object-role modelling', *Data & Knowledge Engineering*, vol. 15, no. 3, pp. 251–281.
15. NORMA website: http://www.ormfoundation.org and http://sourceforge.net/projects/orm.
16. Object Management Group 2003, *UML 2.0 Infrastructure*, URL: http://www.omg.org/uml.