Subtyping Revisited

Terry Halpin

Neumont University
South Jordan, Utah. USA
terry@neumont.edu

Abstract: In information systems modeling, the business domain being modeled often exhibits subtyping aspects that can prove challenging to implement in either relational databases or object-oriented code. In practice, some of these aspects are often handled incorrectly. This paper examines a number of subtyping issues that require special attention (e.g. derivation options, subtype rigidity, subtype migration), and discusses how to model them conceptually. Because of its richer semantics, the main graphic notation used is that of Object-Role Modeling (ORM). However, the main ideas could be adapted for UML and ER, so these are also included in the discussion. A basic implementation of the proposed approach has been prototyped in an open-source ORM tool.

1 Introduction

An information system in the wider sense corresponds to a business domain or universe of discourse rather than an automated system. As the name suggests, the universe of discourse is the world, or context of interest, about which we wish to discourse or talk. Most business domains involve some *subtyping*, where all instances of one type (e.g. Manager) are also instances of a more encompassing type (e.g. Employee). In this example, Manager is said to be a subtype of Employee (a supertype).

Various information modeling approaches exist for modeling business domains at a high level, for example Entity-Relationship Modeling (ER) [1], the Unified Modeling Language (UML) [16, 17, 20], and Object-Role Modeling (ORM) [13]. These modeling approaches provide at least basic subtyping support. In industrial practice however, certain aspects of subtyping are often modeled or implemented incorrectly. This is sometimes due to a lack of appropriate modeling constructs (e.g. derivations to/from subtypes, subtype rigidity declarations), or to a lack of an obvious way to implement a subtyping pattern (e.g. historical subtype migration). This paper proposes solutions to some of these issues. Because of its richer semantics, the main graphic notation used is that of ORM 2 (second generation ORM), as implemented in NORMA, an open source ORM 2 tool. However, the main ideas could be adapted for UML and ER.

Section 2 overviews basic subtyping and its graphical depiction in ORM, UML, and ER, and identifies the condition under which formal derivation rules are required. Section 3 proposes three varieties of subtyping (asserted, derived, and semi-derived). Section 4 distinguishes rigid and role subtypes, relates them to changeability settings on fact type roles, and discusses related subtyping patterns. Section 5 notes implementation issues, summarizes the main results, and suggests future research topics.

2 Basic Subtyping and the Need for Derivation Rules

Fig. 1(a) shows a simple case of subtyping in ORM 2 notation. Patients are identified by their patient numbers and have their gender recorded. Patient is specialized into MalePatient and FemalePatient. Pregnancy counts are recorded for, and only for, female patients. Prostate status is recorded only for male patients. In ORM 2, object types (e.g. Patient) are depicted as named, soft rectangles. A logical predicate is depicted as a named sequence of role boxes, each connected by a line to the object type whose instances may play that role. The combination of a predicate and its object types is a fact type—the only data structure in ORM (relationships are used instead of attributes). If an object type is identified by a simple fact type (e.g. Gender has GenderCode) this may be abbreviated by placing the reference mode in parentheses.

A bar spanning one or more roles depicts a uniqueness constraint over those roles (e.g. **Each** Patient has **at most one** Gender). A large dot depicts a mandatory constraint (e.g. **Each** Patient has **some** Gender). The circled dot with a cross through it depicts an exclusive-or constraint (**Each** Patient **is a** MalePatient **or is a** FemalePatient **but not both**). Overviews of ORM may be found in [13, 12], a detailed treatment in [9], and a metamodel comparison between ORM, ER, and UML in [10]. Various dialects of ORM exist, for example NIAM [23] and PSM [14].

Fig. 1(b) shows the same subtyping arrangement in UML. In UML, the terms "class" and "subclass" are used instead of "object type" and "subtype"". The "{P}" is a non-standard addition to UML to indicate that an attribute is (at least part of) the preferred identifier for instances of the class. ORM and UML show subtypes outside their supertype(s), and depict the "is-a" relationship from subtype to supertype by an arrow. The Barker ER notation [1], arguably the best industrial ER notation, uses an Euler diagram, placing the subtype shapes within the supertype shape, as shown in Fig. 1(c). In spite of its intuitive appeal, the Barker ER subtyping notation is less expressive than that of ORM or UML (e.g. it cannot depict multiple inheritance).

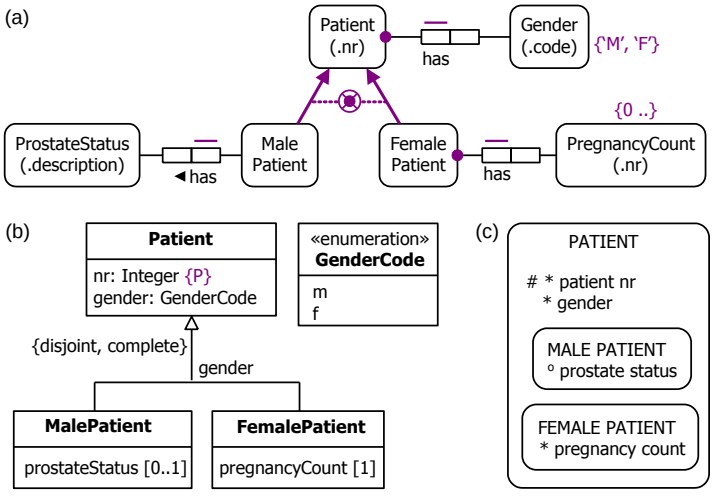

Fig. 1. Partitioning Patient into subtypes in (a) ORM, (b) UML, and (c) Barker ER

The patient example illustrates the three main purposes of subtyping: (1) to indicate that some properties are specific to a given subtype (e.g. prostate status is recorded only for male patients); (2) to permit reuse of supertype properties (e.g. gender is specified once only (on Patient) but is inherited by each of its subtypes); (3) to display taxonomy (e.g. patients are classified into male and female patients).

In this example, the taxonomy is captured in two ways: (1) the subtyping; (2) the gender fact type or attribute (this may be needed anyway to record the gender of male patients with no prostate data). Both ORM and UML allow the possible values for gender (via gender code) to be declared. All of the diagrams in Fig. 1 are conceptually incomplete, since they provide no formal connection between the two ways of displaying the classification scheme for patient. For example, there is nothing to stop us from assigning the gender code 'F" to patient 101 and then assigning the prostate status 'OK' for that patient. Even including "gender" as a discriminator to the subtyping, as allowed in UML (see Fig. 1(b)) and some other versions of ER, will not suffice because there is still no formal connection between gender codes and the subtypes.

ORM traditionally solved this problem by requiring every subtype to be defined formally in terms of role paths connected to its supertype(s). For example, the ORM schema in Fig. 2(a) adds the subtype definitions: **Each** MalePatient **is a** Patient **who** has Gender 'M'; **Each** FemalePatient **is a** Patient **who** has Gender 'F'. In ORM, an asterisk indicates "derived". In this example, the subtype definitions are derivation rules for deriving the subtypes. In previous versions of ORM, all subtypes had to be derived. ORM 2 removes this restriction, so an asterisk is added to indicate the subtype is derived.

The subtypes and fact types in ORM schema in Fig. 2(a) are populated with sample data. The population of the subtypes is derivable from the subtype definitions. The exclusive-or constraint is also derivable (as indicated by the asterisk) from these definitions given the mandatory and uniqueness constraints on the gender fact type.

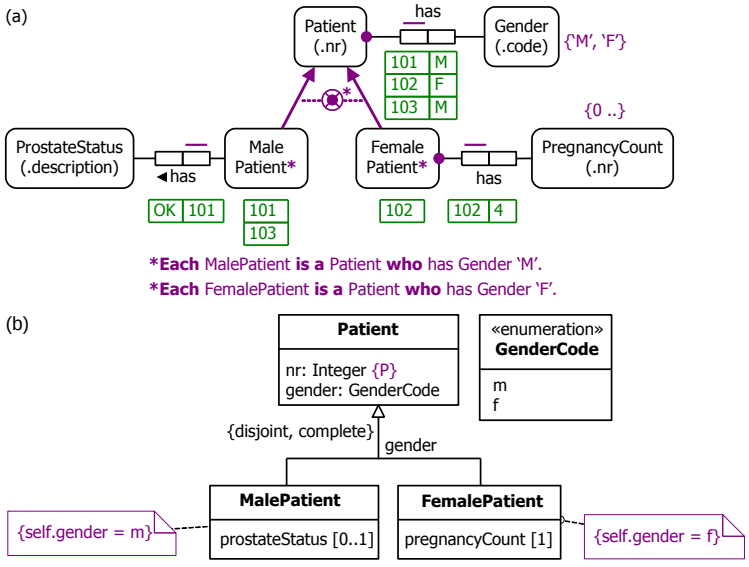

Fig. 2. Adding definitions for derived subtypes in (a) ORM and (b) UML

While UML does not require subtype definitions, one could add them as notes in a language like the Object Constraint Language (OCL) [21], as shown in Fig. 2(b). Adding the constraints shown effectively redefines gender for the specific subclasses. A similar refinement technique is used in the MADS (Modeling of Application Data with Spatio-temporal features) approach [18, p. 47]. Industrial versions of ER typically have no facility for defining subtypes, but could be extended to support this.

As discussed in the next section, subtype definitions/restrictions are not the only way to align multiple ways of depicting classification schemes. The main point at this stage is that *if a taxonomy is specified in two ways (via both subtypes and fact types/attributes) then derivation rules or constraints must be provided to formally align these two mechanisms*.

3 Asserted, Derived, and Semi-derived Subtypes

In previous versions of ORM, all subtypes had to be derived. We recently relaxed this restriction to permit three kinds of subtype: asserted, derived, and semi-derived. An *asserted subtype* (or declared subtype) is simply declared without a definition. Asserted subtypes have always been permitted in UML and ER.

For example, if a gender fact type or attribute is excluded, then the patient subtypes may be simply asserted as shown in Fig. 3. In this case, the exclusive-or constraint indicating that Patient is partitioned into these two subtypes must be explicitly declared, since it is not derivable. In ORM 2, this is shown by the lack of an asterisk beside the exclusive-or constraint. In this case, the classification scheme is depicted in only one way (via subtyping), so there is no need to provide any derivation rules.

Suppose however, that we still wish to query the system to determine the gender of patients. In this case, we may derive the gender from subtype membership. In Fig. 4(a) the ORM fact type Patient is of Gender is derived (as noted by the asterisk) by means of the derivation rule shown. In Fig. 4(b) the UML gender attribute is derived (as indicated by the slash) by means of the derivation rule shown (here using C#). The UML derivation rule shows just one way to derive gender (e.g. we could instead provide overriding gender functions on the subclasses).

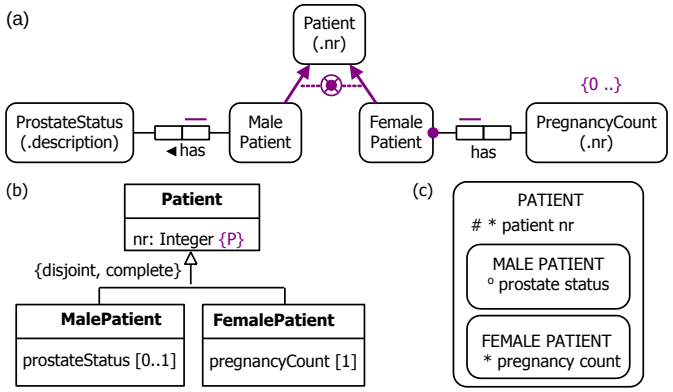

Fig. 3. The subtypes are simply asserted rather than being derived

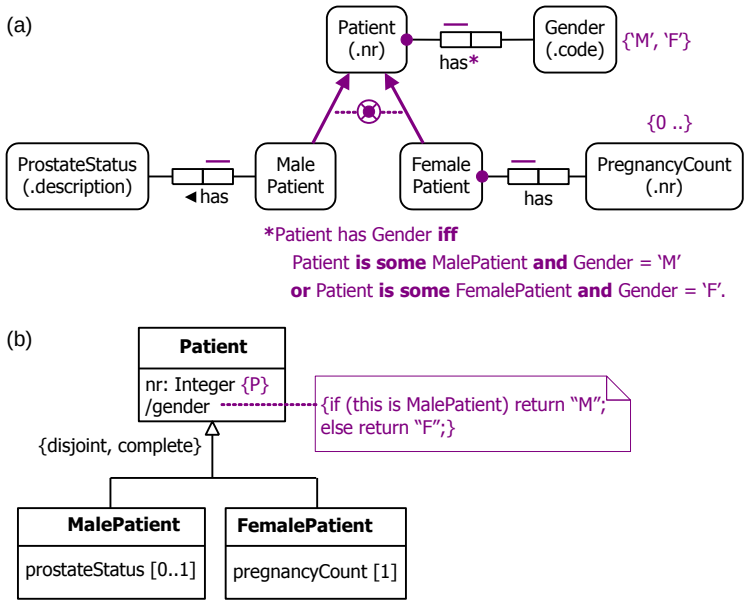

Fig. 4. The subtypes are asserted, and gender is derived

A *derived subtype* is fully determined by the derivation rule that defines it. For example, the subtypes in Fig. 2 are derived (from gender), not asserted. Notice that Fig. 4 is the reverse of the situation in Fig. 2. Conceptually, a constraint applies between gender and the subtypes, and different modeling choices are available to satisfy this constraint (e.g. derive the subtypes from gender, or derive gender from the subtypes). Industrial ER typically does not support derivation rules in either direction.

Recently we introduced *semi-derived subtypes* to ORM 2 to cater for rare cases such as that shown in Fig. 5(a). Here we have incomplete knowledge of parenthood. If we know that person *A* is a parent of person *B* who is a parent of person *C*, then we may derive that *A* is a grandparent. If we know that someone is a grandparent without knowing the children or grandchildren, we can simply assert that he/she is a grandparent. The population of the subtype may now be partly derived and partly asserted. In ORM 2, the semi-derived nature is depicted by a "+" (intuitively, half an asterisk, so half-derived). We use the same notation for fact types, which may also be classified as asserted, derived, or semi-derived. A semi-derived status is much more common for fact types than for subtypes. We note in passing that the parenthood fact type has a spanning uniqueness constraint (hence is many:many), an alethic acyclic constraint, and a deontic intransitive constraint.

Currently UML has no notation for semi-derived (e.g. see Fig. 5(b)). The situation could be handled in UML by introducing an association or attribute for asserted grandparentood, adding a partial derivation rule for derived grandparenthood, and adding a full derivation rule to union the two (cf. use of extensional and intensional predicates in Prolog). This is how we formerly handled such cases in ORM [9].

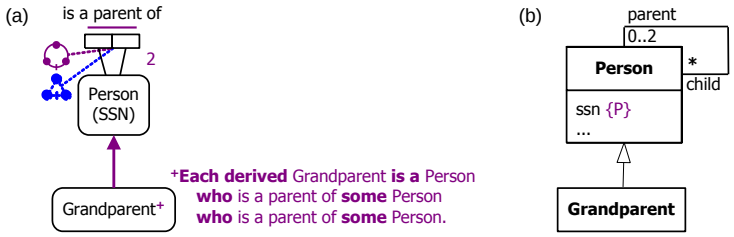

Fig. 5. In the ORM schema (a) the subtype Grandparent is semi-derived

4 Rigid Subtypes and Role Subtypes

Recent proposals from the ontology engineering community have employed 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, non-rigid, or anti-rigid. *Rigid* properties (e.g. being a person) necessarily apply to all their instances for their entire existence. *Non-rigid* properties (e.g. being hard) necessarily apply to some but not all their instances. *Anti-rigid* properties (e.g. being a patient) apply contingently to all their instances. One may then apply a meta-constraint (e.g. anti-rigid properties cannot subsume rigid properties) to impose restrictions on subtyping (e.g. Patient 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. This modeling profile is used by Guizzardi in his doctoral thesis [7] on ontological foundations for conceptual information models.

While we believe that the above research provides valuable contributions to ontology engineering, we have some reservations about its use in industrial information systems modeling. Our experience with industrial data modelers suggests that the 7-stereotype scheme would seem overly burdensome to the majority of them. To be fair, we've also had pushback on the expressive detail of ORM, to which we'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 that we are currently considering 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, 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 we use "role" liberally to include a role played by an object (e.g. Manager, Student, Patient—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).

Though this rigid/role classification scheme applies to any type, we typically require this distinction to be made only for subtypes (our main purpose is to control subtype migration, as discussed shortly; also we wish to reduce the classification bur-

den for modelers). As a simple example, Fig. 6 shows how Dog and Cat might be depicted as rigid subtypes in ORM and UML. The rigidity notation tentatively being considered for ORM is square bracketing of the subtype name (violet for alethic as here; blue with "○" for deontic, e.g. changing from male to female might be possible but forbidden). For UML we have chosen a rigid stereotype. Our next example identifies a case where the rigidity of a root type (here Animal) should also be declared.

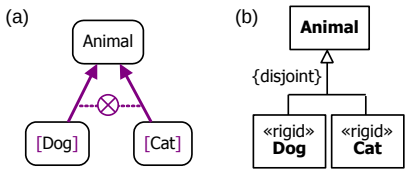

Fig. 6. Rigid subtypes depicted in (a) ORM and (b) UML

Notice that rigidity is a *dynamic constraint* rather than a static constraint since it restricts state changes (e.g. no dog may change into a cat). 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 Fig. 7(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 Fig. 7(b) the defining animal kind attribute is constrained to be read-only (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 and derive animal kind from subtype membership, the changeability/rigidity settings would still need to be kept in sync. Notice that even if we declare gender to be unchangeable in Fig. 4, MalePatient and FemalePatient are not rigid unless Patient is rigid (and that depends on the business domain).

UML 2 [16, 17] 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 non-transferable relationships, but this may not be used for attributes.

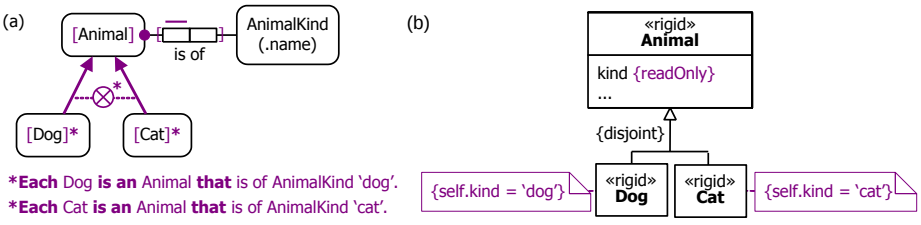

Fig. 7. Rigidity of subtypes is now derived (given that Animal is rigid)

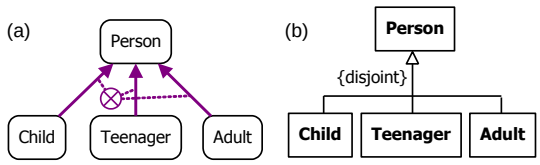

Fig. 8. Migration between role subtypes is allowed

To avoid explicitly declaring role subtypes as such, we propose that subtypes may be assumed to be role subtypes by default. This is similar to the default assumption in MADS that is-a clusters are dynamic [18, p. 44], but it is unclear whether MADS provides any graphical way to override the default. 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 Fig. 8).

An extension to ER to distinguish between "static subtypes", "dynamic subtypes" and "roles" has been proposed by Wieringa [22, pp. 96–99], but this proposal is problematic, as it conflates roles with role occurrences, and reified roles (e.g. employee) with states of affairs (e.g. employment).

Some decades ago, we met our first application where we had to retain history of objects as they passed through various roles (e.g. applicant, employee, past-employee etc.). Although such cases often arise in industry, we are unaware of their discussion in published papers on conceptual modeling. We have space here to discuss just one of the patterns we developed for dealing with such *historical subtype migration*.

The simplest pattern deals with linear state transitions. For example, in Fig. 9(a) each role has specific details, and we wish to maintain these details of a person (e.g. favorite toy, favorite pop group) as he/she passes from one role to another.

An appropriate pattern for this case is to start with a supertype that disjoins all the roles, then successively subtype to smaller disjunctions, as shown in Fig. 9(c). We call this the *decreasing disjunctions pattern*. Depending on the business domain, a simple name may be available for the top supertype (e.g. Person).

As a related issue, consider the well known Party pattern shown in Fig. 10. 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.

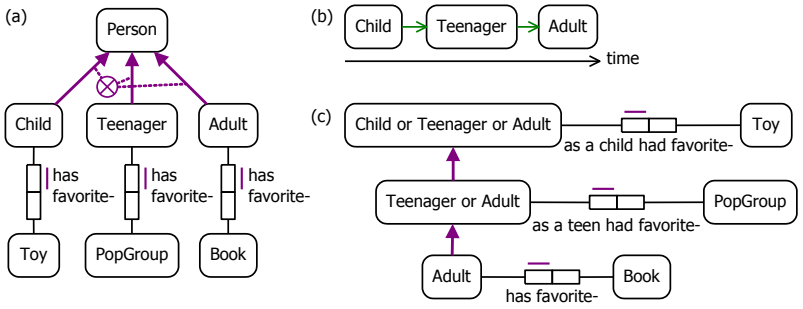

Fig. 9. Retaining history of subtype-specific details as a person changes roles

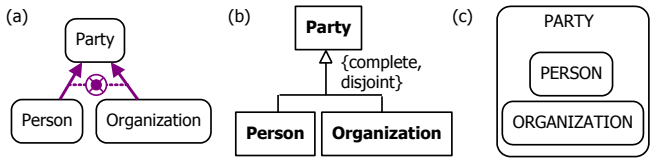

Fig. 10. The Party pattern

But what if "Party" has the sense of a role type (e.g. Customer)? If we replace "Party" by "Customer" in Fig. 10, then Guizzardi [7, p. 281] claims the schema is not well-formed because a rigid universal (e.g. Person) cannot be a subtype of an anti-rigid 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). Our definition of rigid type 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 our sense. One of many possible solutions using Party as a mixin type is shown in Fig. 11. This solution differs from that of Guizzardi [7, p. 282], where Person and Customer have no common supertype. Our 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).

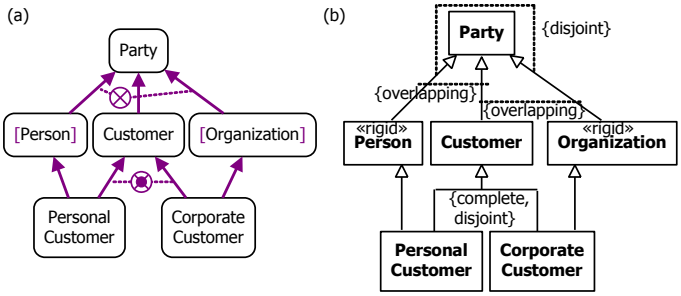

Fig. 11. Remodeling is needed when Customer is a role type

However to avoid unnatural introduction of supertypes, we and a colleague 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 [8]. The same symbol is now used for this purpose in the MADS approach. To reduce notational clutter, and as a relaxation for ORM 2, we 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 Fig. 11.

Such a relaxation however should be used with care. For example, a UML class diagram produced by a UML expert depicted the classes Cashier and Customer. When

we 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 a customer could be a cashier transaction (another class on the diagram). 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, semi-automated, 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, one should explicitly indicate if one's current model is to be interpreted informally.

A radically different "two-layered approach" by Parsons and Wand [19] allows instances to be asserted without requiring them to belong to a type. While interesting, this approach seems unattractive on both conceptual grounds (e.g. even with surrogate identifiers, instances must also be assigned natural identifiers, which are typically definite descriptions that invoke types) and implementation grounds.

5 Conclusion

This paper discussed a number of ways to enrich the modeling of subtyping, especially within ORM 2. With options for asserted, derived, and semi-derived subtypes, classification schemes may be specified in two ways, and derivation rules/constraints are then needed to keep these consistent, regardless of the direction of derivation. A lean ontological extension was proposed based on rigid and role subtypes, mainly to control subtype migration, with appropriate mechanisms for synchronizing subtype rigidity with fact type/attribute changeability. The decreasing disjunctions data model pattern was provided to deal with a common case of historical subtype migration, the party pattern was analyzed to highlight a fundamental difference between information models and ontologies, and a refinement was suggested for determining whether top level types are mutually exclusive.

The proposals for ORM 2 discussed in this paper are being implemented in NORMA [4], an open-source plug-in to Visual Studio that can transform ORM models into relational, object, and XML structures. Relational mapping of subtypes has been largely discussed elsewhere (e.g. [9]), and the dynamic aspects (e.g. non-updatability for rigid properties) can be handled by appropriate triggers. Object models treat all classes as rigid. To cater for migration between role subtypes as well as multiple inheritance when mapping to single-inheritance object structures (e.g. C#), is-a relationships are transformed into injective associations, an approach that bears some similarity to the twin pattern for implementing multiple inheritance [15].

Further research is needed to refine both the graphical and textual languages of ORM 2 for advanced subtyping aspects, and to improve the code generation capabilities of NORMA to ensure an optimal implementation of these features.

References

1. Barker, R. 1990, *CASE*Method: Tasks and Deliverables*, Addison-Wesley, Wokingham.
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.
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 Inf. Sys. 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. & Proper, H. A. 1995, 'Subtyping and polymorphism in object-role modelling', *Data & Knowledge Engineering*, vol. 15, no. 3, pp. 251–281.
9. Halpin, T. 2001, *Information Modeling and Relational Databases*, Morgan Kaufmann, San Francisco.
10. Halpin, T. 2004, 'Comparing Metamodels for ER, ORM and UML Data Models', *Advanced Topics in Database Research, vol. 3*, ed. K. Siau, Idea Publishing Group, Hershey PA, USA, Ch. II (pp. 23-44).
11. Halpin, T. 2005, 'Higher-Order Types and Information Modeling', *Advanced Topics in Database Research, vol. 4*, ed. K. Siau, Idea Publishing Group, Hershey, pp. 218-237.
12. Halpin, T. 2005, 'ORM 2', *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, eds. R. Meersman, et al., Cyprus. Springer LNCS 3762, pp 676-87.
13. Halpin, T. 2006, 'Object-Role Modeling (ORM/NIAM)', *Handbook on Architectures of Information Systems, 2nd edition*, Springer, Heidelberg, pp. 81-103.
14. ter Hofstede, A. H. M., Proper, H. A. & Weide, th. P. van der 1993, 'Formal definition of a conceptual language for the description and manipulation of information models', *Information Systems*, vol. 18, no. 7, pp. 489-523.
15. Mössenböck, H., 'Twin—A Design Pattern for Modeling Multiple Inheritance', Online: http://www.ssw.uni-linz.ac.at/Research/Papers/Moe99/Paper.pdf.
16. Object Management Group 2003, *UML 2.0 Infrastructure Specification*. Online: www.omg.org/uml.
17. Object Management Group 2003, *UML 2.0 Superstructure Specification*. Online: www.omg.org/uml.
18. Parent, C., Spaccapietra, S. & Zimányi, E. 2006, *Conceptual Modeling for Traditional and Spatio-Temporal Applications*, Springer-Verlag, Berlin.
19. Parsons, J. & Wand, Y. 2000, 'Emancipating Instances from the Tyranny of Classes in Information Modeling', *ACM Transactions on Database Systems*, vol. 5, no. 2, pp. 228-268.
20. Rumbaugh, J., Jacobson, I. & Booch, G. 1999, *The Unified Language Reference Manual*, Addison-Wesley, Reading, MA.
21. Warmer, J. & Kleppe, A. 2003, *The Object Constraint Language, 2nd Edition*, Addison-Wesley.
22. Wieringa, R. J. 2003, *Design Methods for Reactive Systems*, Morgan Kaufmann, San Francisco.
23. Wintraecken J. 1990, *The NIAM Information Analysis Method: Theory and Practice*, Kluwer, Deventer, The Netherlands.