Predicate Reference and Navigation in ORM

Terry Halpin

LogicBlox, Australia and INTI Education Group, Malaysia
terry.halpin@logicblox.com

Abstract. A conceptual schema of an information system specifies the fact structures of interest as well as related business rules that are either constraints or derivation rules. The sole data structure used in fact-oriented modeling approaches is the fact type, which may be understood as a set of typed predicates. In spite of the central role played by predicates in fact-orientation, several issues need to be resolved before their full potential can be fully realized. This paper identifies a number of these issues relating to predicate reference and navigation, and proposes some solutions. Specific issues addressed include predicate disambiguation and formalization, role navigation, and automated verbalization of predicate paths. While the discussion focuses largely on Object-Role Modeling (ORM), many of the issues discussed are also relevant to other fact-oriented approaches, such as Cognition enhanced Natural Information Analysis Method (CogNIAM) and the Semantics of Business Vocabulary and Business Rules approach (SBVR), as well as attribute-based approaches like Entity Relationship modeling and the Unified Modeling Language.

1 Introduction

With the rise of model-driven engineering, information systems are increasingly based on high level information models that can be more easily validated with business users. Such an information model includes a conceptual schema as well as a population (set of instances). A conceptual schema specifies the fact structures of interest as well as applicable business rules. Business rules are constraints or derivation rules that apply to the relevant business domain. Constraints restrict the possible or permitted states or state transitions of fact populations. Derivation rules enable some facts to be derived from others.

In fact-oriented approaches, all facts are treated as instances of *fact types*, which may be existential (e.g. Patient exists) or elementary (e.g. Patient smokes, Patient is allergic to Drug). From a logical perspective, a fact type may be treated as a set of typed *predicates*. For example, the readings "Person is employed by Company" and "Company employs Person" denote the same fact type, but involve two different predicates that are inverses of one another. In attribute-based approaches such as Entity Relationship modeling (ER) [4] and the class diagramming technique within the Unified Modeling Language (UML) [19], facts may be instances of attributes (e.g. Patient.isSmoker) or relationship types (e.g. Patient is allergic to Drug). Both of these structures may be formalized in terms of logical predicates.

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 723–734, 2009.

In spite of the central role played by predicates in information modeling, several issues need to be resolved. This paper examines a number of these issues relating to predicate reference and navigation within Object-Role Modeling (ORM), a prime exemplar of the fact-oriented approach based on an extended version of Natural Information Analysis method (NIAM) [24]. An introduction to ORM may be found in [10], a thorough treatment in [15], and a comparison with UML in [13].

Fact-oriented modeling includes other closely related approaches, such as Cognition-enhanced NIAM (CogNIAM) [19], Predicator Set Model (PSM) [16, 17], and Fully-Communication Oriented Information Modeling (FCO-IM) [1]. The Semantics of Business Vocabulary and Business Rules (SBVR) initiative is also fact-based in its use of attribute-free constructs [4, 22]. Recently ORM was extended to second generation ORM (ORM 2) [9], the version used in this paper. An overview of fact-oriented modeling approaches, including history and research directions, may be found in [12]. Although the following discussion focuses largely on ORM, as supported by the Natural ORM Architect (NORMA) tool [6], many of the issues are relevant to other fact-oriented approaches as well as attribute-based approaches like ER and UML.

The rest of this paper is structured as follows. Section 2 discusses predicate disambiguation, including formalization options. Section 3 addresses role navigation, with special reference to *n*-ary relationships. Section 4 provides an algorithm for automated verbalization of predicate paths, and applies this to join constraint verbalization. Section 5 summarizes the main results and outlines future research directions.

2 Predicate Disambiguation

Figure 1(a) depicts three fact types in ORM notation: Person runs Barbershop; Person runs Race; Horse runs Race. Reference schemes for the entity types are omitted for simplicity. Figure 1(b) shows an equivalent UML class diagram.

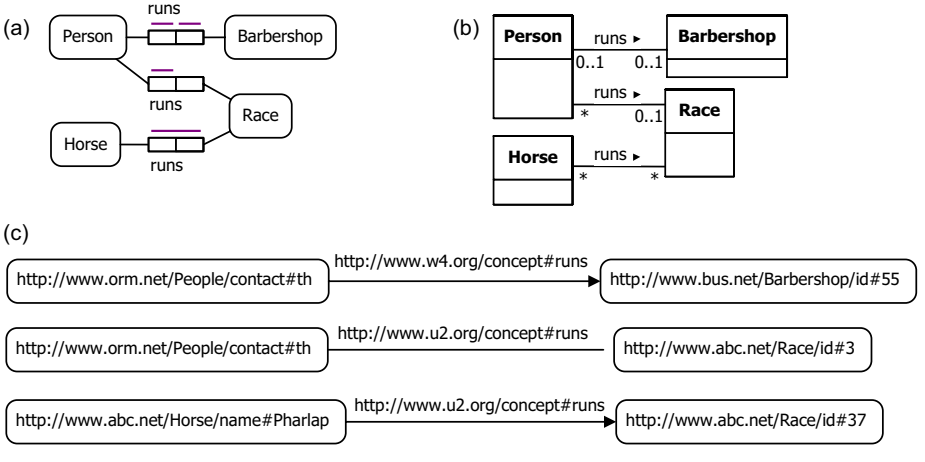

Fig. 1. Multiple "runs" predicates in (a) ORM, (b) UML, and (c) RDF

Each of the three fact types involves a binary predicate displayed with the same verb phrase "runs". If we use the term "predicate" for the intended semantics of a verb phrase occurrence, then the "runs" predicate in the fact type Person runs Race may be considered the same predicate as the "runs" predicate in the fact type Horse runs Race while differing from the "runs" predicate in the fact type Person runs Barbershop. However, the correspondence between two of these "runs" occurrences is lost in both ORM and UML, since neither provides a practical mechanism for indicating such a correspondence.

Although supertyping could achieve this purpose, this would not always be sensible. For example, we could introduce Animal as a supertype of Person and Horse, and replace the two race fact types with the single fact type Animal runs Race. In practice however, the identification schemes for Person and Horse might differ, and be based on different data types (e.g. numeric or string), leading to awkward solutions such as adding artificial identifiers. Moreover, the constraints on the predicate occurrences might differ (as in this example), thus adding to the complexity of unifying the types. For another example of this problem based on ownership, see [15, pp. 262-263].

The Resource Description Framework (RDF) does provide a practical mechanism for predicate identification. In line with the Semantic Web proposal [2], documents may be enriched with global identifiers and structure, using Uniform Resource Identifier references (URIrefs) and embedded tags, to reveal the semantics of what is to be shared in a way that is accessible to automated agents. Ontology languages such as the Web Ontology Language (OWL) are built on top of RDF [25]. Figure 1(c) shows three RDF statements that instantiate the fact types in the ORM and UML schemas. The predicates depicted as labeled arrows in the bottom two statements are seen to be identical in semantics because they have the same URIref.

This raises the following issue. Should modeling approaches such as ORM, ER, and UML be extended to enable predicate occurrences to be "equated" (in the sense of having the same meaning), and if so, how? An answer to this question may well have bearing on procedures for mapping between these approaches and OWL.

When I formalized ORM many years ago [7], I used unsorted predicate calculus, and treated predicates in different fact types as distinct, even if they had the same short "display name" (e.g. "runs", "has"). Display predicate names that appeared in multiple fact types were internally expanded to distinct, full predicate names. This expansion may be done by including relevant object type names or by subscripting. For example, the predicate names in Figure 1 may be unabbreviated to "personRunsBarbershop", "personRunsRace", and "horseRunsRace", or to "$runs_1$", "$runs_2$" and "$runs_3$". For this example, the first predicate could have been expanded simply to "runsBarbershop", but if we later added another fact type such as Company runs Barbershop, at least one of these predicates would need a different full name.

This convention allows fact types to be formalized via simple typing constraints. For example, the three fact types in our example may be declared thus:

$\forall xy(x \text{ personRunsBarbershop } y \rightarrow \text{Person } x \,\&\, \text{Barbershop } y)$

$\forall xy(x \text{ personRunsRace } y \rightarrow \text{Person } x \,\&\, \text{Race } y)$

$\forall xy(x \text{ horseRunsRace } y \rightarrow \text{Horse } x \,\&\, \text{Race } y)$

Strictly, a fact type corresponds to a set of one or more typed predicates, so alternate readings (e.g. "Person operates Barbershop", "Barbershop is run by Person") may be declared for the same fact type, using equivalence operations to establish the correspondence.

While use of predicate subscripts leads to shorter formulae, this has the disadvantage for humans of needing to remember which meaning is intended by the subscript. For example, consider the uniqueness constraint that each person runs at most one barbershop. If we use "$runs_1$" for the predicate, the constraint is formalized by the first formula below. Without remembering which of the "runs" predicates is captured by "$runs_1$", this is less informative than the longwinded second formula.

$$\forall x\, \exists^{0..1} y(x\ runs_1\ y)$$
$$\forall x\, \exists^{0..1} y(x\ personRunsBarbershop\ y)$$

More recently, I've often used sorted logic to formalize ORM. This leads to more elegant formalizations of constraints, so long as we treat all predicates displayed with the same short name as being the same predicate. For example, the uniqueness constraint now formalizes nicely as:

$$\forall x{:}\mathrm{Person}\ \exists^{0..1} y{:}\mathrm{Barbershop}(x\ runs\ y)$$

Unfortunately, this complicates our predicate declaration via typing constraints, since this now leads to disjunctive typing. For example, the three predicate declarations given earlier are replaced by:

$$\forall xy(x\ runs\ y \rightarrow [(\mathrm{Person}\ x\ \&\ \mathrm{Barbershop}\ y) \vee (\mathrm{Person}\ x\ \&\ \mathrm{Race}\ y) \vee (\mathrm{Horse}\ x\ \&\ \mathrm{Race}\ y)])$$

This has some advantages, because we can now easily formulate queries such as "What runs What?" irrespective of the types involved. This is less modular than the previous approach, because every time a runs predicate is added or deleted the disjunction in the type declaration has to be modified accordingly. Nevertheless, the constraint declarations are stable because the types are always specified.

Neither of these two approaches reveals which "runs" occurrences mean the same. If we want to capture this correspondence visually, here is one possible solution. By default, treat predicates with the same "short name" as distinct (they internally expand to full predicate names including the object type names) unless they are explicitly equated by appending the same subscript, or are related by a specialization relationship (see [15, p. 387] for a discussion of association redefinition in UML and ORM). For our example, this leads to the ORM schema of Figure 3. Because such cases are rare in modeling, this seems the least intrusive diagrammatic solution. A URIref could also be assigned (e.g. on a property grid) for any predicate for which this is desired.

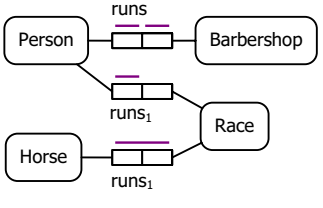

Fig. 2. Equating predicate semantics

The predicate declarations may now be formalized as shown. If a short predicate name occurs in only one fact type, you could instead simply use that (e.g. use "runs" instead of "personRunsBarbershop" in the first formula below) so long as you maintain distinct predicate names if this situation changes.

$$\forall xy(x \text{ personRunsBarbershop } y \rightarrow \text{Person } x \,\&\, \text{Barbershop } y)$$

$$\forall xy(x \text{ runs}_1 \; y \rightarrow [(\text{Person } x \,\&\, \text{Race } y) \vee (\text{Horse } x \,\&\, \text{Race } y)])$$

Care is required when referencing predicates within rules. Consider Figure 3(a), for example. Here the "is licensed" predicate for Doctor means "is licensed to practise medicine", while the "is licensed" predicate for Driver means "is licensed to drive an automobile". A footnoted, deontic rule [11] is specified in FORML for DrivingDoctor. This rule is ambiguous, because there is no formal way to determine which of the "is licensed" predicates is intended.

One way to resolve this ambiguity is to require that in such cases the relevant subtyping relationship(s) must be included in the rule, with an understanding that predicates have minimum scope. Suppose the intent of the rule is that driving doctors ought to be licensed to practice medicine. In this case, the FORML rule can be disambiguated by expansion to "**Each** DrivingDoctor **is a** Doctor **who** is licensed".

A second, and probably preferable, way to resolve the ambiguity is to enforce the following metarule: if object types overlap, and their grammatical predicates have different semantics, then their grammatical predicates must have different readings. Grammatical predicates are basically fact types minus the object type role being predicated. For example, runsRace is Person's grammatical predicate in Person runs Race. For unary fact types only, grammatical predicate names are the short, logical predicate names (e.g. is licensed). For the current example, the schema in Figure 3(a) is then illegal, and the user must rephrase at least one of the "is licensed" predicates to distinguish them. Figure 3(b) renames both predicates, removing the rule ambiguity.

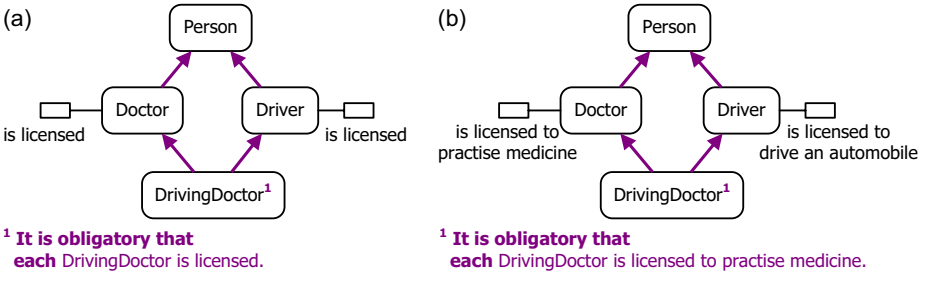

Fig. 3. An ambiguous rule (a) disambiguated (b)

3 Role Navigation

If an object type plays a given role in a predicate, all other roles in that predicate are *far roles* of that object type. Hence, if an object type plays more than one role in a predicate, then all roles in that predicate are far roles for that object type. ORM

requires far role names (if they exist) to be unique for any object type. If an ORM object type *A* has only one of its far roles played by an object type *B*, then by default that far role has *B*'s name as its role name, with its first letter lowercased. These naming rules for roles are similar to those in UML, except UML requires each role to have a name. Unlike ORM, UML does not require predicates to be named, so role names and class names provide the normal way to navigate around a UML class diagram (e.g. using OCL [21, 23]).

ORM allows rules to be formulated in either relational style, using predicate names, or attribute-style, using role names. Navigation via role names is straightforward for binary predicates, since the role name is chosen from the perspective of an instance of an object type for which it is a far role. For example, consider the fact type Company is owned by Company in Figure 4. To find which companies are owned by IBM, we might issue the query in relational style as **List each** Company **that** is owned by Company 'IBM', or in attribute style as Company 'IBM'.subsidiary.

However, role access is less straightforward with *n*-ary fact types ($n > 2$). *In normal practice, roles on n-aries are named with respect to the n-ary relationship itself rather than with respect to one of its object types*. In Figure 4, for example, consider the ternary fact type Company sold Product to Company. Here, the names of the two company roles are "seller" and "buyer", and the product role name is by default "product". These role names are useful for generating column names in a relational table for the sale fact type, but they are not useful for navigating around the schema. For example, an expression such as "company.seller" suggests a company's seller, which is not at all what the expression means.

Suppose that we want to know which products have been sold by IBM. This can be specified in relational style easily enough (e.g. **List each** Product **where** Company 'IBM' sold **that** Product to **some** Company). But this query can't be formulated in attribute style using role names on the ternary. It's no use using "company.product" to navigate from Company to the product role in the ternary, because this doesn't tell us whether a given product was bought by that company or whether it was sold by that company.

To resolve this issue, we introduce derived binary fact types projected from the ternary, and add relevant role names to them, as shown in Figure 4. The query may now be formulated in attribute style thus: Company 'IBM'.productSold.

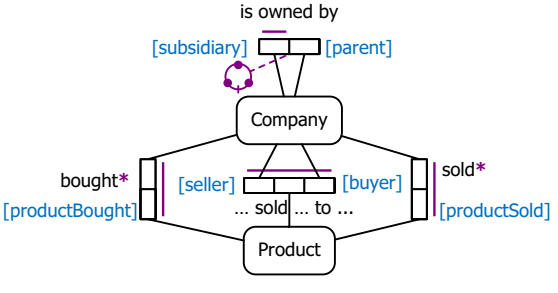

Fig. 4. Navigation using role names

4 Predicate Path Verbalization

Earlier we saw some basic examples of navigating predicate paths in either relational or attribute-style. Navigating from one predicate to another involves a *conceptual join* on the connecting object type(s) [8, 15]. To automate verbalization of constraints involving join paths (e.g. join subset constraints), a mechanism is needed to specify the underlying role paths and the joins involved. This section provides patterns for *verbalizing role paths for the case of inner joins* over roles projected from a role path, and applies the technique to verbalize a join subset constraint.

Figure 5 shows an abstract example, where the external uniqueness constraint is applied to the role pair (r1, r10), which is projected from the *role path* (r1, r2, r3, r4, r9, r10) formed by traversing the predicate sequence *R1*, *R2* and *R5* and performing the inner joins r2 = r3 and r4 = r9. This role path is a *multijoin path* as it involves more than one join. Our role path choice is indicated here by shading the joins, but there are three possible paths between *B* and *C*. Instead of traversing the top predicate *R2*, we could traverse the middle predicate *R3* that implicitly underlies the subtype connection, performing the joins r2 = r5 and r6 = r9. *R3* has the reading "is", and like the other predicates, is an instance-level predicate: it relates instances of *B* to instances of *C*, and is implied by the type-level metapredicate that relates the type *B* to the type *C*. As a third alternative, we could traverse the bottom predicate *R4*, performing the joins r2 = r7 and r8 = r9.

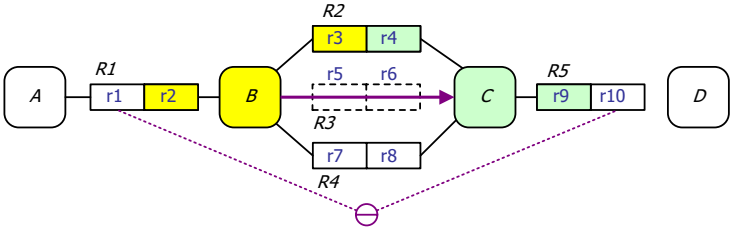

Fig. 5. External Uniqueness Constraint over a multijoin path

Currently NORMA does not graphically highlight the relevant joins (as done here using shading). In general, shading is not adequate since a path may traverse the same predicate more than once. A general graphical solution may be provided by numbering the roles used to perform the joins. An alternative is to use a textual language to formulate constraints for such cases, verbalizing the relevant join paths.

There are two natural ways to verbalize conceptual inner joins in English, one using "**that**" (or "**who**" if the join object type is declared personal) and one using "**and**". The first way may be used if we have a predicate reading that ends with the join's entry role followed by a predicate reading starting at the join's exit role (i.e. one predicate is directed towards, and one predicate reading is directed away from, the join object type). This is illustrated in Figure 6(a), where the join is left-to-right.

The second way may be used if we have a predicate reading that starts at the entry role and another predicate reading that starts at the exit role (i.e. both predicates are directed away from the join object type). This is illustrated in Figure 6(b), where the join is top-to-bottom.

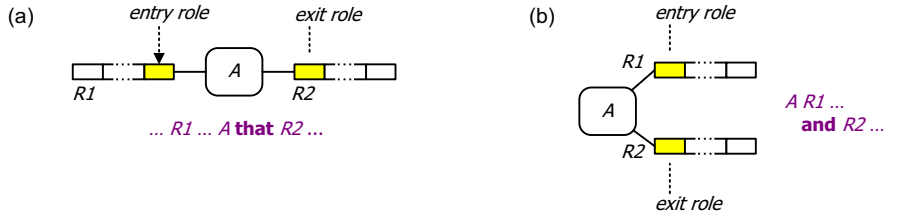

Fig. 6. Two verbalization patterns for inner joins

In the abstract example cited earlier, we have predicate readings *R1*, *R2*, *R3* all in the same direction as the role path. The joins may then be specified using "**that**". The underlying role path which forms the context for the role projection may then be verbalized as: *A R1* **some** *B* **that** *R2* **some** *C* **that** *R5* **some** *B*. The external uniqueness constraint may then be verbalized as shown below. Front text may be accommodated in the usual way, and if an object type plays additional roles on the join path, add subscripts to distinguish its occurrences.

Join Path Context: *A R1* **some** *B* **that** *R2* **some** *C* **that** *R5* **some** *B*.
In this context, **each** *A*, *C* **combination is unique**.

Figure 7 shows a model based on an example from [15] in which the external uniqueness constraint used to provide the preferred identifier for City applies to roles projected from a multijoin path. In this case we do not have a linear sequence of predicate readings for the whole path (we would have if we added the reading Area-Code is of City). The join on City is specified using "**and**" since we have readings from its join roles. The joins on State and Country are specified using "**that**" since we have continuous readings over that path. This uniqueness constraint may be verbalized thus:

Join Path Context: City has AreaCode
and is in State **that** is in Country **that** has RegionCode.
In this context, **each** AreaCode, RegionCode **combination is unique**.

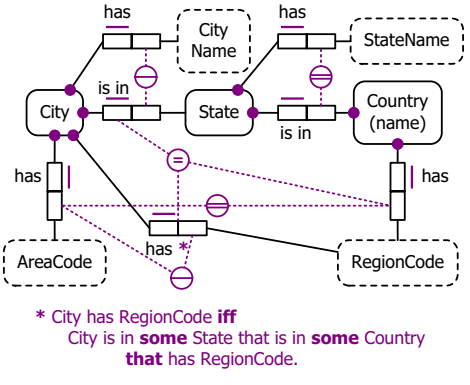

Fig. 7. A complex example of constraints over join paths

The additional information regarding preferred identifier is verbalized as for ordinary cases, as shown below. The three other external uniqueness constraints in the example may be verbalized in the usual way [15].

The unique AreaCode, RegionCode **combination provides the preferred identifier for** City.

If we replace the reading "City is in State" by the inverse reading "State includes City" as in Figure 8(a). The join path may now be declared starting with State, since it begins two continuous paths. Thus:

State includes City **that** has AreaCode
and is in Country **that** has RegionCode.

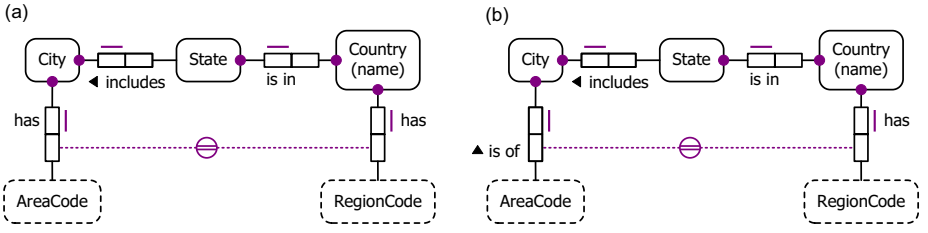

Fig. 8. Join path verbalization depends on predicate direction

But if we also reverse the AreaCode fact type reading, as in Figure 8(b), the City join cannot be verbalized using the two patterns given earlier, since City neither starts nor continues its join predicate readings. To cater for such cases where the join predicate readings end at the entry and exit roles (i.e. both predicates are directed towards the join object type), we introduce the third verbalization pattern shown in Figure 9. If *A* plays additional roles in the path, add subscripts to distinguish its occurrences.

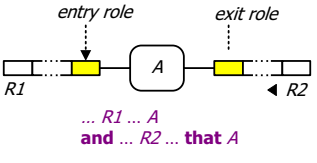

Fig. 9. A third pattern for verbalizing inner joins

Using this third pattern, the join path in Figure 8(b) above may be verbalized as:

State includes City
and is in Country **that** has RegionCode
and AreaCode is of **that** City.

or as: AreaCode includes City
and State includes **that** City
and is in Country **that** has RegionCode.

Heuristics may be added to provide a deterministic choice of the verbalization. Note that the second verbalization above relies on a *minimum backward scope rule for grammatical predicates* to ensure that the subject of "is in Country" is State rather

than AreaCode. If desired, parentheses could be added to explicate the expression for those unfamiliar with this scoping rule, e.g.

AreaCode includes City
and (State includes **that** City
and is in Country **that** has RegionCode).

Note that the three patterns discussed cover all possible cases for an inner join over two roles: (a) one predicate enters the join and one leaves; (b) both predicates start at the join; (c) both predicates end at the join.

We conclude this section by applying the patterns to verbalize a join subset constraint. In general, a subset constraint is directed from one sequence of roles (the source role sequence) to another compatible sequence of roles (the target role sequence). For a join to be involved in the subset constraint, each role sequence must include at least two roles, and at least one of these role sequences must involve at least one join. We consider the common case, where each constrained role sequence is a role pair played by the same object type pair and projected from the ends of a role path of binary predicates with continuous predicate readings in the same direction.

The pattern is shown in Figure 10. The source role pair comprises the roles at the ends of the predicate sequence S_1 .. S_m ($m \geq 1$), and the target role pair comprises the roles at the ends of the predicate sequence R_1 .. R_n ($n \geq 1$). The object types are not necessarily distinct. If A or B are identical to each other or to some of the C or D types, distinguish their occurrences by subscripts. If front text exists, insert it before the relevant "**some**" occurrences and after the relevant "**that**" occurrences (the latter placement is not ideal but will typically suffice; fortunately front text is likely to be rare for this case). If the constraint is deontic, prepend "**It is obligatory that**".

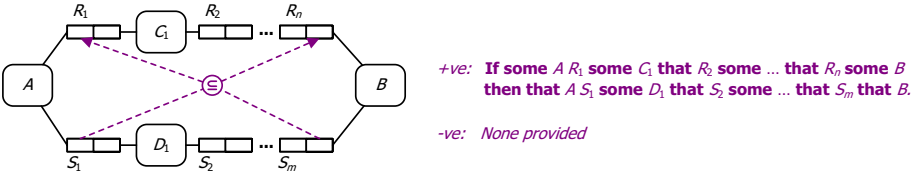

Fig. 10. A pattern for verbalizing a join subset constraint

Applying this pattern to the simple join subset constraint shown in Figure 11 leads to the following verbalization.

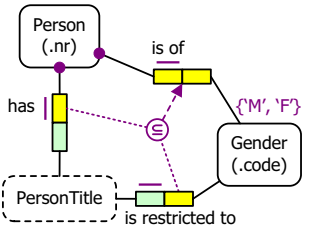

Fig. 11. Example of a join subset constraint

If some Person has **some** PersonTitle **that** is restricted to **some** Gender
then that Person is of **that** Gender.

5 Conclusion

This paper discussed three issues concerning predicate reference and navigation. Although the context for the discussion focused on ORM as supported by NORMA, the issues are relevant to other information modeling approaches. Different options for formalizing and distinguishing predicates were considered. The expanded predicate name option has been implemented as a NORMA extension for generating typed datalog [26]. Empirical research is needed to determine just how useful it would be for modelers to have tool support to indicate when short predicate names are being used with the same underlying semantics.

Two different perspectives for role names were distinguished, depending on fact type arity, and a technique using role names on derived binaries was proposed to enable attribute-style rules to cater for *n*-ary fact types. This technique seems attractive because it does not require any additional ORM constructs to be introduced.

A comprehensive specification was provided for automatically verbalizing role paths involving inner joins, together with an example to illustrate use of the patterns for verbalizing join constraints. This specification is part of a much larger specification that has been completed for automated verbalization of all role paths and join constraints (join subset, join exclusion, join equality, join uniqueness etc.). It is anticipated that this will be implemented in the NORMA tool before the end of 2009.

References

1. Bakema, G., Zwart, J., van der Lek, H.: Fully Communication Oriented Information Modelling. Ten Hagen Stam (2000)
2. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American (May 2001)
3. Bloesch, A., Halpin, T.: Conceptual queries using ConQuer-II. In: Embley, D.W. (ed.) ER 1997. LNCS, vol. 1331, pp. 113–126. Springer, Heidelberg (1997)
4. Bollen, P.: SBVR: A fact-oriented OMG standard. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 718–727. Springer, Heidelberg (2008)
5. Chen, P.P.: The entity-relationship model—towards a unified view of data. ACM Transactions on Database Systems 1(1), 9–36 (1976)
6. Curland, M., Halpin, T.: 'Model Driven Development with NORMA. In: Proc. 40th Int. Conf. on System Sciences (HICSS-40). IEEE Computer Society, Los Alamitos (2007)
7. Halpin, T.: A Logical Analysis of Information Systems: static aspects of the data-oriented perspective, doctoral dissertation, University of Queensland (1989), Online as an 18 MB file at `http://www.orm.net/Halpin_PhDthesis.pdf`
8. Halpin, T.: Constraints on Conceptual Join Paths. In: Krogstie, J., Halpin, T., Siau, K. (eds.) Information Modeling Methods and Methodologies, pp. 258–277. Idea Publishing Group, Hershey (2005)
9. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)

10. Halpin, T.: ORM/NIAM Object-Role Modeling. In: Bernus, P., Mertins, K., Schmidt, G. (eds.) Handbook on Information Systems Architectures, 2nd edn., pp. 81–103. Springer, Heidelberg (2006)
11. Halpin, T.: Modality of Business Rules. In: Siau, K. (ed.) Research Issues in Systems Analysis and Design, Databases and Software Development, pp. 206–226. IGI Publishing, Hershey (2007)
12. Halpin, T.: Fact-Oriented Modeling: Past, Present and Future. In: Krogstie, J., Opdahl, A., Brinkkemper, S. (eds.) Conceptual Modelling in Information Systems Engineering, pp. 19–38. Springer, Berlin (2007)
13. Halpin, T.: A Comparison of Data Modeling in UML and ORM. In: Khosrow-Pour, M. (ed.) Encyclopedia of Information Science and Technology, 2nd edn., Information Science Reference, Hershey PA, US, vol. II, pp. 613–618 (2008)
14. Halpin, T., Curland, M.: Automated Verbalization for ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1181–1190. Springer, Heidelberg (2006)
15. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
16. ter Hofstede, A., Proper, H., van der Weide, T.: Formal definition of a conceptual language for the description and manipulation of information models. Information Systems 18(7), 489–523 (1993)
17. Hoppenbrouwers, S.J.B.A., Proper, H.A(E.), van der Weide, T.P.: Fact Calculus: Using ORM and Lisa-D to Reason about Domains. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 720–729. Springer, Heidelberg (2005)
18. Meersman, R.: The RIDL Conceptual Language, Int. Centre for Information Analysis Services, Control Data Belgium, Brussels (1982)
19. Nijssen, M., Lemmens, I.M.C.: Verbalization for Business rules and Two Flavors of Verbalization for Fact Examples. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 760–769. Springer, Heidelberg (2008)
20. Object Management Group, UML 2.0 Superstructure Specificatio (2003), http://www.omg.org/uml
21. Object Management Group, UML OCL 2.0 Specification (2005), http://www.omg.org/docs/ptc/05-06-06.pdf
22. Object Management Group, Semantics of Business Vocabulary and Business Rules, SBVR (2008), http://www.omg.org/spec/SBVR/1.0/
23. Warmer, J., Kleppe, A.: The Object Constraint Language, 2nd edn. Addison-Wesley, Reading (2003)
24. Wintraecken, J.: The NIAM Information Analysis Method: Theory and Practice. Kluwer, Deventer (1990)
25. World Wide Web Consortium, OWL 2 Web Ontology Language, W3C Working Draft (2009), http://www.w3.org/TR/2009/WD-owl2-overview-20090611/
26. Zook, D., Pasalic, E., Sarna-Starosta, B.: Typed Datalog. In: Gill, A., Swift, T. (eds.) PADL 2009. LNCS, vol. 5418, pp. 168–182. Springer, Heidelberg (2009)