Join Constraints

Terry Halpin

Microsoft Corporation, USA
e-mail: thalpin@attbi.com
website: www.orm.net

Abstract: Many application domains involve constraints that, at a conceptual modeling level, apply to one or more schema paths, each of which involves one or more conceptual joins (where the same conceptual object plays roles in two relationships). Popular information modeling approaches typically provide only weak support for such join constraints. This paper contrasts how join constraints are catered for in Object-Role Modeling (ORM), the Unified Modeling Language (UML), the Object-oriented Systems Model (OSM), and some popular versions of Entity-Relationship modeling (ER). Three main problems for rich support for join constraints are identified: disambiguation of schema paths; disambiguation of join types; and mapping of join constraints. To address these problems, some notational, metamodel, and mapping extensions are proposed.

1 INTRODUCTION

At the analysis phase of information system development, a conceptual schema may be used to describe the structure of the application domain in way that is easily understood and validated by the domain expert. Once validated, the conceptual schema can then be mapped to logical/physical/external schemas using automated and/or manual processes. For industrial database applications, the high level data modeling is typically performed using a version of Entity Relationship (ER) modeling [6], such as Information Engineering (IE) [9], Barker ER [1], or Integration Definition 1 extended (IDEF1X) [18]. Recently, Object-Role Modeling (ORM) [10] and Unified Modeling Language (UML) [19] class diagrams have gained some industrial adoption for information modeling. In addition, modeling techniques from academia, such as Object-oriented Systems Model (OSM) [7, 8], can be used to construct information models.

With increasing competition in the marketplace, and the potential costs of bad data, there has been a growing appreciation of the central role that business rules play in enforcing data integrity. Since such rules need to be validated with domain experts who may have little knowledge of implementation structures, it is best to capture them first in a conceptual schema where they can be readily communicated. In practice, many application domains involve business rules that are essentially constraints over one or more schema paths, each of which involves one or more conceptual joins (where the same conceptual object plays a role in two relationships). Although join constraints often apply to an application domain, they are not always included in the data model for the domain, partly because popular modeling approaches typically provide only weak support for such constraints. This makes it harder for the modeler to detect the rules and decide whether to include them in the application. It also makes it harder for the developer who must now code the rules instead of benefiting from automated code generation from a high-level rule specification.

This paper discusses current support for join constraints in the approaches listed earlier, and suggests ways to improve the support. Three main problems for rich support for join constraints are identified: disambiguation of schema paths; disambiguation of join types; and mapping of join constraints. To address these problems, some notational, metamodel, and mapping extensions are proposed.

Section 2 deals with uniqueness and frequency constraints on a single join path. Section 3 examines set comparison constraints that apply to one or more join paths. Section 4 briefly outlines how join constraints may be catered for by extensions to metamodels and mapping procedures. Section 5 summarizes the main results, and lists references for further reading.

Because of its inherent simplicity and richness, the ORM notation is used for much of the discussion, but the main results have applicability to the other notations. ORM concepts used are briefly defined in this paper as needed. An overview of ORM is given in [10], and a detailed treatment in [12].

2 CONSTRAINTS OVER A SINGLE JOIN PATH

Before discussing constraints on join paths, the notions of internal uniqueness constraint and conceptual join are explained, using a simple example adapted from [12]. Figure 1(a) shows an ORM schema diagram with two base associations (Employee drives Car; Car was imported from Country) and a derived, association (Employee drives Car imported from Country). In ORM, object types are denoted by named ellipses. Simple identification schemes may be abbreviated in parentheses. For example, Employee(empNr) abbreviates the injective (1:1-into) association Employee has EmpNr. Each association is shown as a named sequence of one or more role-boxes. A role in ORM is a part played in an association, where the associations may be unary (one role), binary (two roles), ternary (three roles), or longer. The association is the only data structure in ORM, so associations are always used instead of attributes.

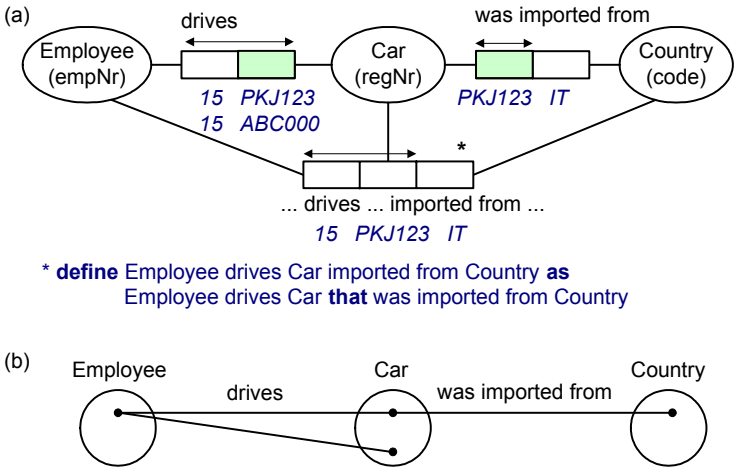

Figure 1 The ternary association is derived from a conceptual inner join of the binary associations

The asterisk on the ternary association indicates that it is derived. The derivation rule indicates how it is derived, and is formally expressed in an ORM conceptual query language known as ConQuer [2]. Here the word "that" is used to perform a conceptual join, indicating that the car driven by the employee must be the same car that is imported from a country. In essence, the derived association is an abbreviation for the schema path that starts at Employee and goes through the drives predicate to Car, and then through the import predicate to Country. As we pass through the Employee object type on this path, we demand that the car playing the role of being driven must be the same car that plays the role of being imported. This entails a conceptual join between the two roles played by Car (shown shaded in the figure). The instance diagram in Figure 1(b) shows another way to picture this. This is sometimes called an object join, since the same object must play both roles.

In ORM, associations not used for primary reference are often called fact types. Each fact type may be populated by a fact table (set of fact instances), whose columns correspond to the roles. Figure 1(a) includes a sample population for the two base fact types, and the resulting population derived for the ternary fact type. Here employee 15 drives two cars, one of which (the car with registration number PKJ123) is imported (from Italy). Since car ABC000 is not imported, it is excluded from the result. If you think of the fact tables as relational tables, then it should be clear than the join here is an inner join.

In contrast, a conceptual left outer join from the drives "table" to the import "table" would include the additional tuple (15, ABC000, ?) in the result, where "?" denotes a null value. In ConQuer, left outer joins may be specified by inserting the modal possibility operator "possibly" immediately after "that" in the path expression.

The arrow tipped-bars spanning one or more roles in the predicates are internal uniqueness constraints. If a role has a uniqueness constraint applying just to it, then entries in its fact column must be unique. For example, the entry PKJ123 may appear only once in the car column of the import fact table. If the uniqueness constraint spans multiple roles, the combination of entries in the associated fact columns must be unique. For example, the combination (15, PKJ123) may appear only once in the drives fact table. The

spanning uniqueness constraint on the drives fact type simply indicates that it is many-to-many, and can be populated only by sets of facts, not bags of facts.

With these concepts understood, we are ready to address external uniqueness constraints and their associated joins. We begin with a simple case where the join path and join type is unambiguous. Figure 2(a) is an ORM diagram with two associations: Room is in Building; Room has RoomNr. A building is identified by its building number. RoomNr identifies itself, so is a value type (shown as a broken ellipse) rather than an entity type. The large black dots on Room are simple mandatory role constraints, to indicate that the roles attached are mandatory for Room (each room must be in a building, and must have a room number). The combination of mandatory and internal uniqueness constraints allows ORM to model cardinality and multiplicity constraints used by other notations, in a way that scales properly to n-ary associations. For a discussion of why other notations such as UML fail to scale properly, see [11, 13].

The circled "u" is an external uniqueness constraint spanning roles played by Building and RoomNr, to indicate that each building, room number combination refers to at most one room. Roles in the same predicate are co-roles of one another. The semantics of an external uniqueness constraint may be determined by finding a schema path between the co-roles of the constrained roles, performing conceptual joins on the intermediate object types, bag-projecting on the constrained roles in the derived association formed from this path, and then asserting that an internal uniqueness constraint applies to this projection. Hence the constraint requires that the bag-projection must yield the same result here as set-projection. In this example, there is exactly one schema path and the conceptual join occurs between the roles played by Room (these roles are shaded in the figure).

Figure 2(b) shows how to model this situation in Barker ER notation. Here the "#" on "room nr" indicates it is part of the primary identifier for Room, and the stroke "|" on the left role of the Room in Building association indicates that the relationship is also part of the primary identifier for Room. Unlike ORM, the Barker ER notation cannot express external uniqueness constraints outside primary identification schemes. For example, if we added a global roomId attribute as the primary identifier of Room, we could no longer assert the external uniqueness constraint in Barker ER.

Figure 2(c) depicts the example in IE notation. There is no graphic notation for external uniqueness in IE, so the constraint needs to be captured informally elsewhere (e.g. in a note). Figure 2(d) models the case using IDEF1x. Unfortunately, the external uniqueness constraint can only be modeled by migrating building number over to the primary key of Room, and declaring it to be a foreign key to Building. This is just a syntactic variant of relational notation, so cannot be regarded as conceptual.

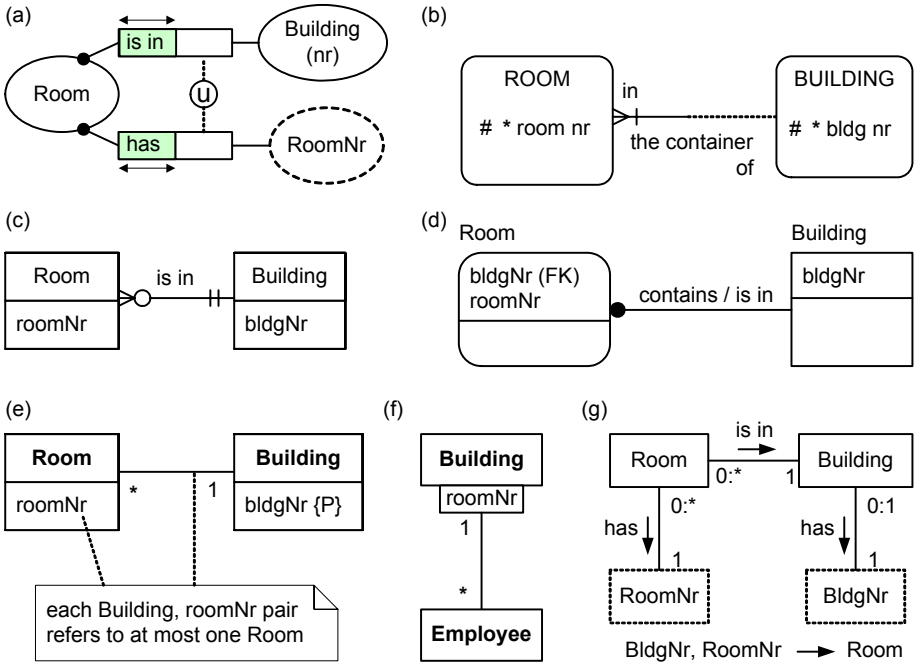

Figure 2 Depicting external uniqueness in various modeling notations

Figure 2(e) shows one way to model this situation in UML. In the current UML 1.4 specification [19], there is no graphic notation for external uniqueness in the general ORM sense, so the constraint has been expressed informally as a note attached to the relevant model elements. UML does not even include a graphic notation for unique attributes, so I have added a non-standard {P} notation to indicate that bldgNr is the primary conceptual means of identifying buildings. UML does provide partial support for external uniqueness by allowing association ends to be qualified, as shown in Figure 2(f). Here an employee may be associated with a room in a building by qualifying Building by roomNr. This effectively partitions building into rooms based on room number. Unfortunately, this technique is of little use if we wish to treat Room as a class itself, as would be normal practice.

Figure 2(g) portrays the example in OSM. Like ORM, OSM adopts an attribute-free approach for data modeling. In ORM, the notation Building(nr) is just an abbreviation for the injective association Building has BuildingNr, so the parallel between OSM and ORM should be obvious. OSM does not have a graphic notation for external uniqueness, but it does support it formally by allowing the diagram to be annotated with a functional dependency expression that captures the constraint, as shown. In OSM this is called a "co-occurrence" constraint.

Like ORM, OSM allows declaration of derived fact types that are formally defined in terms of existing fact types as well as operations such as joins. In OSM these are called "high-level relationship sets" ([7] section 4.1.6; [8] section 4.3). While this feature could be used to emulate a number of ORM's graphic capabilities, it does so at the expense of requiring multiple diagrams (one for defining the abstraction, and another for using it), and may still require additional textual annotations to capture semantics that ORM portrays graphically on a single diagram.

In OSM, the meaning of its arrow symbol "→" is context-dependent. When attached to associations, it shows the direction in which to read the association name (similar to UML's "◗"). When used in co-occurrence constraints, it means "functionally determines". This semantic overload can be confusing for users unfamiliar with the notation, and could be avoided by using different symbols for the different senses.

In the above example, the two roles participating in the conceptual join are both mandatory for the join object type (Room). This ensures that the conceptual join is an inner join. However, if one or more roles involved in the conceptual join are optional, the meaning of the external uniqueness constraint may be ambiguous. As a simple example, consider the two ORM models shown in Figure 3.

In Figure 3(a), each course must have a title, but may or may not be offered by a department (e.g. it could be a department course, a general course, or a course offered by a visiting lecturer). The external uniqueness constraint indicates that if a course is offered by a department, then the combination of the department and course title is unique. In other words, within a given department each course has a different title. But if two courses are not offered by a department, must they have different titles? There are two reasonable answers to this question: Yes, or No. In principle, we could respond Unknown, but in practice this is not an option because we require the semantics to be well defined (in the rare case where there is no domain expert to provide a definite answer, we make a choice ourselves). At any rate, the constraint is ambiguous unless we agree to always answer such a question in the same way.

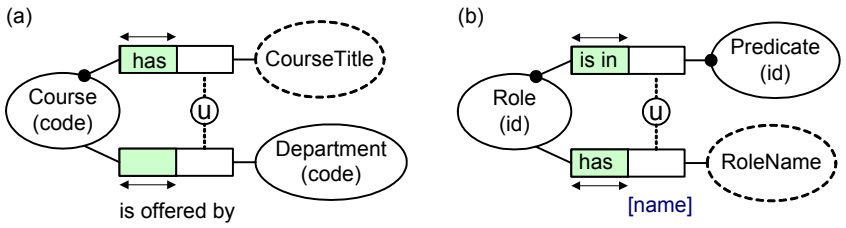

Figure 3 External uniqueness constraints involving optional roles may be ambiguous

If we answer Yes to the question, then the same title may apply to at most one course with no department. The same title may however apply to many courses in different departments. In terms of the underlying join path for the constraint, this interpretation applies conceptual outer joins to optional roles, with the added proviso that null values are treated as ordinary values. The join of the title and department fact types results in a compound ternary fact type equivalent to the relational table scheme in Figure 4.

Course(courseCode, courseTitle, [departmentCode])

C1	Mechanics	PY	
C2	Mechanics	MA	
C3	Mechanics	?	
C4	*Mechanics*	*?*	*-- violates constraint*

Figure 4 Outer join semantics for the external uniqueness constraint in Figure 3(a)

This relational notation underlines uniqueness constraints; if more than one exists, the primary key is doubly-underlined. Optional columns are marked with square brackets. The external uniqueness constraint in Figure 3(a) appears as the internal uniqueness constraint spanning courseTitle and departmentCode.

The sample population shows three courses C1, C2, C3 with the same title, one offered by the physics department, one by the mathematics department, and one offered by no department. The fourth row (C4, Mechanics, ?) is rejected because in the presence of row 3 it violates the constraint that courses with no department must have different titles. For the purpose of comparison, the null values of rows 3 and 4 are treated like ordinary values, and thus are equated.

Many years ago, we introduced this outer join semantics for external uniqueness constraints to model disjunctive reference schemes at the conceptual level, and discussed various ways to implement these constraints in a relational database system [15]. To simplify the discussion above, each course is primarily identified by a course code (e.g. 'C1'), but this is not essential, since the semantics may apply even if the disjunctive reference scheme is the only means of identifying courses. Independently of our work, Thalheim provided an alternative formalization of primary keys with optional attributes [23]. The outer join interpretation is reasonable for the course example, and is absolutely required in many practical situations. For a practical case study of disjunctive reference in botanical naming systems, see [16, 20] for a simplified treatment and [21] for the full treatment.

Now consider the ORM model in Figure 3(b), which is a fragment of an ORM metamodel. Since ORM requires each predicate to have at least one reading, it is optional whether roles are given names. In the figure, the role played by RoleName is given the name "name" (depicted here in square brackets), but the other roles are not named. The external uniqueness constraint indicates that rolenames are unique within a given predicate. However, many roles within the same predicate may have no name. Unlike the course example, this situation requires an inner join interpretation for the join path underlying the external uniqueness constraint. Let us identify the predicate of the association Role is in Predicate as P1, and its left and right roles as r1, r2 respectively. Let us also identify the predicate of the association Role has RoleName as P2, and its left and right roles as r3, r4 respectively.

The join of the two fact types results in a compound ternary fact type equivalent to the relational table scheme in Figure 5. The external uniqueness constraint appears as the internal uniqueness constraint in Figure 5(a). Since we only require roles to have unique names in a predicate if they are named, the conceptual join between the roles of Role in Figure 3(b) is an inner join, resulting in the single row population shown. For comparison, the outer join result is shown in Figure 5(b). If we were to take the outer join interpretation as we did for the course model, the first two rows would violate the compound uniqueness constraint.

(a)

Role(roleId, predicateId, [roleName])

r4	P2	name

(b)

Role(roleId, predicateId, [roleName])

r1	P1	?
r2	P1	?
r3	P2	?
r4	P2	name

Figure 5 Inner join semantics for the external uniqueness constraint in Figure 3(b)

Although this example might seem unusual, we have encountered several real world cases involving this inner join semantics. In practice, uniqueness constraints involving optional roles may have either the inner join or outer join semantics. So we need a way to distinguish them. This can be done graphically, textually, or both. It can also be done on a role basis or a constraint basis.

For ORM, graphical external uniqueness constraints already indicate which roles are involved, so we only need to add a notation for indicating the join type for optional co-roles of the constrained roles. In general, an external uniqueness constraint may span two or more roles, so there could be cases with multiple joins involving an optional co-role. Moreover, as discussed later, even linear join-paths may involve many joins. So a complete solution to the problem theoretically requires that each join on the path be identified in some way as an inner or outer join. While graphical annotations can be invented to do this, we feel that pragmatically there is no need for such a general graphical solution. In the many real world cases we have met, a uniform inner/outer join interpretation has always been intended for optional role joins along the join path, and intuitively this seems sensible. For pathological cases of join paths with mixed join types on optional roles, we recommend that this be specified textually, using a language such as ConQuer, which is easily capable of dealing with such cases.

If we thus assume that all joins on optional roles on the join path underlying an external uniqueness constraint are of the same type (inner or outer), we simply need to indicate which of these types applies to the overall constraint. In ORM we suggest that this is done graphically as follows: use the current notation for the inner join interpretation, and make this the default. For the outer join interpretation, adorn the circled "u" in some way (e.g. use a double line for the circle).

For OSM, the co-occurrence arrow could be flagged as outer by appending a symbol (e.g. "+"). For attribute-association approaches such as ER and UML, an ORM-like notation could be added for linking association roles only, association roles and attributes, as well just attributes. If only attributes within a single entity type or class are involved, a more concise notation could be used by annotating the relevant uniqueness constraint component (e.g. {U1+}) assuming that such constraints are supported at all. Clearly, the attribute-free nature and greater expressive power of ORM and OSM makes it much easier to add support for join type disambiguation.

In the previous examples, the join path underlying the external uniqueness constraint involves only one join. But in practice, the join path may involve many joins. Figure 6(a) shows an ORM schema for an example in [12, p. 647]. The "<<" indicators reverse the normal downwards/rightwards reading direction of predicates. The join path for the external uniqueness constraint involves three conceptual joins, on Country, State and City. Since all the join roles (shaded) are mandatory, the joins are all inner joins. Note that conceptual joins need not give rise to relational joins, since relations may combine multiple fact types. For instance, the ORM schema maps by default to the relational schema shown in Figure 6(b). At this level, the external uniqueness constraint is enforced by naturally joining the tables on country name and then ensuring that the countryCode, areaCode bag-projection is unique (this constraint is easily coded in SQL). The conceptual joins on State and City come for free at the relational level, since the relevant roles map to the same table.

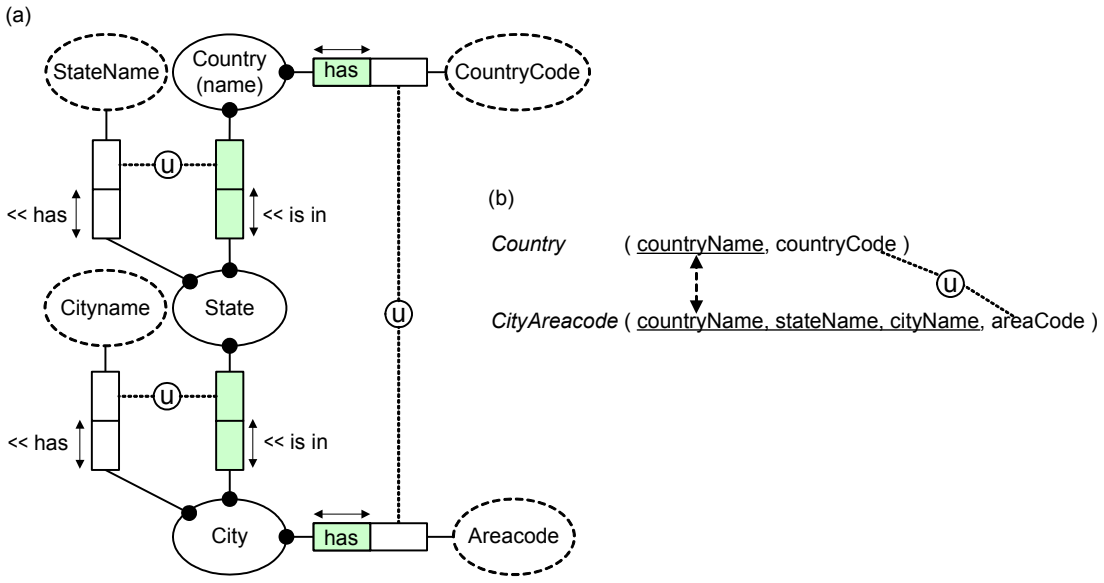

Figure 6 An external uniqueness constraint involving a join path with three joins

The join path in Figure 6(a) is unambiguous, since there is only one direct path between the co-roles of the two roles spanned by the external constraint. However suppose other associations exist between the join object types. For instance, suppose we add the association: City trades with State. There are now two possible join paths, since there are two ways to navigate from State to City.

Figure 7 summarizes the problem of path ambiguity. To find the join path underlying the external uniqueness constraint, we need to find a path connecting roles r2 and r9, moving right from r2 and left from r9. Our choice is shaded, but there are three possible paths we could take in going between B and C. We could traverse the top predicate, performing the joins r2 = r3 and r4 = r9. We could traverse the middle predicate that implicitly underlies the subtype connection, performing the joins r2 = r5 and r6 = r9. Or we could traverse the bottom predicate, performing the joins r2 = r7 and r8 = r9.

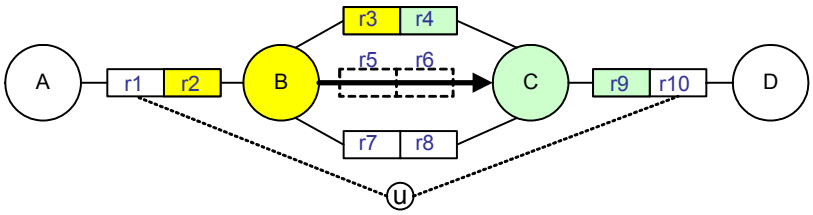

Figure 7 Join paths sometimes require explicit declaration of the join roles

Ways to disambiguate the underlying path for external uniqueness constraints have specified in various procedures, such as the Uniquest algorithm [3]. Currently ORM does not include any graphic notation to highlight the relevant joins (as we have 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, but this can lead to messy diagrams. A tool could prompt the user to select the list of join roles, and then toggle the display of the joins on/off under user control. An alternative is to use a textual language such as ConQuer to formulate all external uniqueness constraints that have multiple candidate join paths.

For non-ORM approaches, first basic support for external uniqueness is required, and then the join paths may be disambiguated graphically as for ORM, or textually if the method includes a sufficiently powerful formal query language. UML could use the Object Constraint Language (OCL) for such specifications [24], although OCL expressions often appear cryptic to non-technical domain experts.

In ORM, a frequency constraint constrains the frequency (number of times) with which an object (or object sequence) that populates a role (or role sequence) may appear there for any given state of the information base. Frequency constraints may include lists and/or ranges of frequencies (e.g. 2, 4..7, >=20). Uniqueness constraints are equivalent to frequency constraints where the frequency equals 1. In conjunction with mandatory role and uniqueness constraints, frequency constraints cater for advanced forms of multiplicity and cardinality constraints, and also scale properly for n-ary associations.

Of the modeling approaches discussed earlier, only ORM and OSM support external frequency constraints (i.e. frequency constraints spanning roles from different predicates). For this, ORM uses a frequency specification (e.g. 3..5) connected by dotted lines to the relevant role sequence, and OSM attaches the frequency specification to the arrow it uses in co-occurrence constraints.

As for external uniqueness, external frequency constraints have an underlying join path. If the joins involve optional roles, then the join type (inner or outer) is potentially ambiguous. However we have never encountered a practical example of this, so recommend that inner joins be assumed, and a formal textual language such as ConQuer be used instead of a graphic notation if ever the outer join interpretation is required.

3 SET-COMPARISON CONSTRAINTS INVOLVING JOINS

Set-comparison constraints restrict the way the population of one role, or role sequence, relates to the population of another. ORM recognizes three kinds of set-comparison constraint, one for each of the subset, equality and exclusion comparators. In the literature, subset and exclusion constraints at the relational level are sometimes called inclusion and exclusion dependencies. The non-ORM modeling

approaches have either no support or only very limited support for these constraints. Detailed discussions of set-comparison constraint support in ER and UML may be found in [12,14]. We confine our attention here to set-comparison constraints between role sequences, at least one of which includes roles from different predicates, and therefore involves an underlying join path.

Figure 8(a) shows an ORM model with a subset constraint from one role-pair to another. The subset constraint is depicted as a circled "⊆" embedded in a dashed arrow running from the source role sequence to the target role sequence. If the roles in the sequence are contiguous, the constraint connects directly to the junction of the roles. Otherwise, the roles in the sequence are indicated by a dotted line connecting the roles. With our example, the subset constraint runs from the role pair comprising the Advisor serves in Country predicate to the role pair comprising the first roles of the Advisor speaks Language and Country uses Language predicates.

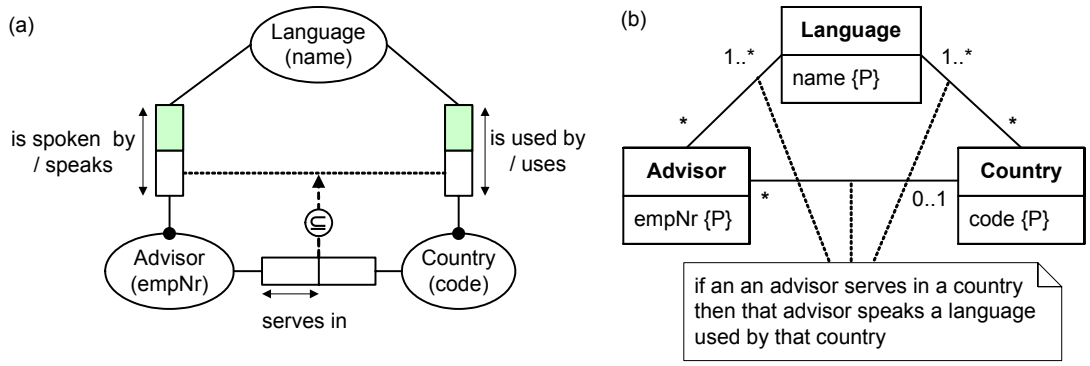

Figure 8 A join-subset constraint expressed in ORM and UML

The constraint means that the set of advisor-country pairs populating the serves-in predicate must be a subset of the set of advisor-country pairs obtained by projecting on the advisor and country roles of the join path Advisor speaks **a** Language **that** is used by Country. This join path involves a conceptual join on Language, equating instances playing its two roles (shaded in the figure). In other words, if an advisor serves in a country, then that advisor must speak at least one language used by that country.

Like the other approaches, UML has no graphic notation for expressing a join-subset constraint, but does allow the constraint to be expressed textually in a note attached to the relevant model elements (in this case the three associations), as shown in Figure 8(b). The other approaches could also be adapted to allow such usage of notes.

In ORM, an equality constraint is equivalent to a subset constraint in both directions, and is denoted using a circled "=". An exclusion constraint indicates that populations of the constrained role sequences must be mutually exclusive, and is denoted using a circled "X". Examples of these may be found in [11, 12]. As for subset constraints, it is possible that one of the role sequences involves a join path.

The same considerations given to join paths underlying external uniqueness constraints may be raised for each join path involved in a set-comparison constraint. Although we have modeled numerous practical application domains involving join set-comparison constraints, the join types have always been inner joins. So rather than complicating the graphic notation for rare cases, we recommend that any set comparison constraints involving outer joins be specified textually using a formal language such as ConQuer. To disambiguate set comparison constraints over role sequences with multiple candidate join paths, exactly the same advice given earlier for external uniqueness constraints in this regard applies.

It is well known that any relational expression may be reformulated as a functional expression and vice versa. Although ORM is primarily disposed towards expressing textual constraints in association form (relational), its allowance of role names permits such constraints to also be expressed in attribute form (functional or multi-valued). Object-oriented languages such as OCL [24] and Object Definition Language (ODL) [5] use attribute form exclusively.

Sometimes association form is preferable, because it verbalizes rules in a more natural way that domain experts can readily comprehend. Sometimes attribute form is preferable because it provides a simple, compact formulation. However, attribute form is inherently less stable than association form, so

there is a trade-off. For example, the join subset constraint in Figure 8, may be verbalized in association form as shown in R1.

(R1) **if an** Advisor serves in **a** Country
then that Advisor speaks **a** Language
that is used by **that** Country

A high-level, attribute-form language would enable the rule to be formulated simply using role path expressions and a built-in subset operator, as in R2.

(R2) Advisor: self.country **is subset of** self.languages.countries

In practice, OCL and ODL require more circuitous formulations, but in principle the R2 syntax could be supported. This formulation assumes the names of roles (association ends) in Figure 8(b) are implicitly provided by names of the classes, pluralizing the names where the maximum multiplicity exceeds one. If the existence of other associations ambiguates this assumption, then role names should be explicitly provided in the model.

Since business changes may cause attributes to be remodeled as associations or classes, rules in attribute form are inherently less stable than rules in association form. Another disadvantage of rolename-based path expressions is that rolename plurality is potentially unstable. For example, suppose the advisory business decides to change its policy of allocating advisors to at most one country, instead allowing the same advisor to serve in more than one country. In the ORM model of Figure 8(a) this causes the uniqueness constraint on Advisor serves in Country to expand to cover both roles, but this has no impact on rule R1, which still applies. In the UML model of Figure 8(b), the "0..1" multiplicity constraint changes to "*", so that the associated role name change should change from "country" to "countries". This means rule R2 also has to be changed since Advisor.country needs to be replaced by Advisor.countries. For manual rule specification, there is no way around this problem other than to ignore singular/plural name distinctions, rendering the rule less understandable. With tool support, this problem could be ameliorated by having the user select the roles from the diagram and letting the tool generate the rule verbalization, automatically updating it as needed when multiplicities change.

4 METAMODELING AND MAPPING

Adding new notations to a modeling language enables the modeler to capture more constraints in the language. For automated support however, the notations need to be catered for in the language's metamodel, and mapping procedures need to be specified so that the high level rule specifications can be automatically translated into implementation code (e.g. relational DDL scripts).

Our comparative review of ER, ORM and UML metamodels for data modeling purposes in [13] ignored any discussion of join type disambiguation and role path disambiguation for join constraints. Recently we extended the ORM metamodel to cater for these aspects. There is no space here for a detailed discussion of these extensions, but the metamodel fragment in Figure 9 provides the basic idea.

A single role constraint (mandatory, unique etc.) spans a single list of roles. This is captured by the ternary SingleRoleListConstraint spans Role in Position. An external frequency constraint or external uniqueness constraint has an underlying join path as discussed earlier. If the join path is ambiguous (as in Figure 7), the user needs to specify the joins used along the path. The metamodel captures this join path by recording the list of entry and exit roles for each object type along the join path (see the other ternary). A similar approach may be adopted to capture role paths involved in set-comparison constraints over role sequences. Finally, if an external uniqueness constraint has an optional role in its join path, an inner join is assumed unless the constraint itself is specified as outer. This outer join setting applies to all optional roles along the constraint's join path.

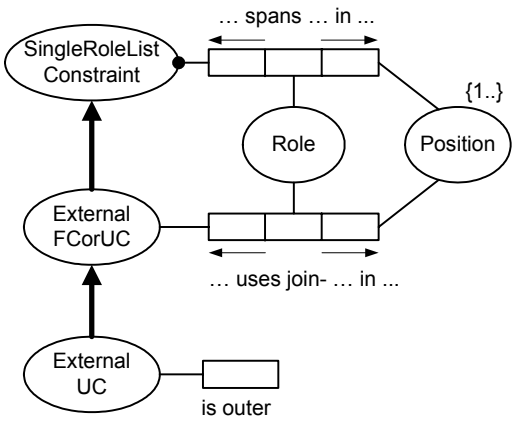

Figure 9 ORM metamodel fragment to capture the join type and join path of some constraints

Algorithms to map ORM schemas to normalized relational schemas have been detailed elsewhere, e.g. [12, 20, 21]. These algorithms have been largely implemented in various tools, including the ORM source model solution in Microsoft Visio for Enterprise Architects. Currently however, the only ORM tool to fully automate the mapping of general ORM join paths with inner or outer joins is ActiveQuery [2], and this is a conceptual query tool rather than a conceptual modeling tool.

Regardless of whether the modeling tool is based on ORM, ER, UML or OSM, mapping of join types and join paths to DDL for implementation in industrial DBMSs needs to cater for the differences in DBMS dialects. Even a basic discussion of the mapping process would need a paper in itself, so we confine our discussion to a simple example. Recall that the ORM schema in Figure 3(b) maps to the relation scheme in Figure 5(a): *Role*(roleId, predicateId, [roleName]). How do we enforce the uniqueness constraint on (predicateId, roleName) in SQL? It is unsafe to simply generate the table check clause **unique**(predicateId, roleName), because this has different semantics for different back ends. Although the SQL standard interprets unique in the inner sense described earlier, SQL Server allows at most one null if a unique constraint or unique index is declared. To implement the inner join semantics required for this case, we generate the following table check clause.

```
check( not exists
        (select * from Role
        where roleName is not null
        group by predicateId, roleName
        having count(*) > 1 ) )
```

To apply outer-join semantics to the uniqueness constraint, simply remove the where-clause. If the model is mapped to a non-relational target, such as C# code, then the constraint is handled very differently. In spite of such mapping complexities, the conceptual model is the right place to specify the rules in the first place, where they can be readily understood and validated by the domain expert. Implementing mapping algorithms to transform the high level representation to code is a lot of work, but once done the pay off is significant.

5 CONCLUSION

This paper examined a set of constraints that are often neglected by modeling approaches. Support for these constraints in various modeling approaches was discussed, and proposals for improved support were suggested. These constraints involve one or more conceptual join paths in the underlying conceptual schema. For external uniqueness constraints, joins on optional roles may be inner or outer, so the relevant choice needs to be clearly made and catered for in mapping algorithms. For join constraints with more than one candidate join path, the relevant path needs to be identified and mapped. Basic extensions to notations, metamodels and mapping procedures were outlined to address these issues

There are different viewpoints on how complete a conceptual modeling language should be for capturing constraints, especially graphically. Practical experience indicates that the kinds of constraint discussed in this paper frequently exist in the application domain being modeled. Including a constraint in the graphical language often makes it easier for the modeler to think about the constraint, but there is a limit to how many concepts may be included in a graphical language before making it too complex for the modeler to work with. We have found it convenient in real world modeling work to incorporate the kinds of constraints discussed in this paper in the graphical modeling language, but only empirical studies can show whether this perception is valid for the wider community. We recommend such a study as a topic for future research.

References

1. Barker, R. 1990, *CASE*Method: Entity Relationship Modelling*, Addison-Wesley, Wokingham, England.
2. Bloesch, A. & Halpin, T. 1997, 'Conceptual queries using ConQuer-II', *Proceedings of the 16th International Conference on Conceptual Modeling ER'97* (Los Angeles), Springer LNCS 1331 (Nov.) 113-126.
3. van Bommel, P., ter Hofstede, A.H.M. & van der Weide, Th.P. 1990, 'Semantics and Verification of Object-Role Models', Tech. Report 90-13, Dept. of Informatics, University of Nijmegen.
4. Booch, G., Rumbaugh, J. & Jacobson, I. 1999, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading MA, USA.
5. Cattell, R.G.G. & Barry, D.K (eds) 2000, *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann Publishers, San Francisco.
6. Chen, P.P. 1976, 'The entity-relationship model—towards a unified view of data'. *ACM Transactions on Database Systems*, 1(1), pp. 9–36.
7. Embley, D.W. 1998, *Object Database Management*, Addison-Wesley, Reading MA, USA.
8. Embley, D.W., Kurtz, B.D. & Woodfield, S.N. 1992, *Object-Oriented Systems Analysis: A Model-Driven Approach*, Yourdon Press, Englewood Cliffs, NJ, USA.
9. Finkelstein, C. 1998, 'Information engineering methodology', *Handbook on Architectures of Information Systems*, eds. P. Bernus, K. Mertins & G. Schmidt, Springer-Verlag, Berlin, Germany, pp. 405-427.
10. Halpin, T.A. 1998, 'ORM/NIAM Object-Role Modeling', *Handbook on Information Systems Architectures*, eds P.Bernus, K. Mertins & G. Schmidt, Springer-Verlag, Berlin, pp. 81-101.
11. Halpin, T.A. 2001, 'Supplementing UML with concepts from ORM', *Unified Modeling Language: Systems Analysis, Design, and Development Issues*, eds K. Siau & T. A. Halpin, Idea Publishing, Hershey PA.
12. Halpin, T.A. 2001, *Information Modeling and Relational Databases*, Morgan Kaufmann, San Francisco.
13. Halpin, T.A. 2002, 'Metaschemas for ER, ORM and UML: A Comparison', *Journal of Database Management*, Idea Group Publishing, Hershey PA, pp. 4-13.
14. Halpin, T.A. & Bloesch, A.C. 1999, 'Data modeling in UML and ORM: a comparison', *Journal of Database Management*, vol. 13, no. 2, Idea Group Publishing, Hershey PA, pp. 20-30.
15. Halpin, T.A. & Ritson, P.R. 1992, 'Fact-oriented modelling and null values', *Research and Practical Issues in Databases*, eds B. Srinivasan & J. Zeleznikov, World Scientific, Singapore.
16. Halpin, T.A. & Ritson, P.R. 1996, 'Entity integrity revisited', *Australian Computer Journal*, vol. 28, no. 3, pp. 73-80.
17. 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.
18. NIST (1993). *Integration definition for information modeling (IDEF1X)*. FIPS Publication 184, National Institute of Standards and Technology. [Online}. Available: http://www.sdct.itl.nist.gov/~ftp/idef1x.trf.
19. OMG UML Revision Task Force, UML 1.4 specification, online at www.omg.org/uml.
20. Ritson, P.R. & Halpin, T.A. 1993, 'Mapping integrity constraints to a relational schema', *Proc. 4th Australian Conf. on Inform. Systems*, Dept. of Commerce, University of Queensland, pp. 381-400.
21. Ritson, P.R. 1994, Use of conceptual schemas for a relational implementation, PhD thesis, University of Queensland.
22. Rumbaugh, J., Jacobson, I. & Booch, G. 1999, *The Unified Modeling Language Reference Manual*, Addison-Wesley, Reading MA, USA.
23. Thalheim, B. 1989, 'On semantic issues connected with keys in relational databases permitting null values', *J. Inf. Process. Cybern.* 1989 1/2, pp. 11-20.
24. Warmer, J. & Kleppe, A. 1999, *The Object Constraint Language: precise modeling with UML*, Addison-Wesley, Reading MA, USA.