A Role Calculus for ORM

Matthew Curland, Terry Halpin, and Kurt Stirewalt

LogicBlox, USA
{matt.curland,terry.halpin,kurt.stirewalt}@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. Constraints restrict the possible or permitted states or state transitions, while derivation rules enable some facts to be derived from others. Graphical languages are commonly used to specify conceptual schemas, but often need to be supplemented by more expressive textual languages to capture additional business rules, as well as conceptual queries that enable conceptual models to be queried directly. This paper describes research to provide a role calculus to underpin textual languages for Object-Role Modeling (ORM), to enable business rules and queries to be formulated in a language intelligible to business users. The role-based nature of this calculus, which exploits the attribute-free nature of ORM, appears to offer significant advantages over other proposed approaches, especially in the area of semantic stability.

1 Introduction

To promote correctness and clarity, an information model should first be specified at the conceptual level, where it can be more easily validated by the business users most qualified to act as domain experts, before the model is forward engineered to an implementation. We use the term "model" to include both the schema (structure) as well as a population (set of instances). A conceptual schema specifies the fact structures of interest as well as related business rules that are either constraints or derivation rules. Constraints restrict the possible or permitted states or state transitions, while derivation rules enable some facts to be derived from others.

Graphical diagrams are convenient for displaying the main features of a conceptual schema, but these often need to be supplemented by textual formulations in order to capture business rules for which no graphical notation exists. Textual languages may also be used to query conceptual models directly.

Although information models are often specified using attribute-based approaches such Entity Relationship modeling (ER) [5] and the class diagramming technique within the Unified Modeling Language (UML) [18], fact-oriented approaches seem to offer some clear advantages for conceptual modeling. For example, the attribute-free nature of fact-orientation facilitates natural verbalization of models, while promoting semantic stability (e.g. the restructuring needed in ER or UML when one later decides to record facts about an attribute is simply avoided). Interest in fact-orientation has recently been sparked by the adoption by the Object Management Group of the Semantics of Business Vocabulary and Business Rules (SBVR) approach [20].

R. Meersman, P. Herrero, and T. Dillon (Eds.): OTM 2009 Workshops, LNCS 5872, pp. 692–703, 2009.

Object-Role Modeling (ORM) is a prime exemplar of the fact oriented approach, based on an extended version of Natural Information Analysis method (NIAM) [23]. An introduction to ORM may be found in [9], a thorough treatment in [14], and a comparison with UML in [12]. An overview of fact-oriented modeling approaches, including history and research directions, may be found in [11].

This paper describes research efforts to provide a *role calculus* to underpin textual languages for ORM, to enable business rules and queries to be formulated in a language that is intelligible to business users. The role-based nature of this calculus, which exploits the attribute-free nature of ORM and related tooling support, appears to offer significant advantages over other proposed approaches, especially in the area of semantic stability.

The rest of this paper is structured as follows. Section 2 briefly overviews related research. Section 3 discusses a metamodel fragment for a role calculus to capture ORM role paths and derivation rules. Section 4 illustrates the use of the role calculus with examples of rules for derived subtypes and derived fact types. Section 5 summarizes the main results, outlines future research directions, and lists references.

2 Related Approaches

Currently the most popular textual rule language for information models is the Object Constraint Language (OCL), which is used to augment UML class models with constraints and derivation rules that cannot be expressed graphically in UML [19, 22]. In spite of its usefulness, OCL has technical drawbacks (e.g. rule contexts are restricted to classes) as well as pragmatic drawbacks (e.g. OCL expressions are often too technical for business users to understand and hence validate). While the intelligibility issue could be addressed by a friendlier surface syntax, this has yet to occur. Some textual languages for ER have been proposed (e.g. see section 16.3 of [14]), but these are limited in scope, and share with OCL the problem of semantic instability caused by an underlying attribute-based model.

Various templates and guidelines exist to assist users to formulate business rules in an intelligible way, with RuleSpeak (http://www.rulespeak.com/en/) being a prominent example. However, while helpful, these initiatives do not currently provide a formal syntax and semantics to support unambiguous, executable rules.

The first textual language for fact-oriented modeling was Reference and IDea Language (RIDL), which has a high level syntax for declaring and querying NIAM models [17]. In the 1980s, the model declaration part was implemented in the RIDL* tool, but relationships were restricted to binary relationships, and the query part was never implemented. Later on, other fact-oriented rule languages were developed. The Language for Information Structure and Access Descriptions (LISA-D), based on the Predicator Set Model (PSM) which extended NIAM with collection types (e.g. power types and sequences types), included support for specifying constraints, updates, and queries. While expressive and formal, LISA-D has not yet been fully implemented.

Early tool support for ORM introduced two textual languages. Formal ORM Language (FORML), used mainly for specifying constraints, was supported as an output verbalization language in the InfoModeler tool and in the ORM solution within Microsoft Visio for Enterprise Architects. Conceptual Query Language (ConQuer)

enabled ORM models to be queried, and was implemented in the InfoAssistant and ActiveQuery tools [3, 4]. The ConQuer language was used only for formulating conceptual queries. Although it could have been extended to capture constraints and derivation rules, this never happened, and tool support for it is no longer available.

Recently, ORM was extended to second generation ORM (ORM 2) [8], with tool support provided by Natural ORM Architect (NORMA) [6], including improved constraint verbalization in FORML [13] as well as further rule options such as semiderived types [14], deontic rules [10], and deep support for conceptual outer joins [7, 14]. The role calculus work described in this paper is currently being implemented in extensions to NORMA. Apart from supporting a richer version of ORM, NORMA provides more flexible support for role-based model changes, resulting in greater semantic stability. For example, roles may be inserted into, repositioned, or deleted from existing predicates without impacting rules based on other roles.

The NORMA screenshot in Figure 1 shows a request to insert a role after a selected role of a ternary fact type. Basing rule structures on roles, which have rigid internal identifiers, also ensures that rule structures are not impacted by changing user-supplied role names, preferred predicate readings, or object type names, or even changing the host language (English, German etc.). NORMA's automatic verbalization facility is designed to generate readings on demand.

Recently, a Fact Calculus with similar objectives to our work was developed based on extensions to the earlier work with LISA-D [16]. Unlike our approach, this is based on PSM rather than ORM 2, and requires forward and reverse role names as well as role-pair connector names to navigate along role paths. While enjoying strong formal foundations, the fact calculus does not appear to have tooling support, it does not cater for conceptual outer joins or deontic rules, and its rule formulations are typically less intelligible than ours. For example, the fact calculus rule "NO Official-paper of A Car being returned BUT NOT being returned" [16] is verbalized in our approach as "**No** Car **that** is returned has **some** OfficialPaper **that** is **not** returned".

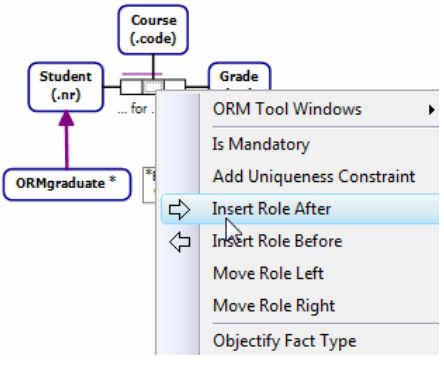

Fig. 1. NORMA screen shot showing some role options. Some context menu options (e.g. Delete Role) are omitted.

3 Role Paths

The fundamental notion of role calculus is that all relationships between instances of one or more object types in an ORM model can be expressed by navigating role paths between those object types. A structured grouping of *Role* elements can therefore be used as the basis of the definition for any relationship. Of the fundamental building blocks of an ORM metamodel (*ObjectType*, *FactType*, *Role*), *Role* is the only element that functionally determines the other two. Therefore, while from a natural language perspective it is easier to talk about objects and facts, from the metamodel perspective it is natural to use *Role* as the pivotal meta element for constraints and paths because the associated *ObjectType* and *FactType* information is readily available.

A primary goal with role paths is to reuse the same underlying path notion as the basis for multiple constructs (constraint join paths, subtype derivation, fact type derivation, and complex constraint definitions). To support this reuse, we need a basic path metamodel that, given an instance of a starting object type, fully defines a restriction over fact and object instances that are part of a path starting at that object type. The metamodel includes a primary 'structural' section for defining connected role paths, and a smaller 'binding' section that uses role paths for capturing derivation rules. The binding metafragment for subtype and fact type derivations is shown in Figure 2. The structural metafragment is shown in Figure 3.

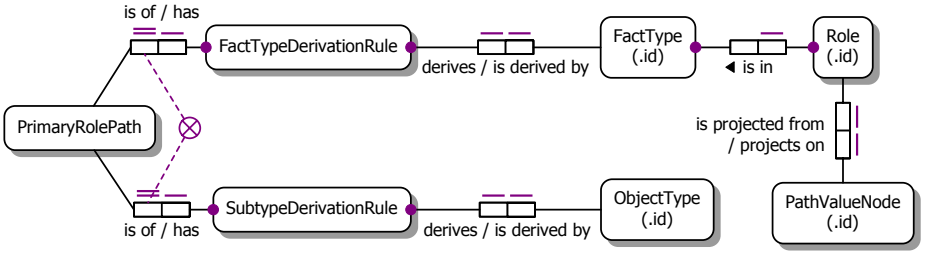

Fig. 2. Binding metamodel using subtypes of PrimaryRolePath to define subtype and fact type derivation rules. Some additional restrictions on the paths apply. For example, the root object type of the SubtypeDerivationRule must be a direct or indirect supertype of the derived subtype. PrimaryRolePath and PathValueNode details are in Figures 3 and 4.

3.1 Path Structure

A role path represents a traversal of related roles, starting with one or more roles connected to a root object type. Each subsequent role in a path either is a role in the same fact type as the previous role, or involves a join operation to a role with the same role player. Path splits represent branches along multiple continuations of the path, combined with a logical operator. The structure is easily validated as the model evolves because adjacent path roles must be related via a shared fact type or role player.

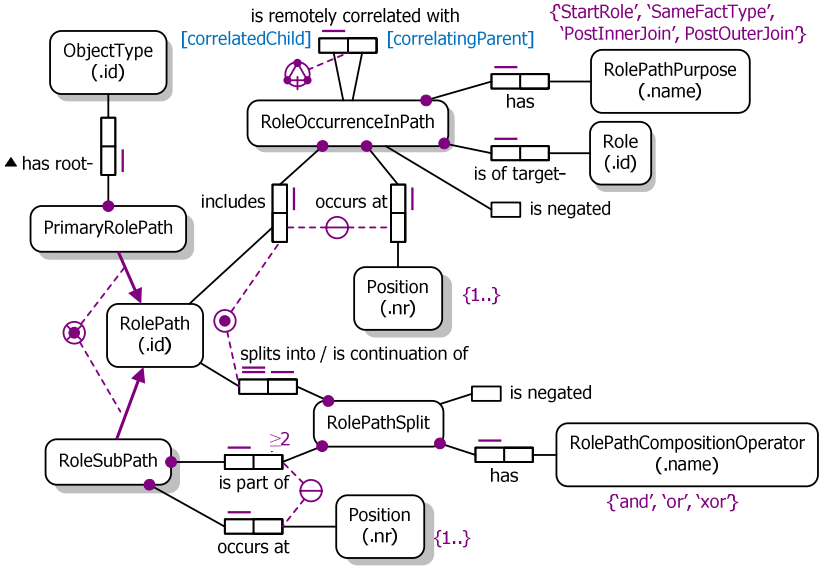

Fig. 3. Structure metamodel combining Role elements into primary paths and subpaths using tail-split semantics. Logical modifiers indicating negation and split combinations are shown, along with correlation capabilities to link elements from different branches, and role usage information to indicate joins.

Some applications of role paths may be alternately modeled as a series of joins across object types, where each join has an input and output role. However, this join-centric approach is less flexible because roles that are not involved in a join are difficult to talk about. We indirectly incorporated the notion of join in our model by allowing a role used in the path to be marked as *post inner join* or *post outer join*. The corresponding join-over-object-type notion is trivially derived using the previous role in the path and the shared role player.

Two important pathing requirements are to split a path and to equate (or correlate) elements from different parts of the path. With the ability to correlate, there is no need for an alternate mechanism to rejoin a split path. We propose a simple model using a tail split mechanism, meaning that a path either ends outright or is split into two or more subpaths combined with a logical operator. Each subpath is treated as a continuation of the path it splits from, so role information is not repeated.

Two classes of path information may be implied in a role path. The first relates to the preferred identification scheme, since inferring identifier information allows changes to the identification scheme without modification of the role path. For example, if we correlate two *RoleOccurrenceInPath* instances attached to entity types, this is interpreted as a correlation based on the identification scheme of the entity type without explicitly including roles from the identifying fact types in the role path. The second class of implied information is found in subtype graphs. If a join role occurs after a role with a role player that is a supertype or subtype of the role player for the joined role, then any subtyping links relating the two types are implied as part of the path. While the design allows direct use of identification roles or subtyping roles,

allowing these parts of the path to be implied enhances semantic stability by permitting the identification schemes and subtype graphs to be modified without affecting attached role paths.

Before applying further conditions to a role path, a simple example to demonstrate a basic subtype definition using a role path is in order. For compact role path representations, we use four symbols prepended to object type names in typed predicates to represent path information. The four lead symbols (>>, >, >+, >?) correspond respectively to the RolePathPurpose values (StartRole, SameFactType, PostInnerJoin, and PostOuterJoin). An identifier in square brackets follows the lead symbol, and "=" is used inside the square brackets after the identifier to indicate correlation with another identifier, so ">+[ro5=ro3]*Entity*" indicates that the *Entity* role is used in the path with identifier *ro5* as the right hand role occurrence of an inner join and is remotely correlated with *ro3* (correlation is discussed in section 3.2). Subpaths are indicated via indentation with the composition operator. By convention, identifiers use an increasing numbering scheme to simultaneously indicate path order.

A basic example is a subtype definition for GrandParent using the fact type Person is a parent of Person. In this case, GrandParent may be defined using a role path as:

Starting with: Person
>>[ro1]Person is a parent of >[ro2]Person
>+[ro3]Person is a parent of Person

By applying the simple rule that the path population requires a non-empty population of all roles before the first outer join (there is no outer join in this case), this three step path provides the equivalent of the FORML representation of the same rule: **Each** GrandParent **is a** Person **who** is a parent of **some** Person **who** is a parent of **some** Person. We chose forward predicate text for clarity, the GrandParent derivation rule may be expressed using the alternate Person is a child of Person reading. The rule using the reverse reading is represented as ">[ro2]Person is a child of >>[ro1]Person, Person is a child of >+[ro3]Person" and the corresponding FORML is the much clumsier "**Each** GrandParent **is a** Person **where some** Person_2 **is a** child of **that** Person **and some** Person_3 is a child of Person_2". By formalizing the derivation rule as a role path, ORM tool implementations can automatically verbalize the most readable form of the derivation rule for the current state of the model.

3.2 Calculations and Conditions

The usefulness of role paths is severely restricted if we are limited to population-based sets. Clearly, we require the ability to restrict the path population based on conditions applied to candidate path instances, and we need both value-based operations (numeric operators and functions) and bag-based operations (aggregation functions) to support common query scenarios.

The first thing we need to do is define what we mean by a *function*. In the context of this discussion, a function takes zero or more inputs and returns a single value. A function input may be either a single value or a bag. A brief discussion on how we are *not* modeling functions provides perspective.

It is common to think of functions simply as special fact types in an ORM model. These *algorithmic fact types* take two forms: comparison operators, modeled as binary fact types with a spanning uniqueness constraint; and functions, modeled with a uniqueness constraint spanning the input roles and the remaining role representing the

calculated output. We chose *not* to model functions this way because the approach does not properly model bag inputs, it requires special semantics for functions with single-valued inputs, and it requires meta-relationships between the fact type and a function implementation specification like the one we've defined.

The semantic differences between asserted or derived fact types and algorithmic fact types occur in the areas of population, role player type, and unary fact type interpretations. Unlike normal fact types, algorithmic fact types are implicitly populated only: if a role in a derived fact instance is calculated using *5 * 10 = 50*, then there is no requirement for a (5, 10, 50) tuple to be stored or derivable in the model. Algorithmic fact types are also heavily overloaded, with different implementations required for different role player combinations. Overloading issues can be resolved by introducing a *Thing* object type, but this requires special handling by type compatibility rules, which assume top-level types are disjoint. Finally, a nullary function (with no input) is not the same as an asserted unary fact type. For example, calling the *Today* nullary function is not the same as recording today's date and asserting an 'is today' unary fact type. Such a fact type needs an external data source to be correctly populated and updated. The *Today* function, on the other hand, is valid without ongoing maintenance.

Given these limitations of algorithmic fact types, we chose an alternate approach for metamodeling calculated values (see Figure 4). This provides for function inputs of either bags or single values, and single-valued outputs. From the metamodel perspective, we treat all comparison operators (=, !=, >, etc.), mathematical operators (+,–,×,÷), simple functions (sine, cosine, etc.), and aggregation functions (sum, avg, min, max, count, count distinct, etc.) in the same way, working under the assumption that these different classes of operations will be verbalized and parsed in their standard infix or functional notations. Apart from support for Boolean results, we ignored data types associated with function inputs and outputs, and assumed that calculated values are implicitly typed. A full discussion on typed function inputs based on the data types of associated role players and other factors is out of scope for this paper.

The notion of function scope is used to provide a starting point in the path to relate multiple input roles, or to determine the contents of bag input roles. For a function with multiple single-value inputs, such as the *multiply* function that calculates the LineItem.subTotal shown later in Figure 5, the scope of a LineItem role relates the *price* and *quantity* values for that LineItem and allows a tool to use the ORM uniqueness constraint patterns to test whether, given any LineItem, there is at most one price and value to satisfy the single-value input requirements of the function, or whether multiple derived fact instances are needed to represent the result. For bag functions (count, sum, etc.), the scope determines the number of instances included in the bag. For example, given the model in Figure 3, the request count(RoleSubPath) has multiple interpretations, depending on the path. A scope of a parent RolePath specifies a count of all child subpaths, and a scope of a PrimaryRolePath indicates all RoleSubPath instances recursively contained within in the path. In the first case, multiple count values are calculated for each PrimaryRolePath, whereas a single equal or larger count will be produced for the broader scope. We can also get a global RoleSubPath count with a path starting at RolePath and participating in the supertype instance role that is automatically defined for each subtype link.

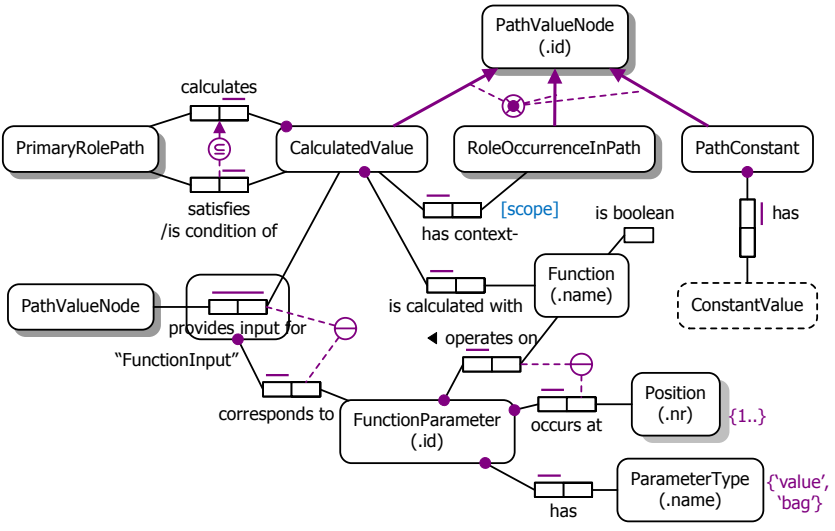

Fig. 4. PathValueNode encompasses raw instance data, calculated data, and constant data. All values can be input to function evaluations, projected as outputs for queries and derived fact types, or used as Boolean conditions to be satisfied by instances in the populated role path.

Fortunately, the role path provides a natural scoping mechanism for function evaluation. The context for a calculation must be a role occurrence on a direct or correlated role path that is directly linked to all role occurrences used as inputs to the function. In a role path, correlated role occurrences correspond to variable names in FORML and other text-based representations. The rules for variable generation are based on the RolePathPurpose values: all StartRole nodes correspond to one variable, each SameFactType node gets its own variable and each Post*Join node gets the same variable as the node before it. The RoleOccurrenceInPath is remotely correlated with RoleOccurrenceInPath fact type is used for explicit correlation across splits in the path. See Figure 7 later for an example using remote correlation to combine naturally disjoint variables.

4 Capturing Business Rules

We now consider some examples using role paths to define derived fact types. Functions and fact type derivation require two additional constructs to our textual shorthand: calculated values are represented using *FunctionName[calculationId* ***of*** *scopeId](input1, input2,...)*, and binding of derived roles with *Derivation: RolePlayer1=****PathValueNodeId*** *predicate text RolePlayer2=****PathValueNodeId2***. If function scope is omitted, then the initial instance of the root object type is used as the default scope. We demonstrate calculations in derived fact types with Figure 5, stability of a role path over model changes in Figure 6, and a remote correlation case in Figure 7.

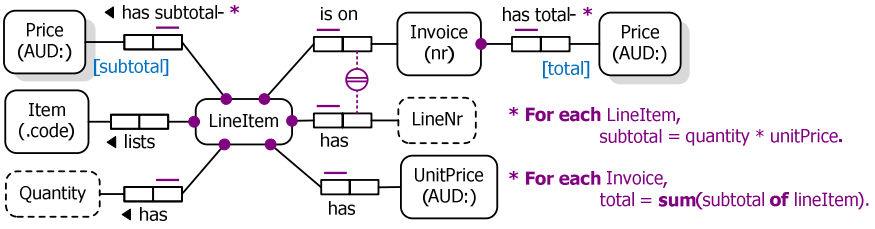

Fig. 5. Sample model with derived fact types containing calculated roles

The first task is to derive LineItem has subtotal- Price. The default scope is sufficient to multiply Quantity and UnitPrice, each of which occurs once for each LineItem.

```
Starting with: LineItem
and
  >>[ro1]LineItem has >[ro2]UnitPrice; (; means end of sub path)
  >>[ro3]LineItem has >[ro4]Quantity
multiply[f1](ro2, ro4)
Derivation: LineItem=ro1 has subtotal- Price=f1
```

The total price calculation needs a scope with a multi-valued path step between the scope and input role occurrence. Multiplicity of start or join role occurrences is determined with the FactType's uniqueness pattern. Multiplicity within the same FactType is *one* because ORM facts have single-value role players. The transition from Invoice into ro1 is the only multi-valued step; therefore *sum* uses the default scope.

```
Starting with: Invoice
>[ro2]LineItem is on >>[ro1]Invoice
>+[ro3]LineItem has subtotal- >[ro4]Price
sum[f1](ro4)
Derivation: Invoice=ro1 has total- Price=f1
```

To demonstrate nested function calls, we also show derivation of the total price without using a subtotal. In this case, the scope of the *multiply* operation produces a bag of results (one for each LineItem), which are then input to the *sum* function.

```
Starting with: Invoice
>[ro2]LineItem is on >>[ro1]Invoice
and
  >+[ro3]LineItem has >[ro4]UnitPrice;
  >+[ro5]LineItem has >[ro6]Quantity
multiply[f1 of ro2](ro4,ro6)
sum[f2](f1)
Derivation: Invoice=ro1 has total- Price=f2
```

The next example (see Figure 6) illustrates the semantic stability of our role-based approach. The initial ORM schema models student results using a ternary fact type, which is used as the basis for a subtype definition (shown in FORML). Later on, the decision is made to allow students to repeat courses and to maintain full history of their results. Using NORMA, a temporal role (r4) is inserted into the ternary and the predicate renamed as shown. Since the roles r1..r3 are unimpacted by this change, the underlying role path for the derivation rule remains the same, and the FORML verbalization is automatically updated.

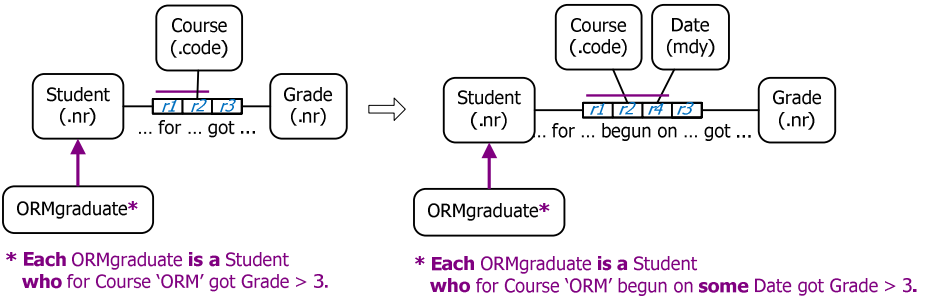

Fig. 6. A starting version of a model with a subtype derivation based on a fact type that is later extended by inserting a role for Date. Although the verbalization of the rule is different, the internal role path form of the subtype derivation rule does not change.

This is a population-based subtype derivation rule that requires an ORMgraduate student to participate in the ternary fact type with both conditions satisfied.

```
Starting with: Student
>>[ro1]Student for >[ro2]Course begun on Date got >[ro3]Grade
Satisfies:
equals[f1](ro2,'ORM')
greaterThan[f2](ro3,3)
```

Figure 7 shows a final example involving remote correlation and multiple uses of the same role in a path. In the role-based formulation, {ro1,ro3,ro5} represent the same employee instance because start roles are implicitly correlated. Similarly, {ro6,ro7,ro9} represent a second employee instance because the join operations also imply correlation. However, by default, ro2/ro8 may be different City instances and ro4/ro10 different Country instances. The explicit remote correlation (ro8=ro2) equates the City instances, and the inequality operator ensures different Country instances. With scope unspecified for the inequality function, the root object type is the default scope, which results in potentially many ro10 values for each ro4. The single-valued input on the inequality function forces a separate evaluation for each occurrence of a supervised Employee.

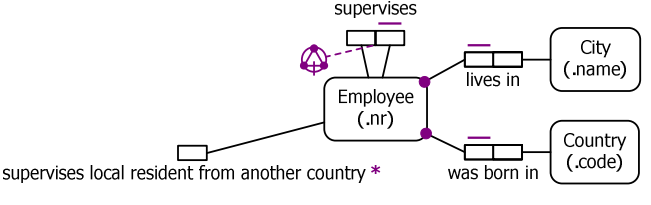

Fig. 7. A unary fact type derivation using explicit remote correlation. The definition for the unary fact type corresponds to the user-friendly query *'Who supervises an employee who lives in the same city as the supervisor but was born in a different country from the supervisor'*.

```
Starting with: Employee
and
  >>[ro1]Employee lives in >[ro2]City;
  >>[ro3]Employee was born in >[ro4]Country;
  >>[ro5]Employee supervises >[ro6]Employee
  and
    >+[ro7]Employee lives in >[ro8=ro2]City;
    >+[ro9]Employee was born in >[ro10]Country
Satisfies:
inequality[f1](ro4,ro10)
Derivation: Employee=ro1 supervises local foreigner
```

5 Conclusion

Role paths provide a highly stable, low-level representation of navigation through ORM space that can be used as a basis for specifying multiple advanced ORM constructs. This paper discussed role paths as a foundation for both subtype and fact type derivation rules. The metamodel described here is currently being implemented as the basis for formal derivation rules in the NORMA tool. Users will formulate derivation rules via high level graphical and textual views that are automatically transformed into the low level role path structures. Intelligible verbalizations of the rules will be generated on demand using the optimal available predicate text, role names, and other elements in the model. Users will also be able to specify different verbalization preferences such as *of*, *dot*, or *relational* styles [14 p. 98].

The same basic path construct may also be used as the basis for dynamic queries specified against a complete ORM model, join path specification for constraints, and complex rules that have generally been loosely classified as "textual constraints". Future research will investigate combinations of paths and normal constraints, where one or more paths act to define restricted subsets which are subject to additional standard constraints. Research into verbalization and parsing of textual representations of role paths in various languages will also be conducted, along with approaches for guided entry to assist with designating role paths.

References

1. Balsters, H., Carver, A., Halpin, T., Morgan, T.: Modeling Dynamic Rules in ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1201–1210. Springer, Heidelberg (2006)
2. Balsters, H., Halpin, T.: Formal Semantics of Dynamic Rules in ORM. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2008. LNCS, vol. 5333, pp. 699–708. Springer, Heidelberg (2008)
3. Bloesch, A., Halpin, T.: ConQuer: a conceptual query language. In: Thalheim, B. (ed.) ER 1996. LNCS, vol. 1157, pp. 121–133. Springer, Heidelberg (1996)
4. 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)
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 Press, Los Alamitos (2007)
7. 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)
8. Halpin, T.: ORM 2. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM-WS 2005. LNCS, vol. 3762, pp. 676–687. Springer, Heidelberg (2005)
9. 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)
10. 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)
11. 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)
12. 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, USA, vol. II, pp. 613–618. (2008)
13. 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)
14. Halpin, T., Morgan, T.: Information Modeling and Relational Databases, 2nd edn. Morgan Kaufmann, San Francisco (2008)
15. 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)
16. 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)
17. Meersman, R.: The RIDL Conceptual Language, Int. Centre for Information Analysis Services, Control Data Belgium, Brussels (1982)
18. Object Management Group, UML 2.0 Superstructure Specification (2003), `http://www.omg.org/uml`
19. Object Management Group (2005), UML OCL 2.0 Specification, `http://www.omg.org/docs/ptc/05-06-06.pdf`
20. Object Management Group, Semantics of Business Vocabulary and Business Rules (SBVR) (2008), `http://www.omg.org/spec/SBVR/1.0/`
21. van Bommel, P., Hoppenbrouwers, S., Proper, H., van der Weide, T.: Giving Meaning to Enterprise Architecture Principles with ORM and ORC. In: Meersman, R., Tari, Z., Herrero, P. (eds.) OTM 2006 Workshops. LNCS, vol. 4278, pp. 1138–1147. Springer, Heidelberg (2006)
22. Warmer, J., Kleppe, A.: The Object Constraint Language, 2nd edn. Addison-Wesley, Reading (2003)
23. Wintraecken, J.: The NIAM Information Analysis Method: Theory and Practice. Kluwer, Deventer (1990)