Database Schema Transformation & Optimization

T. A. Halpin and H. A. Proper

This paper was presented at OOER'95: 14th International Conference on Conceptual Modeling, Springer LNCS, vol. 1021, pp. 191-203, and is reproduced here by permission.

An application structure is best modeled first as a conceptual schema, and then mapped to an internal schema for the target DBMS. Different but equivalent conceptual schemas often map to different internal schemas, so performance may be improved by applying conceptual transformations prior to the standard mapping. This paper discusses recent advances in the theory of schema transformation and optimization within the framework of ORM (Object-Role Modeling). New aspects include object relativity, complex types, a high level transformation language and update distributivity.

Introduction

When designing an information structure for a given universe of discourse (UoD), it is best to model first at the conceptual level. This helps us to capture semantics from users and to implement the model on different platforms. Conceptual modeling may include process and behavior specification, but this paper focuses on data modeling.

Although heuristics may be used to elicit UoD descriptions from users, the same UoD might be described in many different ways. By adhering to one modeling method, the variation in description is reduced but not eliminated. This paper adopts the *Object-Role Modeling (ORM)* approach, because of its advantages over Entity-Relationship (ER) modeling (e.g. ORM facilitates communication between modeler and client, is semantically rich, and its notations are populatable). ORM pictures the UoD in terms of objects that play roles (either individually or within relationships), thus avoiding arbitrary decisions about attributes. ER views may be abstracted from ORM models when desired [3].

ORM versions include BRM (Binary-Relationship Modeling [27]), NIAM (Natural language Information Analysis Method [29]), MOON (Normalized Object-Oriented Method [10]), NORM (Natural Object-Relationship Model [8]), PSM (Predicator Set Model [22]) and FORM (Formal Object-Role Modeling [15]). An overview of ORM may be found in [16], and a detailed treatment in [15].

At the conceptual level, the basic structural unit is the elementary fact type [14], which stores information in simple units. For implementation, a conceptual schema is mapped to a logical schema, where the information is grouped into structures supported by the logical data model (relational, hierarchic, network, object-oriented etc.). For instance, the conceptual fact types might be partitioned into sets, with each set mapped to a different table in a relational database schema.

The conceptual schema also records constraints which limit the allowable fact populations, as well as derivation rules to declare how some fact types may be derived from others. These constraints and derivation rules should be mapped to corresponding constraints (e.g. foreign key clauses) and derivations (e.g. views) in the logical schema. Additional non-logical details (e.g. indexes, clustering, column order, final data types) may now be specified to complete the internal schema for implementation on the target DBMS. External schemas (e.g. form and report interfaces) may also be defined for end-users, but we ignore these in this paper.

We use the term "optimization" to describe procedures for improving the performance of a system, whether or not this is optimal in the sense of the "best possible". There are three basic phases: "conceptual optimization"; logical optimization; internal optimization. A correct conceptual schema may often be reshaped into an equivalent (or nearly equivalent) conceptual schema which maps via the standard mapping algorithm to a more efficient logical schema. We call this reshaping "conceptual optimization" since it is done at the conceptual level– of course, optimization is not a conceptual issue. At the logical level the schema may be reshaped into an equivalent logical schema giving better performance (e.g. controlled denormalization may be applied to improve query response). Finally the internal schema may be tuned to improve performance (e.g. by adding indexes).

The optimization depends on many factors, including the logical data model (relational, etc.), the access profile (query/update pattern for focused transactions), the data profile (statistics on data volumes) and the location mode (centralized, distributed, federated etc.). Although a vast literature exists on internal and logical optimization, research on conceptual optimization is still relatively new, and most of this assumes the mapping will be to a relational model. A detailed ORM-to-relational mapping procedure (Rmap) is described in [32]. Similar though less complete mapping algorithms from ER to logical models exist (see, e.g. [1]).

A formal approach to ORM was introduced in [11] and extended in [12, 13, 15]. For related work on ORM schema transformations see [9, 21, 22]. Other researchers have examined schema equivalence within the relational model [23] and the (E)ER model [7, 1, 19]. The optimization history of a schema may be seen as an evolution-worm through a model space via successive transformations. In [25] the predicate calculus-based language used in [11] to specify ORM transformations is used as a metalanguage to specify transformations between ORM and relational schemas. A general ORM-based optimization framework is specified in [2] which exploits data and access profiles as well as choices of different logical data models.

While schema transformations have many uses (e.g. to compare or translate models) in this paper we focus on their use in optimization. In the next section, a simple example illustrates conceptual optimization in ORM. The following section outlines some recent extensions, including object relativity, visualization choice and complex types. The subsequent section introduces a high level language for specifying schema transformations, and examines a basic restriction on the derivation and update rules. The conclusion summarizes the contributions and outlines future research directions.

A simple example of schema optimization

Figure 1 is a simple ORM schema. Here “hor”, “vert”, “loc”, “nat”, “CD” and “FD” abbreviate “horizontal”, “vertical”, “local”, “national”, “compact disk” and “floppy disk”. Entity types are shown as named ellipses, with their reference schemes in parenthesis. Logical predicates are displayed as box-sequences (one box for each role), with their name in mixfix notation starting at their first role-box.

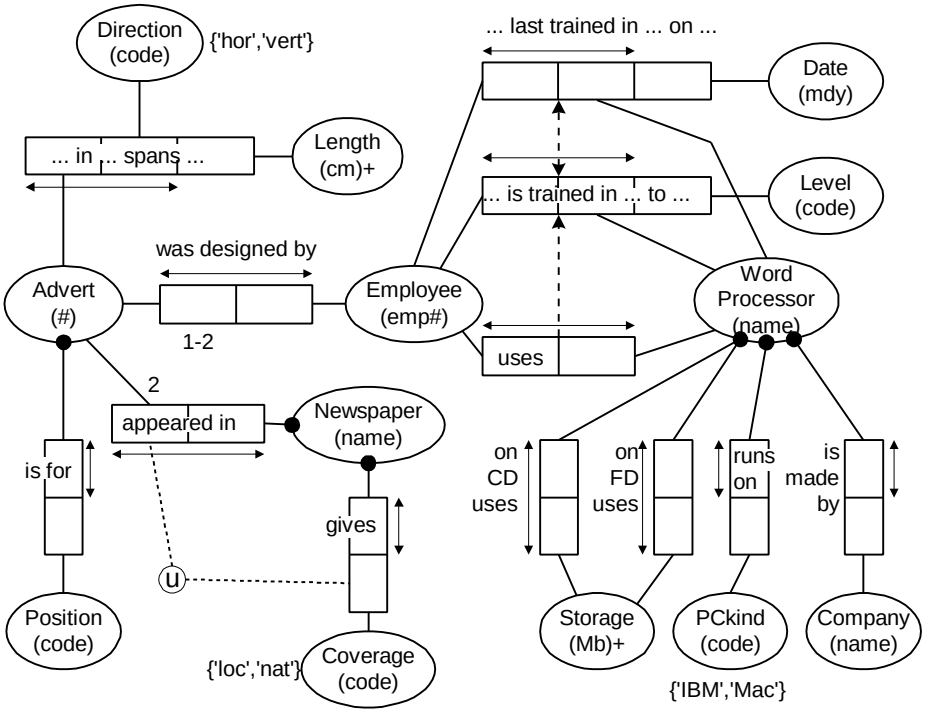


Figure 1: An unoptimized ORM schema.

An arrow-tipped bar across a role sequence indicates an internal *uniqueness constraint* (e.g. each advertisement is for at most one position, and each advertisement in a given direction spans at most one length). A circled “u” denotes an external uniqueness constraint. For example, each advert-coverage pair is associated with at most one newspaper (each advert is placed in at most one local and at most one national newspaper).

A dot where n role-arcs ($n > 0$) connect to an object type indicates the disjunction of the roles is *mandatory* (each object in the database population of that type must play at least one of those roles). For example, each advertisement is for at least one position, and each wordprocessor is stored on CD or FD. A number or number range written beside a role is a *frequency constraint* (e.g. each advertise-ment was designed by at most two employees). A list of values in braces depicts a *value-constraint* (e.g. each direction is either horizontal or vertical).

A dotted arrow from one role-sequence to another is a *subset constraint* (e.g. each employee who uses a wordprocessor is trained in it to some level). A double-headed

arrow denotes an *equality constraint* (e.g. we record an employee's training if and only if we record when this training last occurred).

Suppose the conceptual schema of Figure 1 is to be mapped to a relational schema. To avoid redundancy, the Rmap procedure [24, 15] maps each fact type with a composite internal uniqueness constraint to a separate table. Hence Figure 1 maps to eleven relation schemes, one for each fact type. For example, the top-left ternary fact type maps to the relation scheme: *AdvertSize* (advert#, direction, length). In practice a schema normally has many functional fact types which can be grouped together, but we have omitted these to demonstrate optimization with a small example. Regardless, the 11-table relational schema would typically be inefficient for most plausible query/update patterns on this UoD.

Without data and access profiles, a default optimization procedure can still be specified mainly based on reducing the number of tables (and hence joins). Figure 1 may be reshaped into Figure 2. A *predicate specialization* transformation absorbs the object types Direction, Coverage and PCkind into the relevant predicates, specializing them into separate cases, one for each value in the associated value constraint. The inverse transformation *predicate generalization* extracts the object type DiskType. A *nesting* transformation objectifies the Training relationship type. These transformations preserve equivalence.

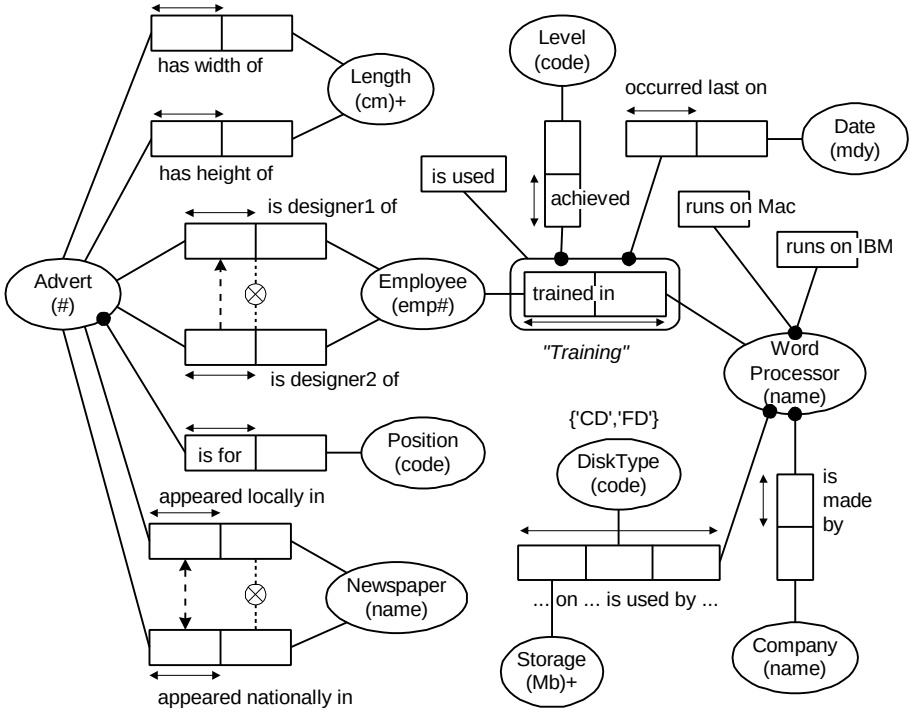


Figure 2: An "optimized" version of Figure 1.

Another predicate specialization transformation specializes the was designed by predicate into two cases, one for each frequency in the associated frequency constraint:

this transform strengthens the schema (the original schema does not tell us which of the designer1 and designer2 fact types to use for some designer).

Each of the original constraints has re-appeared as a corresponding constraint in the new schema. For example, the uniqueness constraint that each newspaper gives only one coverage (local or national) is now captured by the exclusion constraint (marked "⊗") between the newspaper roles in the second schema. The basic ORM transformation theorems include corollaries to specify the effect of additional constraints. The list of ORM transformation theorems and the associated optimization procedure to fire transformations are extensive (e.g. see ch. 9 of [15]).

Figure 3 shows the relational schema obtained by mapping the optimized schema. There are now just four tables, instead of eleven. Queries that previously involved multiple joins will now run much faster. The constraint pattern is also simplified, so updates will often be faster too. In Figure 3, keys are underlined and optional columns are placed in square brackets, as for BNF. Subset and equality constraints are marked with arrowed lines. The inequality constraint applies within the same row, while the exclusion constraint applies between the column-sets. The codes "y" and "n" are used for "yes" and "no", since many systems do not support Booleans. The mapping and optimization can be automated, but names generated for the new predicates, object types, tables and columns are not always as intuitive as those in Figures 2 and 3. The modeler should be prompted to provide better names.

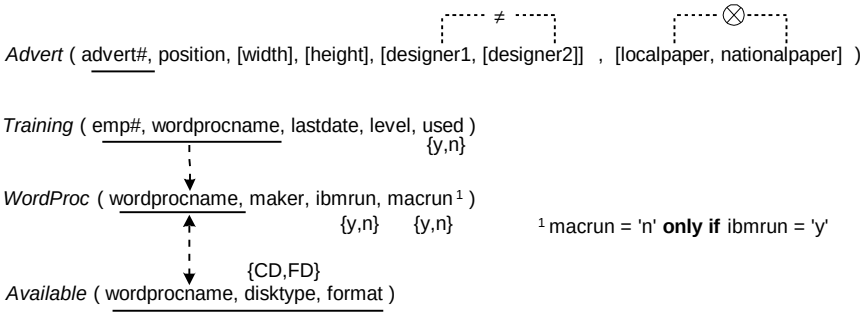


Figure 3: The relational schema obtained by Rmapping Figure 2.

Schema equivalence and modeling choices

The notion of reshaping one schema to another via an equivalence transformation is fundamental not only to optimization procedures, but also to the general problems of schema re-engineering, schema evolution and full or partial schema integration (see e.g. [4, 6, 26]). Although some rigorous treatments of schema equivalence and transformations exist for relational and ER schemas (e.g [23], [18], [28]), analyses of this notion at the conceptual level have often suffered from obscurity, limited scope or lack of rigor. It is indeed a challenge to present a theory of transformations which is sound, expressive in terms of constraint classes, and easy to understand.

Within the ORM community, the main formal contributions to schema transformation theory have been either set-based (e.g. [20]) or logic-based (e.g. [11, 9]). The approach introduced in [11] is distinct in that it specifies the precise isomorphism mappings in a way that can be proved using first order logic: its central notion is that of *contextual equivalence.* We briefly outline this approach and then refine it by introducing the notion of object relativity.

Let CS1 and CS2 be conceptual ORM schemas (possibly subschemas), each of which is expressed as a conjunction of first order sentences (an algorithm for translating any ORM schema to such a formula is specified in [11]). Let D1 be a conjunction of first order sentences defining the symbols unique to CS2 in terms of the symbols of CS1. Similarly, let D2 provide the context for defining extra symbols of CS1 in terms of CS2. Using "&"for conjunction and "⇔" for necessary equivalence, we say that CS1 is contextually equivalent to CS2 (under D1/D2) if and only if CS1 & D1 ⇔ CS2 & D2. The first order theories CS1 & D1 and CS2 & D2 provide a conservative extension ([5]) to the theories CS1 and CS2 respectively, since no new primitives are added.

The creative aspect of discovering transformation theorems is now reduced to providing the derivation rules or contexts (D1, D2) which allow the symbols in one representation to be translated into those of the other. If the two schemas are already declared it is first necessary to resolve any naming conflicts (e.g. the same symbol may have different connotations in the two schemas). For the purposes of generating a new, more optimal schema from a given schema, this problem does not arise.

For example, consider the transformation between the top-left ternary in Figure 1 and the two top-left binaries in Figure 2. The following derivation context may be added to the first schema:

Advert has width of Length	**iff**	Advert in Direction 'hor' spans Length
Advert has height of Length	**iff**	Advert in Direction 'ver' spans Length

and the following derivation context may be added to the second schema:

Advert in Direction spans Length **iff**
Advert has width of Length **and** Direction has DirectionCode 'hor'
or
Advert has height of Length **and** Direction has DirectionCode 'ver'

The second schema requires the introduction of the object type Direction, along with its reference scheme. In [11] any object type unique to one schema is explicitly introduced into the other schema before the equivalence test is conducted. This was because the equivalence proofs were conducted using a first order technique (deduction trees, i.e. semantic tableaux amenable to natural deduction). At that time, proofs were done manually. Later a theorem-prover was built which enabled typical equivalence proofs or counterexamples to be generated in under one second. In rare cases, the expressibility of ORM encroached on some undecidable fragments of predicate logic. A modified logic is currently under development in which it is hoped to avoid these problems by incorporating some finiteness axioms.

In revising the underlying logic, it has been decided to abandon, in one sense, the normal requirement of classical logic that equivalent theories must match in their domain of individuals. This domain-matching requirement is unrealistic, since it presumes an absolute theory of objects. For modeling purposes, objects rather lie in the eye of the beholder. Consider a UoD populated by a single advertisement of length 3 cm and height 5 cm. Ignoring the values (e.g. numbers) used to reference entities, how many entities are in this UoD? A person who models this world in terms of the two binary fact types will "see" three entities: one advertisement and two lengths. But a person who models this same world in terms of the ternary fact type will "see" five entities: one advertisement, two directions, and two lengths.

It is thus relative to the observer whether directions are to be thought of as objects. This is what we mean by *object relativity*. Seen in this light, objects that occur explicitly in only one schema (viewpoint) may always be implicitly introduced in the other schema. After this is done, classical proof techniques may still be used.

We now sketch some of our recent work on "conceptual" optimization. Previously, we simply replaced the original conceptual schema by the optimized version. However, practical experience has shown that sometimes a conceptual transformation used for optimization may make the conceptual schema harder for a modeler to understand. This tends to be more dependent on the modeler than the transformation, though nesting transformations often appear to be involved. For example, some modelers would prefer to see the three fact types which associate Employee and WordProcessor in Figure 1 just like that. To them, the nested version shown in Figure 2 seems more awkward. We now believe it is best to present the modelers with the optimized version of a schema fragment and allow them to adopt or reject this as a way of visualizing the UoD. If they retain the previous version, they should be given the option of having the optimization step being performed "under the covers" before the schema is passed to the standard mapper. So one "truly conceptual" schema may be used for clarity and another for efficiency.

Annotations may be used to declare all "non-conceptual" decisions which impact on the ultimate mapping (e.g. overriding of default mapping choices for 1:1 cases and subtyping, controlled denormalization decisions etc.), allowing an exact correspondence with the resulting logical schema. As the modeler will not always wish to view these annotations, their display should be toggled.

Recent extensions to ORM include join-constraints and constructors for complex object types [15, 21]. Join constraints allow set-comparison constraints to be specified between compatible role-sequences spanning any number of predicates. Apart from the need to cater for these constraints, it is possible to use this notation to specify many derivation rules, including portions of various translation contexts. Work is also underway to refine the textual version of the ORM language. The textual version is more expressive than the graphical version, and sometimes a graphic constraint is transformed into a textual constraint, or vice versa.

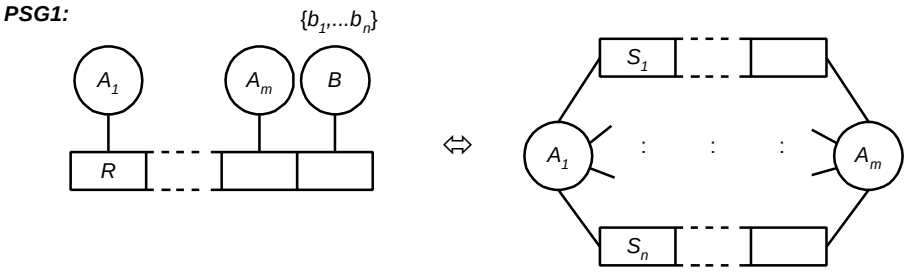
Complex types allow compound fact types to be specified directly on the conceptual schema. For example sets, lists and bags may now be modeled directly. New classes of

transformations arise. In [17]. we discuss a simple example of modeling choices involving set constructors. Apart from the extra transformation theorems needed to license the shift between viewpoints, the schema design discipline needs to be extended to guide modelers in their use of constructors.

A transformation language

In previous work, individual transformation theorems were specified in a combined graphical / textual language based on predicate-calculus, but for compact communication of transformation classes, an informal metalanguage was used. As a foundation for automated support, we now sketch a formal language to define schema transformations, with appropriate attention to update aspects.

A general format is introduced in which classes of schema transformations can be defined using a *data schema transformation scheme*. When applying such a transformation scheme to a concrete data schema, the transformation scheme is interpreted in the context of that schema, leading to a concrete transformation. A schema transformation scheme is typically centered around a set of parameters that have to be instantiated with actual components from the particular application data schema that is to be transformed.



where $m \geq 1$, and each S_i corresponds to R where $B = b_i$

Figure 4: A basic schema equivalence theorem.

Figure 4 depicts one equivalence theorem which allows a predicate to be specialized by absorbing an enumerated object type (transform to right), or for compatible predicates to be generalized by extracting this object type (transform to left). The transformation of the advertisement ternary in Figure 1 to two binaries was one application of this theorem. Another application is shown in Figure 5. The binaries in (b) may be generalized into one ternary by extracting MedalKind.

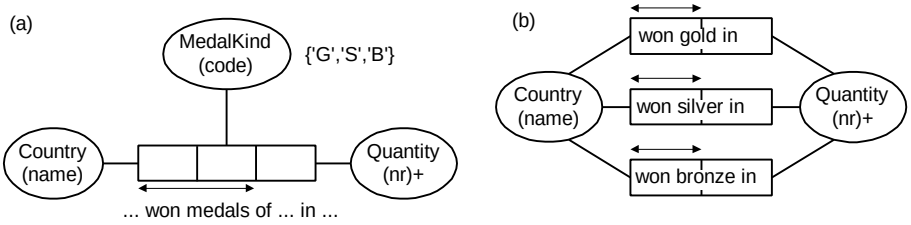


Figure 5: Three binaries (b) transform to a ternary (a) by extracting object type MedalKind.

The transformation from right to left in Figure 4 may be set out textually as in Table 1. The parameters to this transformation schema are $x!n$; $(r!n)!m$; $s!n$; u; y; l; d; $v!m$. This is a set of parameters with a variable size.

Table 1: An example transformation scheme (transforming right to left in Figure 4).

TransSchema OTExtraction ($x!n$; $(r!n)!m$; $s!n$; u; y; l; d; $v!m$);
  Type: $x!n$;
  RelType: $(f = [(x : r)!n])!m$; $h = [(x : s)!n; y : u]$;
  EntityType: $(l : d)$;
  Constraints: $c1$: **each** l **is in** $v!m$;
  From: $f!m$;
    To: y; l; d; h; $c1$;
    DerRules: $(f =$ **proj** $[(r = s)!n]$ **sel**$[u = v]$ $h)!m$;
  UpdRules: $h =$ **union of** (**proj** $[(s = r)!n; u = v]f$ $)!m$;
End TransSchema.

The expression $x!n$ denotes the list of parameters $x_1,...,x_n$, where n is not *a priori* known. Analogously, $(r!n)!m$ denotes the sequence of parameters: $r_{1,1},...,r_{n,1},...,...,r_{1,m},...,r_{n,m}$. So this transformation scheme takes $n + nm + n + 4 = n(m + 2) + 4$ parameters, where n and m are determined at application time. A concrete example is set out below. This depicts the transformation from (b) to (a) in Figure 5 (ignoring uniqueness constraints).

OTExtraction ([Country; Quantity],
[[won-gold-in-1, won-gold-in-2], [won-silver-in-1, won-silver-in-2], [won-bronze-in-1, won-bronze-in-2]],
[won-medals-of-in-1, won-medals-of-in-3], won-medals-of-in-2, MedalKind, MedalKindcode, char,
  ['G', 'S', 'B'])

We use the convention that won-gold-in-1, won-gold-in-2 refer to the first and second roles of the fact type labelled won-gold-in respectively. In the example we have $n = 2$ and $m = 3$. In this paper we ignore the introduction of names for newly introduced object types or predicates, or with mixfix predicate slots. For the example we have the following instantiations to the transformation scheme: x_1 = Country, x_2 = Quantity, $r_{1,1}$ = won-gold-in-1, $r_{2,1}$= won-gold-in-2, $r_{1,2}$ = won-silver-in-1, $r_{2,2}$ = won-silver-in-2, $r_{1,3}$ = won-bronze-in-

1, $r_{2,3}$ = won-bronze-in-2, s_1 = won-medals-of-in-1, s_2 = won-medals-of-in-3, u = won-medals-of-in-2, y = MedalKind, l = MedalKindcode, d = char, v_1 = 'G', v_2 = 'S', v_3 = 'B'.

This allows us to further fill in the transformation scheme. The Type statement simply requires that $x_1,...,x_n$, y are some kind of type in the ORM schema. In our case we have to verify that Country and Quantity are some kind of types, which they indeed are. A RelType statement requires the presence of a proper fact type in the universe of ORM schemes. In our case we have:

1. For each $1 \leq i \leq m$ there is a relationship type f_i in the ORM universe such that Roles(f_i) = $\{r_{1,i},..., r_{n,i}\}$ and $\forall_{1 \leq j \leq n}$ [Player($r_{j,i}$) = x_j]. The function Roles returns the roles in a fact type, whereas Player returns the player of the role.

2. There is a relationship g in the ORM universe such that Roles(g) = $\{s_1,...,s_n\}$ and $\forall_{1 \leq j \leq n}$ [Player(s_j) = x_j].

It is easy to verify that this holds for our example with: f_1 = won-gold-in; f_2 = won-silver-in; f_3 = won-bronze-in; g = won-medals-of-in. The EntityType statement requires y to be an entity type with label type l and domain d. In our example we require that MedalKind is an entity type identified through value type MedalKindcode, where this value type has the domain char. In the example this is indeed the case.

With the Constraints statements we capture the requirement that there exists a constraint $c1$ in the universe with definition: each l is in $v!m$. In our example case this becomes: each MedalKindcode is in {'G', 'S', 'B'}. Then there is a listing of components that are to be replaced (From) by the transformation; in our case: won-gold-in, won-silver-in, and won-bronze-in. Similarly the To statement lists the components added by the transformation. In our example these components are: MedalKind, MedalKindcode, char, won-medals-of-in.

Finally, the UpdRules and DerRules provide the translation of populations between the schema before and after the transformation. We have specified these rules in a relational algebra like language. In this article we are not concerned with a concrete language for these purposes. The completely substituted transformation is shown in Table 2.

Table 2: The example transformation

```
TransSchema OTExtraction;
Type:          Country, Quantity;
RelType:       won-gold-in =      Country:won-gold-in-1, Quantity:won-gold-in-2;
               won-silver-in =    Country:won-silver-in-1, Quantity:won-silver-in-2;
               won-bronze-in =    Country:won-bronze-in-1, Quantity:won-bronze-in-2;
               won-medals-of-in = Country:won-medals-of-in-1, Quantity:won-medals-of-in-3,
                                  MedalKind:won-medals-of-in-2;
EntityType:    MedalKind(MedalKindcode:char);
  Constraints: c1: each MedalKindcode is in 'G', 'S', 'B';
  From:        won-gold-in, won-silver-in, won-bronze-in
  To:          MedalKind, MedalKindcode, char, won-medals-of-in, c1;
DerRules:
   won-gold-in =   proj[won-gold-in-1 = won-medals-of-in-1, won-gold-in-2 = won-medals-of-in-3]
```

```
                    sel[won-medals-of-in-2 = 'G'] won-medals-of-in
won-silver-in =  proj[won-silver-in-1 = won-medals-of-in-1, won-silver-in-2 = won-medals-of-in-3]
                    sel[won-medals-of-in-2 = 'S'] won-medals-of-in
won-bronze-in = proj[won-bronze-in-1 = won-medals-of-in-1, won-bronze-in-2 = won-medals-of-in-3]
                    sel[won-medals-of-in-2 = 'B'] won-medals-of-in
UpdRules:
won-medals-of-in =  proj[won-medals-of-in-1 = won-gold-in-1, won-medals-of-in-3 = won-gold-in-2,
                          won-medals-of-in-2 = 'G'] won-gold-in
                    union
                    proj[won-medals-of-in-1 = won-silver-in-1, won-medals-of-in-3 =
                won-silver-in-2, won-medals-of-in-2 = 'S'] won-silver-in
                    union
                    proj[won-medals-of-in-1 = won-bronze-in-1, won-medals-of-in-3 =
                    won-bronze-in-2, won-medals-of-in-2 = 'B'] won-bronze-in
End TransSchema.
```

The transformation scheme does not cater for the transformation of the unique-ness constraints on the relationships involved in the transformation. In [11] and [15] constraint transformation was covered by corollaries to the basic schema transformations. While useful, this approach leads to many corollaries to deal with different classes of constraints and it currently ignores most textual (non-graphical) constraints that must be formulated in some formal textual language. Our approach for now is to enforce the (uniqueness) constraints on the transformed relationships on the (now) derived relationships. For example, in schema (a) of Figure 5 the won-gold-in relationship is derivable, and we enforce the uniqueness of its first role on this derived relationship. Currently we are researching ways to develop a general constraint re-writing mechanism to (as much as possible) re-write constraints enforced on derivable relationships to constraints on the non-derivable base types.

Although we do not provide a formal semantics of the language used to specify the transformation schemes, we do presume the existence of three functions providing these semantics. When a precise language is defined these functions would become concrete. The (partial) functions are: From: TransSchema $\times$ ParList $\rightarrow$ SCH; To: TransSchema $\times$ ParList $\rightarrow$ SCH; Schema: TransSchema $\times$ ParList $\rightarrow$ SCH.

Here TransSchema is the language of transformation schemes, and ParList is the set of parameter lists that can be built from the roles and types in the ORM schema SCH. The three functions are partial since some combinations of transformation schemes and lists of parameters may define incorrect transformations. The From function returns the schema components that will be changed by the transformation. What exactly happens with these schema components depends on the aim with which the transformation is applied– for example: (1) select a conceptual schema alternative as a preferred way of viewing the UoD; (2) enrich an existing schema with an extra view; (3) optimize a final conceptual schema.

In case (1) the components listed in the From statement need to be removed from the existing schema. In case (2) none of the components nominated by the From statement need to be removed. In case (3) the components in the From statement will not be removed as they remain part of what the user sees as the conceptual schema. They will, however, be marked as derivable (using the specified derivation rules).

The result of the From statement is given as a (sub) schema. As this usually is a subschema without a proper context, this is not likely to be a complete and correct ORM schema. In our example the schema resulting from the From function contains only the three fact types from the (b) variation of the Olympic games schemas, without the Country and Quantity entity types.

The To function returns the added schema components as a subschema. This is usually incomplete since it misses the proper context. The To function returns the components listed by the To statement, and also returns the derivation and update rules (in the resulting schema these rules are required to derive the instances of the transformed types and translate the updates of the transformed types to updates of the new types). Finally, the Schema function yields all schema components listed in the transformation scheme, and returns this as the schema. However, the update rules are ignored. The resulting schema is the context of the schema transformation.

When transforming a conceptual schema to another data schema, the user will still want to perform the updates and queries as if they are done on the original conceptual schema (not the optimized schema). This is why we added the derivation and update rules– they allow us to define the official conceptual schema as an updatable view on the actually implemented schema. In supporting this approach however, we must avoid the view update problem. To allow the user to specify updates directly on the conceptual level, the update rules must be *update distributive*. The update rules can be regarded as function μ : POP " POP that transforms a population of the original schema to a population of the actually stored data schema. The derivation rules perform the opposite function μ^{-1}, and for an equivalence preserving schema transformation this μ^{-1} is the inverse function of μ.

Let p_1, p_2 be populations of an ORM schema. We can generalize each binary operation 1 on sets (of instances) to populations as a whole by: $(p_1 \Theta p_2)(x) = p_1(x) \Theta p_2(x)$. A function μ: POP " POP is *update distributive* if and only if for $\Theta \in \{\cup, -\}$ and a correct schema SCH we have: IsPop(p, SCH) & IsPop($p \Theta x$, SCH) $\Rightarrow \mu(p \Theta x) = \mu(p) \Theta \mu(x)$. Here IsPop($p$, SCH) means that p is a correct population of schema SCH. If μ is the population transformation function following from the update rules from a given transformation scheme t, then μ must be update distributive. With such a μ we can safely translate any update of the population of the original schema to an update of the transformed schema.

Conclusion

Schema transformations at the conceptual level may be used to improve the clarity of a conceptual schema or the efficiency of the final database application. This article surveyed the state of the art on conceptual optimization, and then discussed several new contributions, including object relativity, visualization choices, complex types, a formal language for transformation schemes, and update distributivity. Though couched in terms of Object-Role Modeling, the approach may be adapted to Entity-Relationship Modeling so long as a supplementary textual language is available to specify domains, as well as ORM constraints and derivation rules.

We are developing a constraint re-writing mechanism to re-write constraints enforced on derivable types to constraints on fundamental types. This is needed to optimize the enforcement of constraints and to support the mapping of data schemas to internal schemas. Further heuristics and algorithms are being developed to cater for transformation and optimization of additional constructs at the conceptual level, and mapping to different logical data models. The ideas presented in this article will be implemented in a prototype schema transformation and optimization tool.

References

1. Batini, C., Ceri, S. & Navathe, S.B. 1992, *Conceptual database design: an entity-relationship approach*, Benjamin / Cummings, Redwood City CA, USA.

2. Bommel, P. van & Weide, Th.P. van der 1992, 'Reducing the search space for conceptual schema transformation', *Data and Knowledge Engineering*, v.8, pp. 269-92.

3. Campbell, L. & Halpin, T.A. 1994a, 'Abstraction techniques for conceptual schemas', *ADC '94: Proc. 5th Australasian Database Conf.*, World Scientific, Singapore.

4. Campbell, L. & Halpin, T.A. 1994b, 'The reverse engineering of relational databases', *Proc. 5th Workshop on Next Generation CASE Tools*, Utrecht, The Netherlands (June).

5. Chang, C.C. & Keisler, H.J. 1977, *Model theory*, 2nd edn, North-Holland, Amsterdam.

6. Dupont, Y. 1994, 'Resolving fragmentation conflicts in schema integration', *Proc. 13th Entity-Relationship Conf.*, Springer-Verlag LNCS vol. 881, pp. 513-32.

7. D'Atri, A. & Sacca, D. 1984, 'Equivalence and mapping of database schemas', *Proc. 10th Int. Conf. On Very Large Databases*, VLDB, Singapore, pp. 187-95.

8. De Troyer, O.M.F. 1991, 'The OO-Binary Relationship Model: a truly object-oriented conceptual model', *Proc. CAiSE-91*, Springer-Verlag LNCS, no. 498, Trondheim.

9. De Troyer, O.M.F. 1993, 'On data schema transformations', PhD thesis, University of Tilburg (K.U.B.), Tilburg, The Netherlands.

10. Habrias, H. 1993, 'Normalized Object Oriented Method', in *Encyclopedia of Microcomputers*, vol. 12, Marcel Dekker, New York, pp. 271-85.

11. Halpin, T.A. 1989, 'A Logical Analysis of Information Systems: static aspects of the data-oriented perspective', PhD thesis, University of Queensland, Brisbane, Australia.

12. Halpin, T.A. 1991, 'A fact-oriented approach to schema transformation', *Proc. MFDBS-91*, Spinger-Verlag LNCS, no. 495, Rostock, Germany.

13. Halpin, T.A. 1992, 'Fact-Oriented Schema Optimization', *Proc. CISMOD-92*, pp. 288-302, Indian Institute of Science, Bangalore, India.

14. Halpin, T.A. 1993, 'What is an elementary fact?', *Proc. First NIAM-ISDM Conf.*, eds G.M. Nijssen & J. Sharp, Utrecht, The Netherlands (Sep).

15. Halpin, T.A. 1995, *Conceptual Schema and Relational Database Design*, 2nd edn, Prentice Hall, Sydney, Australia.

16. Halpin. T.A. & Orlowska, M.E. 1992, 'Fact-Oriented Modelling for Data Analysis', *Journal of Inform. Systems*, vol. 2, no. 2, pp. 1-23, Blackwell Scientific, Oxford.

17. Halpin, T.A. & Proper, H.A. 1995, 'Subtyping and polymorphism in Object-Role Modeling', *Data and Knowledge Engineering*, vol. 15, pp. 251-281, Elsevier Science.

18. Hainut, J-L 1991, 'Entity-generating schema transformation for entity-relationaship models', *Proc. 10th Entity-Relationship Conf.*, San Mateo (CA), North-Holland, 1992.

19. Hainaut, J-L., Englebert, V., Henrard, J., Hick, J-M., Roland, D. 1994, 'Database evolution: the DB-MAIN approach', *Proc. 13th ER Conf.*, LNCS vol. 881, pp. 112-31.

20. Hofstede, A.H.M. ter, Proper, H.A. & Weide, Th.P. van der 1993, 'A note on schema equivalence', *Tech. Report 92-30*, Dept of Inf. Systems, University of Nijmegen.

21. Hofstede, A.H.M. ter, 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.

22. Hofstede, A.H.M. ter & Weide, Th.P. van der 1993, 'Expressiveness in conceptual data modelling', *Data and Knowledge Engineering*, vol. 10, no. 1, pp. 65-100.

23. Kobayashi, I. 1986, 'Losslessness and semantic correctness of database schema transformation: another look at schema equivalence', *Information Systems*, 11 (41-49).

24. Ritson, P.R. & Halpin, T.A. 1993, 'Mapping Integrity Constraints to a Relational Schema', *Proc. 4th ACIS*, Brisbane, Australia (Sep.), pp. 381-400.

25. Ritson, P.R, 1994, 'Use of conceptual schemas for a relational implementation', PhD thesis, University of Queensland, Brisbane, Australia.

26. Shoval, P. & Shreiber, N. 1993, 'Database reverse engineering: from the relational to the binary relational model', *Data and Knowledge Engineering*, vol. 10, pp. 293-315.

27. Shoval, P. & Zohn, S. 1991, 'Binary-relationship integration methodology', *Data and Knowledge Engineering*, vol. 6, no. 3, pp. 225-50.

28. Thalheim, B. 1994, 'State-conditioned semantics in databases', Proc. 13th Int. Conf. On the Entity-Relationship Approach, Springer-Verlag LNCS, vol. 881, pp. 171-8.

29. Wintraecken, J.J.V.R. 1990, The NIAM Information Analysis Method: Theory and Practice, Kluwer, Deventer, The Netherlands.