Automated mapping of conceptual schemas to relational schemas

J.I. McCormack, T.A. Halpin and P.R. Ritson

Key Centre for Software Technology
Department of Computer Science
University of Queensland. Australia 4072
email: halpin@cs.uq.oz.au; jonmac@cs.uq.oz.au

Abstract: Many CASE tools for information systems engineering can input a conceptual data model of an application and map this to a logical data model for implementation. Typically this involves mapping an ER (Entity-Relationship) conceptual schema to a relational database schema. Since the graphic notation of ER, or the mapping algorithm itself, fails to capture many constraints and derivation rules, these additional features must be coded up manually. Object-Role Modelling (ORM) provides a simpler and richer notation, enabling most of these additional features to be catered for in the mapping. The most well known version of ORM is NIAM, and a number of CASE tools now support this method. Recently, an extended ORM language called FORML has been developed which is even more expressive, and a complete mapping algorithm has been developed and automated. This paper provides an overview of the mapping algorithm and the use of role-graphs for automation.

1 Introduction

CASE tools are being increasingly used to support various phases of information systems development, from requirements analysis through conceptual modelling, logical, physical and external design and implementation, testing and maintenance (Ovum 1992). This paper focuses on the phase where a conceptual schema is mapped to a logical schema. The process and behaviour-oriented perspectives (Olle et al. 1991) are ignored, and for the target logical model we consider only the relational data model. In spite of some views to the contrary (e.g. Codd 1990), the relational model is regarded as sub-conceptual, since it is too far removed from natural concepts (ISO 1982). The most popular conceptual approach to data modelling is entity-relationship modelling (ER), of which several dozen versions exist. The majority of information systems engineering tools use a version of ER to model the data perspective. For an overview of such tools see Ovum (1992) and Reiner (1992). A state-of-the-art example is discussed in Czejdo et al. (1990).

While ER diagrams provide one useful way of summarizing the main features of an application's data model, they are typically unable to express many constraints and derivation rules that commonly occur in applications. Hence the ER mappers provided in CASE tools usually generate an incomplete logical schema for the application. The features not captured in the ER model are either coded up manually or ignored.

Enhanced versions of ER have been proposed to cater for additional constraints (e.g. Rochfeld et al. 1991, Batini et al. 1992), but their notations cannot be populated with instances for validation purposes. Many algorithms exist for mapping from ER to the relational model (e.g. Hohenstein 1990, Markowitz 1990, Batini et al. 1992), but they ignore constraints such as role-sequence exclusion (e.g. a person cannot referee a paper s/he authors) and ring-constraints (e.g. parenthood is asymmetric). Some CASE tools, such as TEMPORA (Papastamatiou 1992) and CADDY (Hohenstein 1990), augment the ER model with temporal semantics, and map various dynamic constraints. A treatment of dynamic constraints is beyond the scope of this paper.

An alternative conceptual modelling approach is provided by *object-role modelling (ORM)*, also known as fact-oriented modelling. Various versions of ORM include NIAM (Natural-language Information Analysis Method), BRM (Binary-Relationship Modelling), NORM (Natural Object Relationship Model) and PSM (Predicator Set Model). Our version, FORM (Formal Object Role Modelling), is based on extensions to NIAM (Nijssen & Halpin 1989) and has an associated language FORML in both graphical and textual forms. An overview of the method is given in Halpin & Orlowska (1992).

Basically, ORM views the world as objects playing roles, either singly (e.g. Person jogs) or within a relationship (e.g. Person drives Car): it makes no direct use of the attribute concept. In contrast to ER, ORM is closer to natural language, uses diagrams that can be populated with fact instances, is simpler (no attribute construct), is more expressive (e.g. additional static constraints are captured, and semantic domains are fully revealed) and has a stronger formal basis for performing transformations. Since ORM captures so much detail, FORM includes several abstraction mechanisms for hiding information when summary views are required—one of these mechanisms makes use of attributes, providing a bridge to the ER viewpoint.

Several workbenches for object-role modelling exist, either as academic prototypes or commercial tools. The best known of these are probably RIDL* (De Troyer et al. 1988; De Troyer 1989; Nienhuys-Cheng 1990) now marketed by Intellibase, and IAST (Control Data 1982). GISD (Shoval et al. 1988) includes the ADDS (Shoval & Even-Chaime 1987) system to perform a relational mapping, but does not support subtyping. These systems, as well as another (Mark 1987) basically conform to the binary-only version of object-role modelling, though RIDL* has recently added support for fact types of higher arity. ITI (Brisbane) markets two NIAM tools known as SDD and CD. Our approach is adopted in the WISE prototype developed at the University of Queensland (Halpin 1991b), and in the commercial workbench InfoViews, marketed by ServerWare (Bellevue, WA, USA).

An "optimal normal form" (ONF) algorithm for mapping from ORM to normalized relational tables was introduced in NIAM in the 1970s. This basic algorithm ignored certain cases, and provided only a very incomplete specification of how constraints were mapped. Two of the authors have revised and extended this algorithm to provide a comprehensive relational mapping algorithm (*Rmap*) which is capable of completely mapping any conceptual schema expressed in the graphic version of FORML to a redundancy-free, relational schema (Ritson & Halpin 1992). Our approach differs from other mappers, such as RIDL-M (Intellibase 1990) by catering for a wider variety of constraints (e.g. n-ary subset, equality, exclusion, closure, and ring constraints); it also differs in its treatment of subtyping, though this is not detailed here.

The main features of the Rmap algorithm are sketched in this paper, and the notion of a *role-graph* is introduced to provide an intermediate data-structure which can be used to efficiently implement the mapping and to specify arbitrary constraints or queries on a conceptual schema. Section 2 of this paper uses a simple example to explain the main notations of ORM and the basic features of Rmap. Section 3 introduces the notion of a role-graph. Section 4 outlines the procedure for mapping predicates. Section 5 explains how role-graphs are mapped to relational views. Section 6 indicates in more detail how constraints depicted on a conceptual schema are mapped to constraints on a relational schema. Section 7 illustrates the potential use of role-graphs for capturing arbitrary conceptual expressions. The conclusion summarizes the main points and notes some related research under development.

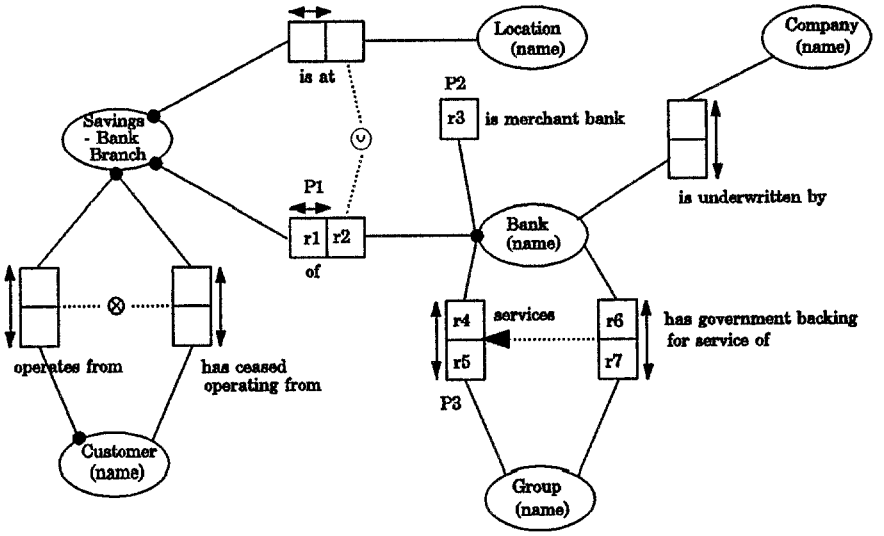


Fig. 1 An ORM conceptual schema diagram

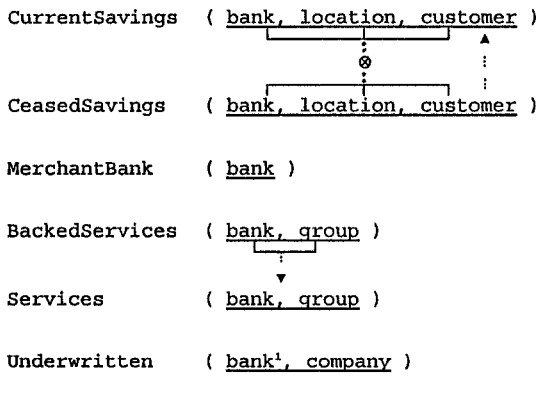
2 An overview of ORM and Rmap

An ORM conceptual schema diagram for a simple application is shown in Figure 1. Object types are depicted as named ellipses (e.g. Customer). Simple reference schemes for entity types are parenthesized (e.g. each Customer is identified by his or her name). SavingsBankBranch has a composite reference scheme, being identified by the combination of its location (e.g. Paris) and its bank (e.g. National Bank); the symbol "Ⓤ" is an external uniqueness constraint indicating that each Location, Bank combination refers to only one branch. Predicates are shown as named box-sequences (one box for each role). The unary predicate "is merchant bank" is a single role. All the other predicates here are binary (two roles). Predicates of any arity (number of roles) are allowed. Predicates are ordered, with their name starting in or beside their first role box. Each role is connected by an edge (line segment) to the object type which plays that role. For example the first role of the binary predicate "operates from" is played by Customer, and its second role by SavingsBankBranch.

A bar across a sequence of one or more roles is an internal *uniqueness* constraint (instantiating object sequences may not be duplicated); arrow tips may be added to the bar (and must be if the roles are non-contiguous). For example, each branch is at only one location, but each customer may operate from many branches. Predicates which are completely spanned by a uniqueness constraint may be objectified; this nesting is shown as a frame (see Figure 6 later). A dot where *n* role-arcs connect to an object type indicates that the disjunction of the *n* roles is *mandatory* or total (each object in the population of that type must play at least one of those roles). For example, each savings bank branch has either current customers or past customers (or both). An ⊗ symbol connecting role-sequences indicates mutual *exclusion* between the populations of these role-sequences (e.g. customers cannot operate from a branch at which they have ceased operations).

A dotted arrow from one role-sequence to another denotes a *subset* constraint (i.e. the population of the source is a subset of the target). For example, if a bank has government backing to service a group then it must service that group. A subset constraint in both directions is an equality constraint, and is shown as a dotted line with arrow-heads at both ends (no example here). A solid arrow from one object type to another indicates the former is a proper subtype of the latter (not shown here, but consider Woman as subtype of Person); subtype definitions are specified at the bottom of the diagram. Value constraints are shown in braces beside the relevant object type (not shown here, but consider Sex {'m','f'}). Other graphic constraints such as ring and closure constraints are not depicted here. For background on ORM see Nijssen & Halpin (1989) or Halpin & Orlowska (1992).

The Rmap algorithm maps the conceptual schema of Figure 1 to the relational database schema shown below. Each fact type is grouped into only one table. Composite keys map to separate tables. Simple keys attached to the same object type are grouped into the same table, keyed on the object type identifier.



[1] in CurrentSavings.bank ∪ CeasedSavings.bank ∪ MerchantBank.bank ∪ Services.bank

All constraints are mapped down. A formal treatment of complete constraint mapping is one way in which Rmap differs from the old ONF algorithm. Keys are underlined; the primary key is doubly underlined if an alternate key exists. Subset

constraints (e.g. referential integrity) are shown as dotted lines: arrow tips must be included to show the direction of the subset linkage unless both directions apply (equality constraint). The qualification index on Underwritten.bank expresses the disjunctive mandatory role constraint on Bank.

This example has been artificially constructed to demonstrate the main ideas of the role-graph procedure discussed later. In practice, subtyping of banks would be more natural, and the operations and services part of the schema would typically be optimized by conceptual transformations before the mapping (see Halpin 1991a, 1992).

3 Role-Sequences

Most static conceptual constraints can be expressed as an operator over one or more role-sequences. A role-sequence is an ordered list of roles (predicate-position pairs) contained in a connected sub-schema. The schema fragment in Figure 1 contains a subset constraint from the role-sequence $rs_1 = (r_6, r_7)$ to $rs_2 = (r_4, r_5)$. This means that each (bank, group) pair instantiating the government backing predicate must also instantiate the services predicate (i.e. if a bank has government backing to service a group then it must service that group).

The meaning of constraints containing role-sequences spanning only one predicate is well defined. Problems start to occur when a role-sequence contains roles from more than one predicate. Figure 2 contains a schema fragment with a role-sequence $rs_1 = r_6, r_8$. In the context of the constraint, this role-sequence may have a number of meanings: ⟨A,D⟩ from the natural join of P3, P5, P4; ⟨A,D⟩ from the natural join of P3, P1, P4; ⟨A,D⟩ from the natural join of P3, P2, P4; or a union or intersection of these. A further problem occurs when role-sequences involve ring predicates. In Figure 3, the interpretation of the role-sequence (r_1, r_3) is ambiguous because predicate P_2 can be joined on either r_3 or r_4 or both.

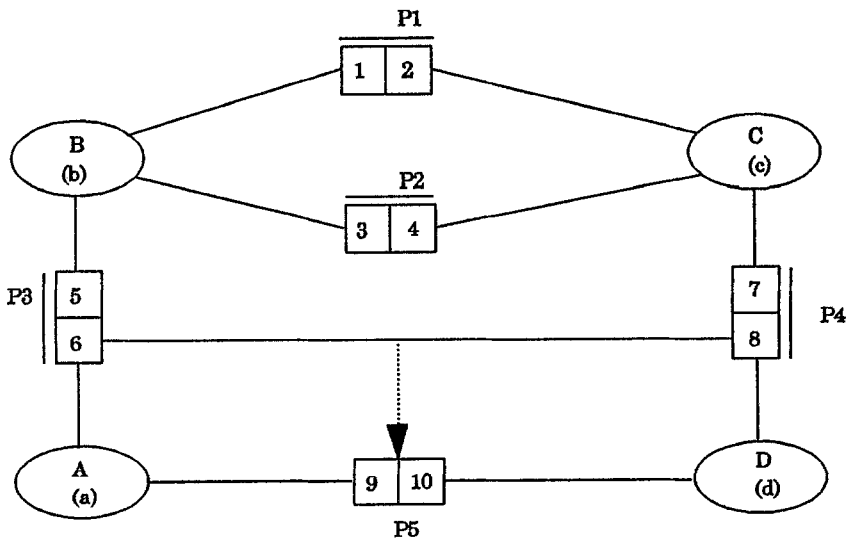


Fig. 2 A schema containing an ambiguous constraint

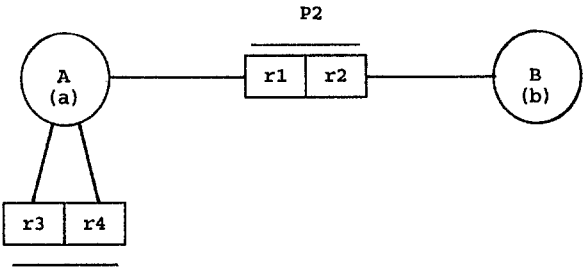


Fig. 3 Both roles of a ring binary are played by the same object type

Such ambiguities can be resolved by indicating in the definition of the role sequence how the joins are to be made. Apart from ambiguity, there is also the question of whether it makes sense to allow some role-sequence as an argument within a given constraint type (e.g. uniqueness or subsethood). Apart from obvious restrictions (e.g. roles being compared in a subset constraint should be distinct and be played by compatible object types) this paper makes no special assumptions in this regard. A separate paper in preparation addresses this question in detail.

The data structure used to represent role-sequences is called a **role-graph** and has two parts. The first is a *schema graph* which is used to specify all of the predicates involved in the schema and how they will be joined. The second part is the *role-sequence*: an ordered list of the relevant roles (this determines the order in which the which the roles are to appear in the final "selection"). We now consider this notion in more detail, with the aid of an example. A schema graph is a connected graph G = (V,E) with the following structure and restrictions:

- V is the set of vertices and E is the set of edges.
- A vertex is a predicate (P) or an object-type (O).
- An edge connects a predicate vertex to an object-type vertex, i.e. E = (P,O).
- All object-type vertices must be internal nodes of G.
- Each edge E has a role associated with it—this a role of P and denotes the join role of P for E. The join role of a E must be a role of P which is played by O in the schema.
- An edge from O to P joined on role R denotes the natural join of P with all other predicates connected to O in the schema-graph based on the population of R.

For simplicity, the following restrictions are made: no predicate may appear in the schema graph more than once; all nested object-types have been transformed into equivalent compositely identified object-types. Figure 4 contains a schema fragment, and Figure 5 shows a schema-graph and role-sequence based on it. A pseudo-SQL interpretation of the schema-graph in Figure 5 is shown below. The interesting aspects of this role-graph are: C was joined to B via P2; P5 was joined to B via r9 (not r10).

```
select r1, r8, r10 from P1, P5, P2, P4
where    (P5.r9 = P1.r2)
  and    (P1.r2 = P2.r3)
  and    (p2.R4 = p4.R7)
```

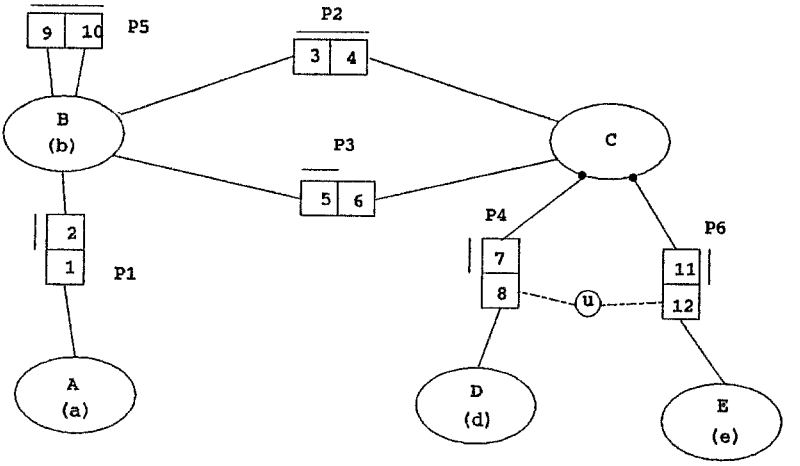


Fig. 4 A conceptual schema fragment

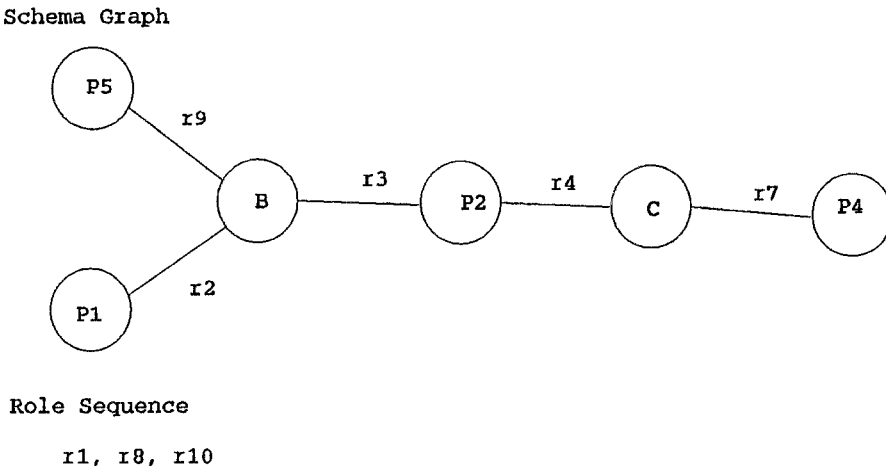


Fig. 5 A role-graph based on selecting roles 1, 8 and 10 from figure 4

4 Mapping predicates during Rmap

During Rmap, predicates are mapped to columns in relational tables. To begin with, predicates are classified as either reference predicates or fact predicates. A reference predicate is one that is involved in the identification of an object-type: the reference scheme might be simple (in which case the predicate is usually displayed implicitly in parentheses) or compound. All predicates which are not reference predicates are fact

predicates. When a predicate is mapped to a table, each of its roles maps to one or more columns in that table (more than one column is needed if the object type playing the role has a composite identification scheme). Fact predicates are only mapped once. Reference predicates however, may be mapped more than once. Figure 6 contains a meta-schema fragment for representing predicate mappings.

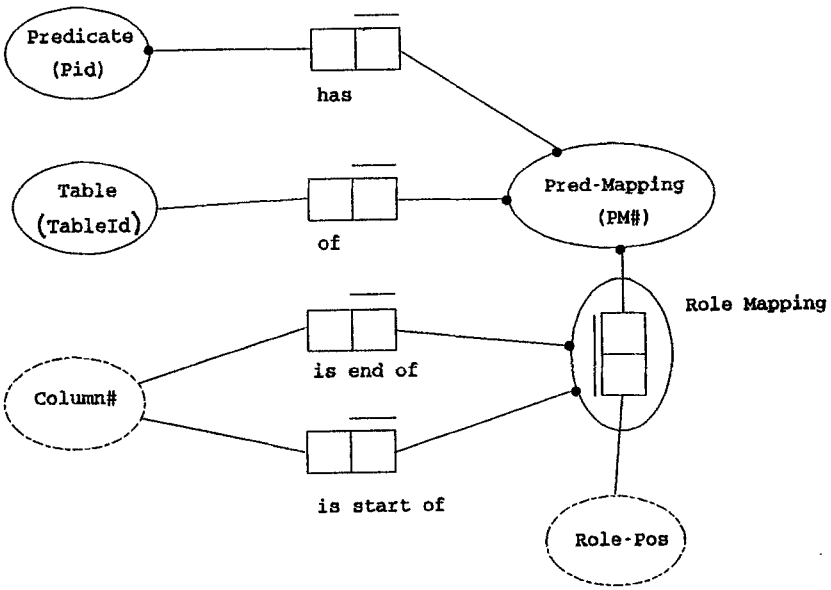


Fig. 6 The data-structure for predicate mappings

For example, the mapping of the predicate ⟨r4,r5⟩ in Figure 1 to the Services table may be recorded in this data-structure as follows:

```
roles:              r4    r5
table:    Services (bank, group)
col#:               1     2

Mapping#1:  Predicate "Bank services Group" has Mapping#1.
            Mapping#1 is to the Services table.
            Mapping#1 has two role-mappings (r4 and r5):
            The Mapping#1, r4 role-mapping starts at col# 1; ends at col# 1
            The Mapping#1, r5 role-mapping starts at col# 2; ends at col# 2
```

The basic algorithm for grouping fact types into tables is trivial, as explained earlier when Figure 1 was mapped. The main steps are clearly explained in Halpin & Orlowska (1992). Finer points on mapping of 1:1 predicates are given by Ritson & Halpin (1993), and a detailed discussion of subtype mapping is contained in Halpin, Harding & Oh (1992). As the coding of the algorithm is straightforward, further details are omitted here to save space.

As another example, Figure 4 maps to the relational schema shown below. Square brackets around a column name indicate that the column is optional (null values are allowed). In Figure 4, *C* is compositely identified by means of the reference predicates P4 and P5; hence *c* instances are referenced by the combination of *d* and *e* in the relational tables. The qualification on [*e*] ensures both components of the identifier are present in any reference of *c*. Although the algorithm is fully automatic, the generated names for tables and columns are often awkward, and these can be edited by the designer.

P1, P3	map to	B (b, a, [d], [e][1])
P5	maps to	BB (b, b1)
P2	maps to	BC (b, d, e)

[1] non-null iff d non-null

If R is the number of roles in the schema which are involved in fact predicates and D is the maximum number of reference predicates involved in the identification of an object-type, the complexity of the predicate-mapping algorithm is O(RD). Since most predicates are binary, and compositely identified object types generally constitute only a small percentage of the object-types in a schema, a practical approximation of the complexity is O(P log P) where P is the number of predicates in the schema.

5 Mapping role-graphs to relational views

The first step in mapping a role-graph to relational views is re-constructing all of the predicates involved in terms of relational constructs. This is done by creating a select statement for each predicate mapping and combining them all using the union operator to form a view equivalent to the re-constructed predicate. Once all of the predicates have been re-constructed their views are joined via the join-roles specified in the schema-graph. Finally the roles specified in the role-sequence are selected out of the join. For example, the view created for predicate P4 in Figure 4 would be:

```
create view P4view ( d, e ) as
  select d, e from  B
  where (( d is not null ) or ( e is not null ))
     and (d is not null )
union
select d, e from BC
where (( d is not null ) or ( e is not null ))
     and (d is not null )
```

These constructions generate the following select query for the schema-graph in Figure 5.

```
Predicate 1:  create view viewP1 (a, b) as
              select a, b from B where (a is not null) and (b is not null)
```

```
Predicate 2:  create view viewP2 (b,d,e) as
              select b,d,e from BC where (b is not null) and (d is not null)
                   and (e is not null)

Predicate 4:  create view P4view ( d, e ) as
              select d, e from  B where (( d is not null ) or ( e is not null ))
                   and (d is not null )
              union
              select d, e from BC where (( d is not null ) or ( e is not null ))
                   and (d is not null )

Predicate 5:  create view viewP5 (b,b1) as
              select b,b1 from BB where (b is not null) and (b1 is not null)
```

The final query is:

```
select  viewP1.A, viewP1.B, viewP2.B, viewP2.D, viewP2.E,
        viewP4.D, viewP4.E, viewP5.B, viewP5.B1
from    viewP1 , viewP2 , viewP4 , viewP5
where   (viewP5.B = viewP1.B)
     and (viewP1.B = viewP2.B)
     and (viewP2.D = viewP4.D)
     and (viewP2.E = viewP4.E)
```

Obviously there is some redundancy in the algorithm as presented so far. The general case problem of optimization is difficult. The following four "peep-hole" optimizations generally result in acceptably efficient queries. These are to be done in order (1 to 4).

1. Eliminating Unions:

If a view contains two or more unioned select statements over the same table and share them with the same selectors, replace them with a single select statement over the same selectors and table, with a where clause which is the disjunct of all of the where clauses in the original statements.

2. Eliminating Views:

If a view V_p contains no unions it must contain only one table T_p, so replace all occurrences of V_p with T_p in the role-graph's select query and conjoin the where clause of V_p with the where clause of the role-graph's select query.

3. Optimizing Joins:

If one table appears two or more times in the role-graph's select query and is joined using the same fields in each case, all of these tables but one may be removed from the from clause of the query.

4. Eliminating Where-Clauses:

If a conjunct of a where-clause is (col is not null) and col is not a nullable table, this conjunct may be removed from the where-clause.

These optimizations reduce the select query previously developed to:

```
create view P4view ( D, E ) as
     select D, E from  B where  (D is not null )
     union
     select D, E from BC

select  B.A, B.B, BC.B, BC.D, BC.E,
        viewP4.D, viewP4.E, BB.B, BB.B1
from    B, BC, BB,  viewP4
where  (BB.B = B.B)
  and  (BB = BC.B)
  and  (BC.D =  viewP4.D)
  and  (BC.E = viewP4.E)
  and  (B.A is not null)
```

The final step in mapping a role-graph is creating a view which selects only the required columns from the schema-graph's select query. For the example, this gives:

```
create view P4view ( D, E ) as
     select D, E from  B where  (D is not null )
     union
     select D, E from BC

create view FinalView ( A, B, D ) as
     select  B.A, viewP4.D, BB.B1
     from      B, BC, BB,  viewP4
     where    (BB.B = B.B)
       and    (BB = BC.B)
       and    (BC.D =  viewP4.D)
       and    (BC.E = viewP4.E)
       and    (B.A is not null)
```

6 Mapping Conceptual Constraints to Relational Constraints

Currently formalised ORM constraints include subset, equality, exclusion, mandatory role, disjunctive mandatory role, frequency, uniqueness, ring, closure and value. This section of the paper discusses how such conceptual constraints are mapped to equivalent relational constraints expressed in SQL. Before the constraints are mapped, the schema should be checked using the MSEX validation algorithm (Halpin & McCormack 1992) to avoid inconsistent, redundant or inefficient constraints. We have space to discuss only the mapping of subset constraints and mandatory roles. This will give the general idea. A detailed treatment of mapping all the constraints is given in Ritson & Halpin (1992).

Role-sequences are sequences of roles and are denoted $rs_i = r_{i1}, \ldots, r_{in}$. If a constraint contains two or more role-sequences, all of the role-sequences are the same length, and all roles in the same positions in role-sequences (r_{ak} , r_{bk}) are played by object-types belonging to the same subtype-tree. All role-sequences are assumed to have role-graphs defined for them.

Subset constraints:

A subset constraint is made up of two role-sequences rs_{subset} and $rs_{superset}$. Create one view for each role-sequence (role-graph) V_{subset} and $V_{superset}$. If both views are made up of single queries which have the same selectors and are over the same tables, they can be combined into the one query using the construction:

V_{subset} = select selectors from tablename where where-clause$_{subset}$
$V_{superset}$ = select selectors from tablename where where-clause$_{superset}$

Adding the following check-clause to tablename will prevent invalid data from being entered into the table.

not where-clause$_{subset}$ or where-clause$_{superset}$

For example, consider the relation Employee (emp#, sex, [car], [licence][1]): [1] **non-null only if** car **non-null.** On the conceptual schema there is a subset constraint that a licence is recorded only if a car is. In the relational schema, violations of this constraint can be detected by the query: **select * from** Employee **where** licence **is not null and** car **is null.** Or better still, violations can be prevented from entering the database by adding the following check clause to the create table statement for Employee: **check** (licence **is null or** car **is not null**). This example can be related to the above cases by using double negation.

In other cases there are three basic ways of generating the relational constraint: V_{subset} subset of $V_{superset}$. If the superset is a primary key or unique then a foreign key clause can be declared (so long as this ANSI SQL feature is supported). With our bank example for instance, the pair-subset constraint BackedServices[bank,group] ⊆ Services[bank,group] may be declared by adding the following clause to the definition of BackedServices: **foreign key** (bank, group) **references** Services, assuming the primary key for Services has been declared. The superset columns may be explicitly included after the target table name, and must be if a unique clause rather than a primary key clause has been declared for them.

In other cases, we can generate procedural code to check for existing violations or generate triggers to prevent violations. The following code will detect any existing violation, using *col1..n* to denote the *n* selectors being compared ($n \geq 1$):

```
select * from subset-table X
where  not exists
  (select * from superset-table Y
   where  Y.col1 = X.col1 and ... Y.col_n = X.col_n)
```

For example, violations of the constraint CeasedSavings.customer ⊆ CurrentSavings.customer could be detected in this way (since the target is not a key a references clause cannot be used). In this trivial case $n = 1$, and no correlation variables are actually needed since the source and target tables differ.

Alternatively, triggers can be written to prevent the violation. With our current example, triggers would be fired on insert or update of CeasedSavings as well as on delete and update of CurrentSavings. See Ritson & Halpin (1992) for further details.

Disjunctive and Non-functional Mandatory Role Constraints:

A role is functional if and only if it has a simple uniqueness constraint. Functional mandatory roles are trivially enforced by not null columns, and subset constraints from other tables where needed. Other case are harder. For the purpose of this paper, they can be viewed as a set of N single role role-sequences -- rs_1 , ... , rs_n where $rs_i = r_k$ and r_k is played by the object-type OT. The first step is to find a view V_{whole} which contains all of the instances of OT.

If OT plays a role R which has a non-disjunctive mandatory role constraint on it, V_{whole} is the view created from the role-sequence containing only R (preference is given to attribute roles). Otherwise, OTroles = { r_1 , ... , r_m } = the set of roles played by OT and subtypes of OT minus the set of roles involved in the disjunctive mandatory role constraint being mapped. V_1 , ... , V_m are the views created from the N single role role-sequences $rs_1 = r_1$, ... , $rs_m = r_m$. V_{whole} is created as the union of V_1 , ... , V_m .

Exclude from V_{whole} any roles which have implied or explicit subset or equality constraints to any roles in V_{cons} . An algorithm for finding implied subset constraints is presented in Halpin & McCormack (1992). Now create V_{cons} as the union of the views created from all of the role-sequences in the disjunctive mandatory role constraint. Optimize V_{whole} and V_{cons} using the same procedures for optimising views and unions discussed earlier.

If V_{whole} and V_{cons} have the same selectors and are over the same table (i.e. V_{whole} = select selectors from tablename where where-clause$_{whole}$, and V_{cons} = select selectors from tablename where where-clause$_{cons}$) the query:

```
select selectors from tablename
where   (where-clause_whole and not where-clause_cons)
```

returns all rows in the table which violate the constraint. Adding the following check-clause to tablename prevents invalid data from being entered into the table:

```
(not where-clause_whole or where-clause_cons)
```

In other cases, code can be generated to detect or prevent violations of the condition: V_{whole} subset of V_{cons} (cf. subset discussion earlier). As a nasty example, consider the disjunctive mandatory role constraint over r2, r3, r4 in Figure 1 (each Bank is either a merchant bank or has a savings branch or services a group). Each role has a view created for it:

```
View_r2 = create View_r2 (Bank) as
          select Bank from CurrentSavings
          union
          select Bank from CeasedSavings

View_r3 = create View_r3  (Bank) as select Bank from MerchantBank

View_r4 = create View_r4 as  (Bank) select Bank from Services.Bank
```

So V_{cons} becomes:

```
create View_cons (Bank) as
    select Bank from CurrentSavings union select Bank from CeasedSavings
```

```
union
select Bank from MerchantBank union select Bank from Services.Bank
```

and V_{whole} becomes:

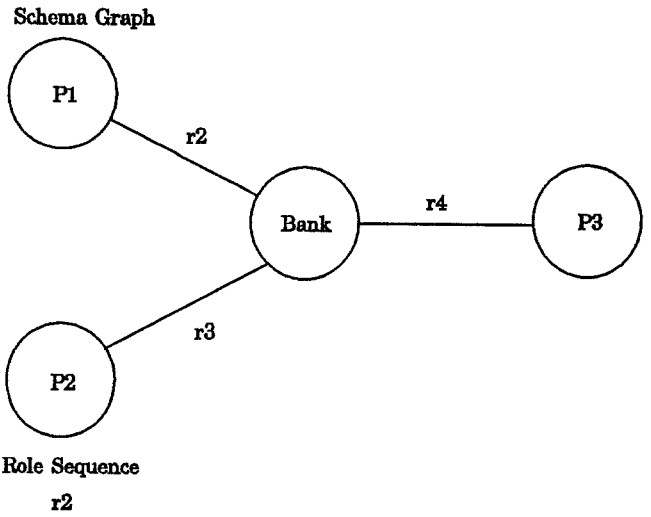
```
create View_whole (Bank) as  select Bank from UnderwrittenBank
```

So the final constraint is: V_{whole} subset V_{cons}. The SQL implementation of this is similar to the subset constraint in the previous example.

7 Mapping "Schema Expressions" to Relational Expressions

Role-graphs are not restricted to capturing constraints. They can be used in other areas of conceptual modelling and mapping as well. One area of particular interest is that of mapping rules and queries expressed conceptually to relational expressions. For related work using ER see Hohenstein & Engels (1991) and Markowitz & Shoshani (1990). While we have no formal results in this area yet, it is evident that role-graphs may be of significant use here. Consider the following query based on the schema in Figure 1: "Which banks are both merchant and savings banks and also service the primary industries group?"

This can be expressed as the following role-graph (r5 is restricted to 'primary industries').



This maps to the SQL query:

```
create view P1 (bank) as
    select bank from CurrentSavings
    union
    select bank from CeasedSavings
```

```
select P1.bank from P1, MerchantBank, Services
where   P1.bank = MerchantBank.bank
  and   MerchantBank.bank = Services.bank
  and   Services.group = 'primary industries'
```

8 Conclusion

This paper presented algorithms for mapping an ORM conceptual schema to a relational database schema, including important constraints that are typically ignored by other mapping algorithms. Role-graphs were introduced as an intermediate data structure to enable efficient automation of the mapping without loss of generality. The algorithms have been implemented in the InfoViews CASE tool, and benchmarks have been encouraging. A conceptual schema with over 250 object types and several hundred constraints was completely mapped, including code generation, in 40 seconds on a 33 MHz 386SX PC running under Microsoft Windows. While the main features of the approach have been illustrated in the paper, various cases have been excluded from the discussion (e.g. subtyping, 1:1 refinements, other constraints). The textual version of the FORML language enables constraints to be specified which cannot be expressed in the graphic notation, and in addition allows subtype definitions and derivation rules to be specified. Research is currently under way to extend the expressive power of this language and to provide automated support for mapping these additional textual specifications.

References

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

Codd, E.F. 1990, *The Relational Model for Database Management: Version 2*, Addison-Wesley, Reading MA.

Control Data 1982, *IAST: Information Analysis Support Tools*, Reference Manual (Control Data Publication no. 60484610).

Czejdo, B., Elmasri, R., Rusinkiewicz, M. & Embley, D.W. 1990, 'A Graphical Data Manipulation Language for an Extended Entity-Relationship Model', *IEEE Computer*, March 1990, pp. 26-37.

De Troyer, O., Meersman, R. & Verlinden, P. 1988, 'RIDL* on the CRIS Case: a Workbench for NIAM', *Computerized Assistance during the Information Systems Life Cycle: Proc. CRIS88*, eds T.W.Olle, A.A. Verrijn-Stuart & L. Bhabuta, North-Holland, Amsterdam.

De Troyer, O. 1989, 'RIDL*: A Tool for the Computer-Assisted Engineering of Large Databases in the Presence of Integrity Constraints', *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, Oregon.

De Troyer, O. 1991, 'The OO-Binary Relationship Model: a truly object-oriented conceptual model', *Advanced Information Systems Engineering: Proc. CAiSE-91*, Springer-Verlag Lecture Notes in Computer Science, no. 498, Trondheim.

Elmasri, R. & Navathe, S.B. 1989, *Fundamentals of Database Systems*, Benjamin/Cummings, Redwood City CA.

Halpin, T.A. 1989a, 'A Logical Analysis of Information Systems: static aspects of the

data-oriented perspective', PhD thesis, University of Queensland.

Halpin, T.A. 1989b, 'Contextual Equivalence of Conceptual Schemas', *Proc. Advanced Database Systems Symposium*, Info. Processing Soc. of Japan, Kyoto, pp. 47-54.

Halpin, T.A. 1991a, 'A fact-oriented approach to schema transformation', *Proc. MFDBS-91*, Springer Verlag Lec. Notes in Computer Science, no. 495, Rostock.

Halpin, T.A. 1991b, 'WISE: a Workbench for Information System Engineering', *Proc. 2nd Workshop on Next Generation of CASE Tools*, Trondheim (May 1991), reprinted in *Next Generation CASE Tools*, eds K. Lyytinen & V.-P. Tahvanainen, IOS Press, Amsterdam 1992.

Halpin, T.A. 1992, 'Fact-oriented schema optimization', *Proc. CISMOD-92*, Bangalore, India, July 1992.

Halpin, T.A., Harding, J. & Oh, C-H. 1992, 'Automated support for Subtyping', *Proc. 3rd Workshop on Next Generation of CASE Tools*, eds B. Theodoulidis & A. Sutcliffe, UMIST, UK.

Halpin, T.A. & McCormack, J. 1992, 'Automated Validation of Conceptual Schema Constraints', *Advanced Inf. Systems Engineering: Proc. CAiSE'92*, ed. P. Loucopoulos, Springer Verlag Lec. Notes in Computer Science, no. 593, pp. 445-62.

Halpin, T.A. & Orlowska, M.E. 1991, 'Fact-Oriented Modelling for Data Analysis', *Journal of Information Systems*, vol. 2, no. 2, Blackwell Scientific, Oxford.

Halpin, T.A. & Ritson, P.R. 1992, 'Fact-Oriented Modelling and Null Values', *Research and Practical Issues in Databases: Proc. 3rd Australian Database Conf.*, eds B. Srinivasan & J. Zeleznikov, World Scientific, Singapore.

Hohenstein, U. 1990, 'Automatic transformation of an Entity-relationship query language into SQL', *Entity-Relationship Approach to database design and querying (Proc. 8th ER conf.)*, ed. F.H. Lochovsky, Elsevier Science Pub., Amsterdam.

Hohenstein, U. & Engels, G. 1991, 'Formal semantics of an entity-relationship-based query language', *Entity-Relationship Approach: the core of conceptual modelling (Proc. 9th ER conf.)*, ed. H. Kangassalo, Elsevier Science Pub., Amsterdam.

Intellibase, 1990, *RIDL-M User's Guide*, Intellibase N.V., Belgium.

ISO 1982, *Concepts and Terminology for the Conceptual Schema and the Information Base*, ed. J.J. van Griethuysen, ISO TC97/SC5/WG3, Eindhoven.

Mark, L., 1987, 'The Binary Relationship Model - 10th Anniversary', *Tech. Report CS-TR-1933*, Uni. of Maryland.

Markowitz, V.M. 1990, 'Referential integrity revisited: an object-oriented perspective', *Proc. 16th VLDB Conf.*, Brisbane.

Markowitz, V.M. & Shoshani, A. 1990, 'Abbreviated query interpretation in extended Entity-Relationship oriented databases', *Entity-Relationship Approach to database design and querying*, ed. F.H. Lochovsky, Elsevier Science Pub., Amsterdam.

Nienhuys-Cheng, S. 1990, 'Classification and Syntax of Constraints in Binary Semantical Networks', *Inform. Systems*, vol. 15, no. 5, pp. 497-513.

Nijssen, G.M. & Halpin, T.A. 1989, *Conceptual Schema and Relational Database Design*, Prentice Hall, Sydney.

Olle, T.W., Hagelstein, J., Macdonald, I.G., Rolland, C., Sol, H.G., Van Assche,

F.J.M. & Verrijn-Stuart, A.A. 1991, *Information Systems Methodologies - A Framework for Understanding, 2nd edn.*, Addison-Wesley, Wokingham, England.

Ovum 1992, *Ovum Evaluates: CASE Products*, Ovum Ltd, London.

Papastamatiou, G. 1992, 'Transaction semantics in a conceptual rule language', *Proc. 3rd workshop on Next Generation CASE Tools*, eds B. Theodoulidis & A. Sutcliffe, UMIST, UK.

Reiner, D. 1992, 'Database Design Tools', in Batini, Ceri & Navathe (op. cit.), Ch. 15.

Ritson, P.R. & Halpin, T.A. 1992, 'Mapping conceptual constraints to a relational schema', *Tech. Report 223*, Dept of Computer Science, University of Queensland.

Ritson, P.R. & Halpin, T.A. 1993, 'Mapping One-to-One Predicates to a Relational Schema', *Proc. 4th Australian Database Conf.*, (Brisbane, February 1993), World-Scientific, Singapore.

Rochfeld, A., Morejon, J. & Negros, P. 1991, 'Inter-Relationship Links in E-R Model',

Shoval, P. & Even-Chaime, M. 1987, 'ADDS: A system for automatic database schema design based on the binary-relationship model', *Data and Knowledge Engineering*, vol. 2, pp. 123-44.

Shoval, P., Gudes, E. & Goldstein, M. 1988, 'GISD: A Graphical Interactive System for Conceptual Database Design', *Inform. Systems*, vol. 13, no. 1, pp. 81-95.