Query Modification in Object-Oriented Database Federations

Mark W.W. Vermeer and Peter M.G. Apers
{vermeer,apers}@cs.utwente.nl

Centre for Telematics and Information Technology
University of Twente
Enschede, The Netherlands

Abstract

We discuss the modification of queries against an integrated view in a federation of object-oriented databases. We present a generalisation of existing algorithms for simple global query processing that works for arbitrarily defined integration classes. We then extend this algorithm to deal with object-oriented features such as queries involving path expressions and nesting. We show how properties of the OO-style of modelling relationships through object references can be exploited to reduce the number of subqueries necessary to evaluate such queries.

1 Introduction

An important issue in database interoperability is the processing of queries against an integrated view in a federation of pre-existing databases. Although a number of different approaches to database interoperation – often divided into so-called loosely-coupled and tightly-coupled approaches [16] – have been proposed, many of them share the characteristic of somehow defining an integrated view of data residing in multiple component databases. As a consequence, queries can be formulated against such a view, and these queries must be answered by transforming them into queries that can be answered by the underlying component databases. The query processing issues that then arise, are largely independent of whether such a view defines a rigid global schema for the entire federation [16], or just a personal view of remote data imported into a local database [8]. In this paper, we refer to processing queries against any kind of integrated view as *global query processing.*

While global query processing has been dealt with for the case of rather elementary queries [3,4], the current popularity of the object-oriented model as a canonical data model in such federations [15] introduces the possibility of formulating queries involving path expressions and nesting against an integrated view. Processing such queries has not received much attention in existing research; it is discussed in this paper. In the process, we generalise existing work on processing simple queries by defining a global query processing algorithm that is independent of the definition of the classes in the integrated view.

1.1 Issues in global query processing

The following issues in global query processing in database federations may be distinguished [13].

- *Query modification*, i.e. the transformation of a query against integrated classes to a query against local and/or remote classes. This is the topic of this paper.
- *Query decomposition*, i.e. the transformation of a modified query against both local and remote classes into several queries each addressing a single component database.
- *Local query processing*, i.e. the transformation of a query addressing the global view of a single component database into a query that can be processed by the local database.
- *Query optimisation*, i.e. the problem of executing a given query as efficiently as possible.

1.2 Existing work

Global query processing is an obvious research goal in the context of database interoperation. Hence its principles were already investigated in the context of early systems [3,4,5]. More recently, these principles

0-8186-7946-8/97 $10.00

have been extended with regard to aspects such as query modification [13], optimisation [11], and object-orientation [9].

None of these approaches has considered the global query modification in its full generality. For example, [3] does not deal with possible value inconsistencies; [4,5] do not consider the case of locally exclusive attributes (they consider integrated classes formed using generalisation, which do not contain attributes that are exclusive to a specific local database). Many approaches, e.g. [9,13,14], treat global query processing in the context of a specific set of integrated class constructors; global query modification is then the dual problem of the materialisation of the virtual integrated classes thus obtained.

Moreover, the specific features of object orientation w.r.t. query processing have not been treated in any depth. In particular, the modification of global queries involving path expressions and nested queries on set-valued attributes has not received much attention.

1.3 Overview

The contribution of this paper is twofold. First and foremost, we try to *complement* these existing works in that we consider query modification in an object-oriented environment, allowing the expression of queries against complex object structures that involve path expressions and nested queries on set-valued attributes. Moreover, we *generalise* existing work in that we consider queries against *arbitrarily defined* integrated classes, that may: (1) contain objects from different component databases, with possible overlap in the real-world objects they describe; (2) have attributes defined in different component databases, where some attributes may be defined in several components; and (3) use decision functions to settle possible value conflicts.

We focus on *query modification*, as the kind of queries we address present new challenges to this phase in particular. The optimisation of the modified queries we obtain, for example, does not differ significantly from those discussed in [4,11]. Our modified queries are themselves 'optimised' in the sense that subqueries which are known to yield empty results are eliminated, however. That is, we show how the specification of global complex objects can be exploited to reduce the number of subqueries necessary to evaluate the global query.

The remainder of this paper is organised as follows. As a context for discussion, we review our instance-based approach to database interoperation in Section 2. To illustrate our claim that this approach allows us to deal with global query processing in a general manner, we develop some vocabulary to be used in our treatment of global query modification, and show how integrated classes as defined by several existing integration methodologies can be described using this vocabulary. Section 3 then presents a general query modification algorithm for simple queries on arbitrarily defined integrated classes. In Section 4, we extend this algorithm to deal with queries involving path expressions and nested queries. Section 5 presents our conclusions.

2 An integrated view of component databases

In this section, we summarise our *instance-based* approach to database interoperation [18], which will form the context for our discussion of global query modification. This approach is characterised by the adaptation of existing schema integration techniques [2] to be applicable at the instance level of database interoperation, thus avoiding the explicit mapping of the different classifications used by the different component databases to describe a similar application domain.

We furthermore introduce an example (Section 2.1) that will be used throughout this paper, and introduce some definitions (Section 2.3) that will be used in upcoming sections. Although these definitions are in terms of the approach described in this section, they can be used to describe any kind of integration classes formed in other approaches. Indeed, these definitions are introduced to abstract from the specific kind of integration classes defined by particular interoperation approaches.

2.1 Example

We here present an example containing two rather similar schemata, which will nevertheless be illustrative enough to discuss global query modification issues. Consider the two databases whose schemata are represented graphically in Figures 1 and 2.

Database `WELLS` is kept by a researcher who keeps track of oil exploration and production. Database `FIELDS` is used by a government institute that registers ownership and use of oil fields. As can be seen from their schemata, the databases have largely overlapping application domains, but focus on slightly different aspects. For example, a field can be concessioned to several companies, as modelled by database `FIELDS`, but a well is owned by a single company, as modelled by database `WELLS`.

We assume that the `WELLS` researcher wishes to define a virtual integrated view of his local database ex-

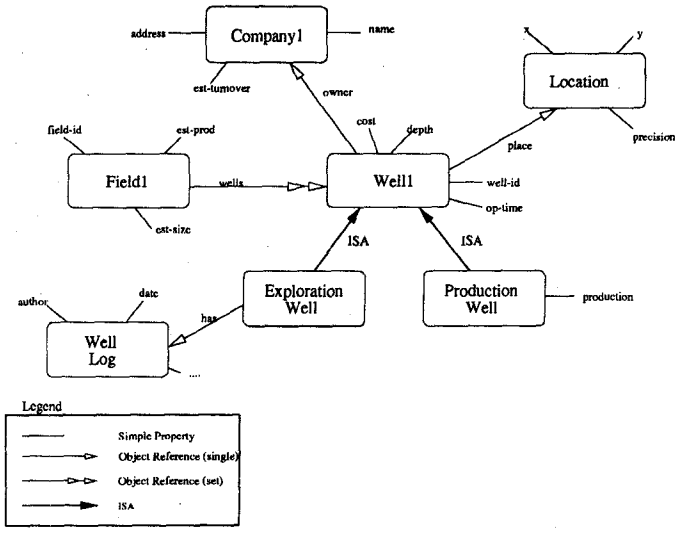

Figure 1. Schema of the WELLS database

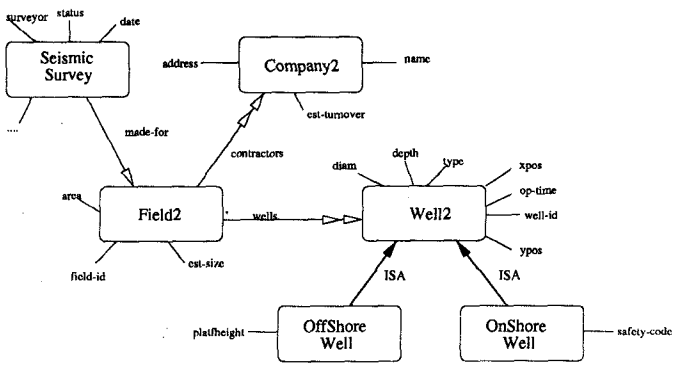

Figure 2. Schema of the FIELDS database

tended with data imported from FIELDS. Hence we will refer to WELLS as the 'local database', and to FIELDS as the 'remote database'.

2.2 Object comparison rules and property equivalences

In [18] we outlined an instance-based approach to database interoperation. As argued in length there, we consider objects rather than classes to be an appropriate basic unit of integration. In short, the motivation for our approach is the argument that in absence of a common semantical context, it is more feasible for disparate sources to agree on relationships among the specific real-world objects that they describe, than to agree on the semantics of possible *classifications* for those objects. We showed that interoperability can be based on the definition of relationships between *objects*, by specifying conditions under which objects from different classes are related in a certain way.

Our approach requires a designer to specify conditions under which a certain relationship ρ between a remote object O' and a local object O or class C holds[1]. The relationships we distinguish are:

- **Equality**. O and O' represent the same real world object [12]. This is represented as $Eq(O', O)$.

  For example, some wells may be known in both FIELDS and WELLS.

- **Strict similarity**. O' would locally be classified under C. This is represented as $Sim(O', C)$.

  For example, some wells in FIELDS may be production wells, and would be classified as ProductionWell if they were known in WELLS.

- **Approximate similarity**. Locally $C \cup \{O'\}$ can be regarded as a more general virtual class C''. This is represented as $Sim(O', C, C'')$.

  For example, well logs and interpreted seismic surveys both describe some kind of interpretation of subsurface data.

- **Descriptivity**. Locally O' is considered a set of values S describing an object O'' which is identical to a local object O or similar to a local class C. This is represented as $Eq(O', O.S)$ or $Sim(O', C.S)$.

  For example, Location objects may refer to locations of well objects in FIELDS.

We require the specification of *object comparison rules* of the form $\rho \leftarrow \Psi$, where ρ is any of the relationships listed above, and Ψ is a conjunction of first-order logic predicates, which might involve additional information such as correspondence tables etc.

Moreover, *property equivalence assertions* must be formulated, specifying to what extent the descriptions provided by DB and DB' overlap. These assertions are of the form $propeq(C.p, C'.p', cf, cf', df)$, where:

- p, p' are basic or derived local and remote properties, respectively,
- cf, cf' are *conversion functions* mapping the domains of p and p' to a common domain D, and
- $df : D \times D \rightarrow D$ is a *decision function* which determines a global value for the property given possible value discrepancies between the local and the remote database. We require that for each

[1] In the remainder of this paper, we use the conventions for symbols s to refer to the local database, s' to refer to the remote database, and $\hat{s}$ to refer to the integrated view of these databases.

decision function df, $\forall a \in D | df(a,a) = a$. In our view, functions such as *sum* used e.g. in [5] define derived global properties rather than determining values for equivalent local and remote properties.

Example For our example databases, we list some sample object comparison rules. Note that the example is used for illustration purposes; it is not intended to demonstrate our integration methodology in full. In particular, the schemata have been kept rather similar for ease of presentation. We also list some example property equivalences, omitting obvious ones. We use predefined conversion functions such as *id*, the identity function, and decision functions such as *trust*, which assigns a specific database as the primary source for a property's value. Note that we abstract from the entity identification problem [12] by assuming that both fields and wells are identified by a generally known id, which is used as a key in both databases.

$Eq(O$:`Field1`, O':`Field2`$) \leftarrow O$.field-id=O'.field-id
$Sim(O'$:`Field2, Field1`$)$
$Eq(O$:`Well1`, O':`Well2`$) \leftarrow O$.well-id=O'.well-id
$Sim(O'$:`Well2, ExplorationWell`$) \leftarrow O'$.type='exploration'
$Sim(O'$:`Well2, ProductionWell`$) \leftarrow O'$.type='production'
$Eq(O$:`Company1`, O':`Company2`$) \leftarrow O$.name=O'.name
$Sim(O'$:`Company2, Company1`$)$
$Eq(O$: `Location`, O' : `Well2`.{xpos,ypos}$)$
$\leftarrow O$.x=O'.xpos $\wedge O$.y=O'.ypos
$Sim(O'$:`SeismicSurvey,WellLog, INTERPRETATION`$)$
$\leftarrow O'$.status='interpreted'

$propeq$(`Field1`.wells, `Field2`.wells, $id, id, union$)
$propeq$(`Field1`.est-size, `Field2`.est-size,
$id, M3ToBarrels, trust$(`Field2`))
$propeq$(`Company1`.est-turnov, `Company2`.est-turnov,
id, id, avg)
$propeq$(`WellLog`.author, `SeismicSurvey`.surveyor, id, id, any)
$propeq$(`Well1`.depth, `Well2`.depth, id, id, avg)
$propeq$(`Well1`.op-time, `Well2`.op-time,
$id, YearsToMonths, trust$(`Well2`))

2.3 Definitions

We now develop some terminology to be used in our discussion of global query modification. Consider a local class C and a remote class C' for which rules of the form

$$Eq(O : C, O' : C') \leftarrow \Phi(O, O')$$
$$Sim(O' : C', C) \leftarrow \Psi(O')$$
$$Sim(O' : C', C, C_{sup}) \leftarrow \Omega(O')$$

have been defined. Descriptivity rules and the associated conformation are discussed later. For simplicity, we here assume that object comparison rules have been defined between C and a single remote class C' only. The case where such rules are defined for multiple remote classes is a straightforward extension.

Attribute types In global query processing, one must distinguish among three disjoint types of attributes:

- *Locally exclusive attributes* $A_{LE} = \{a |\ a \in attribs(C) \wedge \nexists a' \in attribs(C') : propeq(a, a')\}$. These attributes occur in the local database only. An example is the 'est-prod' attribute of `Field1`.
- *Remotely exclusive attributes* $A_{RE} = \{a' |\ a' \in attribs(C') \wedge \nexists a \in attribs(C) : propeq(a, a')\}$. These attributes occur in the remote database only. An example is the 'area' attribute of `Field2`.
- *Overlapping attributes* $A_O = \{a |\ a \in attribs(C) \wedge \exists a' \in attribs(C') : propeq(a, a')\}$. These attributes occur both in the local and the remote database. An example is the 'wells' referential attribute of `Field1` and `Field2`.

Extension subsets We also define the following extension subsets:

- *Local objects* $C_L = \{O | O \in C\}$.
- *Locally exclusive objects* $C_{LE} = \{O | O \in C \wedge \nexists O' \in C' : \Phi(O, O')\}$, the set of objects that appear only locally. For example, some wells may be known in `WELLS`, but not in `FIELDS`.
- *Remote objects* $C_R = \{O' | O' \in C'\}$.
- *Remotely exclusive objects* $C_{RE} = \{O' | O' \in C' \wedge \nexists O \in C : \Phi(O, O')\}$. For example, some fields may be known in `FIELDS` but not in `WELLS`.
- *Strictly similar remote objects* $C_{SS} = \{O' | O' \in C' \wedge \Psi(O')\}$. For example, all `Well2`-objects with type='production' are strictly similar to the local class `ProductionWell`.
- *Approximately similar remote objects* $C_{AS} = \{O' | O' \in C' \wedge \Omega(O')\}$. For example, all `SeismicSurvey` objects with status='interpreted' are approximately similar to the local class `WellLog` (assuming that well logs are interpreted by definition).
- *Overlapping objects* $C_O = \{O | O \in C \wedge \exists O' \in C' | \Phi(O, O')\}$ the set of objects that appear both locally and remotely. For example, some wells may be known in both `WELLS` and `FIELDS`.

Note that some, but not all, of these subsets are disjoint.

2.4 The integrated view

As a result of a specification as defined above, an integrated or global view of the local and the remote database can be constructed. This construction is a two-step process of *conformation* and *merging* analogous to the steps distinguished for schema integration in [2].

2.4.1 Conformation

In the conformation step, the local and remote database are brought into a common semantical context, so that they can be merged. This involves the settling of object-value conflicts resulting from descriptivity relations between objects, by creating virtual objects from values and/or casting objects into property values describing other objects. In our example, the description of a location as a set of simple values describing a well or as a separate object must be conformed. For example virtual `VirtLocation`-objects may be created from the values of `Well2`.{xpos,ypos}.

Moreover, equivalent local and remote properties are turned into conforming properties by assigning them identical names and converting them to identical domains. Examples include the renaming of 'surveyor' to 'author' and conversion of `FIELD` field size estimations from cubic metres to barrels.

Query processing issues related to conformation arise in local query processing only. This is not our focus here; hence for the moment we do not consider structural conflicts, naming conflicts, or domain differences between the component databases.

2.4.2 Merging

In the merging step, *integrated classes* are formed. This involves the definition of both the attribute sets and the extensions of classes that appear in the integrated view. This can be done using the attribute types and extension subsets that we defined above. Our query modification algorithm, however, is independent of the exact definition of these classes; it uses only the definitions of Section 2.3. Examples of definitions of global classes include the following:

- The extension of local classes C is expanded with similar objects from C', as in [6]. We call such expanded classes *enriched local classes*, denoted as $\dot{C}$. For example, `Well2` objects with type 'production' might be added to the local `ProductionWell` class.

  That is, $\dot{C} = C_{LE} \cup C_O \cup C_{SS}$, and $A_{\dot{C}} = A_{LE} \cup A_O$.

- For each pair (C, C') of classes for which equality rules have been defined, a common subclass $\dot{C}_{sub}$ may be created, holding their common instances, as in [9]. For example, a class `OffShoreProductionWells` might be created.

  That is, $\dot{C}_{sub} = C_O$, and $A_{\dot{C}_{sub}} = A_{LE} \cup A_O \cup A_{RE}$.

- For each pair (C, C') of classes for which approximate similarity rules have been defined, a common superclass $\dot{C}_{sup}$ may be created, containing their common attributes, analogous to [5]. An example is the creation of the virtual class `INTERPRETATION` holding both well logs and interpreted seismic surveys.

  That is, $\dot{C}_{sup} = C_{LE} \cup C_O \cup C_{SS} \cup C_{AS}$, and $A_{\dot{C}_{sup}} = A_O$.

Note that other definitions would be possible, e.g. $\dot{C}_{sub} = C_O \cup C_{SS}$, or $A_{\dot{C}} = A_{LE} \cup A_O \cup A_{RE}$. Indeed, different approaches to database interoperation tend to define different kinds of integrated classes. In the next section, we present a general query modification algorithm, adapted from the ones in [3,4,13], that is independent of integrated class definitions, as long as they can be described using the definitions of Section 2.3. It will be used as a basis for our discussion of queries on complex objects.

3 Simple queries

Equipped with a proper terminology, we now turn to the modification of simple queries against integrated classes. A simple query Q is of the form[2]

collect Π
for x **in** $\dot{C}_Q$
iff Ξ

where we assume that $\dot{C}_Q$ is an integrated class, Π is of the form $f(x.a_1, x.a_2, \ldots, x.a_n)$, where none of the attributes a_i is referential, and Ξ is a conjunctive condition.

Query modification principles The generalised query modification algorithm for simple queries presented in this section is based on a combination of principles found in existing literature.

Attributes appearing in either Ξ or Π must be evaluated by consulting the local and/or remote database. The evaluation of an *overlapping* attribute a is supported by both the local and the remote database. However, both component databases provide a value

[2] In this paper we use the TM [1] syntax for queries. This syntax is rather self-explanatory, but will be clarified where needed.

for a global object $\dot{O}$ only if the component's extension contains a representation of $\dot{O}$. If both the local and the remote database contain such a representation, possible *value inconsistencies* must be resolved using the decision function. Hence, analogous to [4,13], the evaluation of overlapping attributes in either Ξ or Π induces a distribution of Q over the disjoint subsets C_{LE}, C_{RE} and C_O.

On the other hand, the evaluation of locally *exclusive* attributes is supported by the local database only; analogously, remotely exclusive attributes are supported by the remote database only. As observed in [3], queries where Ξ contains locally exclusive attributes do not have answers in C_{RE}. In [3], it is assumed that value inconsistencies do not occur; then if $\Pi \cap A_{RE} = \emptyset$, the evaluation of such queries does not involve the remote database at all. In general however, for objects in C_O that satisfy Ξ according to the local database, the evaluation of overlapping (or remotely exclusive) attributes appearing in the **collect** part of Q still involves the remote database. Analogously, queries where Ξ contains remotely exclusive attributes do not have answers in C_{LE}.

Query modification algorithm These principles lead to the formulation of the following generalised algorithm for modifying simple queries on enriched local classes, virtual superclasses and virtual subclasses.

1. First, we define a procedure that, given a global query Q on an integrated class $\dot{C}_Q$ which has a definition as in Section 2.4, finds the set of relevant extensions against which Q is to be decomposed. Relevant extensions are determined by: (1) the query characteristics (in particular, the type of attributes occurring in Ξ and Π); and (2) the definition of $\dot{C}_Q$. The procedure `RelevantExtensions` is shown in Figure 3.

2. We then construct a subquery for each relevant extension. This involves expressing the definition of the extensions as given in Section 2.3 in terms of the query language; this is quite straightforward, although for C_O it requires a nested **collect** in TM(see the example). References to attribute values are modified as follows:

   - In the case of C_{LE}, occurrences of remotely exclusive attributes in Π are replaced with **null**.
   - In the case of C_{RE}, occurrences locally exclusive attributes in Π are replaced with **null**.
   - In the case of C_O, occurrences of an overlapping attribute a in either Ξ or Π are replaced with the decision function specified for a, applied to the local and the remote value of a.
   - In the case of C_L or C_R, attribute references remain as is (note that we have abstracted from local query modification here).

3. The result of Q is the union of the results of the subqueries thus obtained.

```
RelevantExtensions
in:  Q = (Π, C_Q, Ξ)
out: RelExt

%Define overlap of C_Q with possibly relevant extensions
Q_L = C_L ∩ C_Q; Q_R = C_R ∩ C_Q;
Q_LE = C_LE ∩ C_Q; Q_RE = C_RE ∩ C_Q;
Q_O = C_O ∩ C_Q;
%Examine query characteristics to define relevant extensions
If   (attrs(Ξ) ∪ attrs(Π)) ∩ A_O ≠ ∅
Then
     RelExt = {Q_LE, Q_O, Q_RE};
     If   attrs(Ξ) ∩ A_LE ≠ ∅
     Then RelExt = RelExt \ {Q_RE};
     If   attrs(Ξ) ∩ A_RE ≠ ∅
     Then RelExt = RelExt \ {Q_LE};
Else
     If   attrs(Ξ) ⊂ A_LE
     Then RelExt = {Q_L}
     Else If   attrs(Ξ) ⊂ A_RE
          Then RelExt = {Q_R}
          Else RelExt = {Q_O}
```

Figure 3. Procedure `RelevantExtensions`

Example As a representative example, we consider the modification of the query Q:

collect ⟨ x.production, x.platfheight⟩
for x **in** `OffShoreProductionWell`
iff x.depth> 1000

where `OffShoreProductionWell` is an integrated subclass of `ProductionWell` and `OffShoreWell`, 'production' is a locally exclusive attribute, 'platfheight' is remotely exclusive, and 'depth' an overlapping attribute with decision function *avg*. We here assume the definition of the extension of integrated subclasses is $C_{sub} = C_O \cup C_{SS}$.

Note that Ξ contains overlapping attributes; hence `RelevantExtensions` initialises *RelExt* to $\{Q_{LE}, Q_O, Q_{RE}\}$. Since Ξ contains neither locally nor remotely exclusive attributes, the following tests do not lead to a reduction of this set. Now $Q_{LE} =$

$C_Q \cap C_{LE} = (C_O \cup C_{SS}) \cap C_{LE} = \emptyset$, and $Q_{RE} = C_Q \cap C_{RE} = (C_O \cup C_{SS}) \cap C_{RE} = C_{SS} \cap C_{RE}$, and $Q_O = C_Q \cap C_O = (C_O \cup C_{SS}) \cap C_O = C_O$. Hence the relevant subextensions are $(C_{SS} \cap C_{RE})$ and C_O.

Q is then transformed into the following query [3] referencing only `ProductionWell` and `OffShoreWell`. Note how the definition of C_O (the nested part) and $C_{RE} \cap C_{SS}$ is represented in the query.

```
unnest
  collect
      collect ⟨ x.production, y.platfheight ⟩
      for y in OffShoreWell
      iff x.well-id=y.well-id
      and avg(x.depth,y.depth)> 1000      % CO part
  for x in ProductionWell
union
  collect ⟨ null, y.platfheight ⟩
  for y in OffShoreWell
  iff y.type='production' and y.depth> 1000
    and not exists x in ProductionWell |
      x.well-id=y.well-id          % CRE part
```

Compare this to the query

```
collect ⟨ x.platfheight⟩
for x in OffShoreProductionWell
iff x.production> 1000
```

Here Ξ contains only locally exclusive attributes; Π contains remotely exclusive attributes. Hence $RelExt = \{Q_L\}$, where $Q_L = C_Q \cap C_L = (C_O \cup C_{SS}) \cap C_L = C_O$, and the resulting query is:

```
unnest
  collect
      collect ⟨ y.platfheight ⟩
      for y in OffShoreWell
      iff x.well-id=y.well-id
      and x.production> 1000          % CO part
  for x in ProductionWell
```

4 Complex queries

So far, we have discussed the modification of a type of queries that may arise in both the relational and object-oriented data model. We now consider more complex queries against integrated classes that arise in the object-oriented model only, due to the feature of *referential attributes* of this model. A referential attribute is an attribute whose domain is a class. We distinguish between *single-valued* and *set-valued* referential attributes.

[3] This query involves nesting. In TM, **collect** expressions can be nested; a preceding **unnest** causes the result to be a simple set.

4.1 Path expressions

When considering single-valued referential attributes, we allow attribute expressions such as $x.d.a$ to occur in the **collect** and **iff** part of the query, where the referential property $x.d$ has an integrated class D_Q as its domain.

Example Here's an example query Q containing a path expression:

```
collect ⟨ x.cost, x.owner.est-turnov ⟩
for x in OffShoreProductionWell
iff x.owner.est-turnov> 1000 and x.diam < 1
```

Issues in modifying path queries Compared to the simple queries we have considered so far, such queries introduce additional problems:

(1) Object comparison rules may have been defined on D_Q as well. In the case of our example query, the domain class `Company` of the attribute 'owner' is an integrated class constructed from `Company1` and `Company2` due to the identity and similarity rules defined on them.

(2) Like any other attribute, referential properties may be overlapping, and hence a decision function must be definable on them. For the single-valued case, we identify the decision functions $trust(DB)$ and any.

(3) If the referential attribute d in an expression $x.d.a$ itself is locally exclusive, say, then the attribute a of class D_Q may still be overlapping or remotely exclusive. This is illustrated by our example query, where the referential attribute 'owner' is locally exclusive, but 'est-turnov' is provided by both `Company1` and `Company2`.

Our discussion here is restricted to paths of length 1; the generalisation to paths of length n is straightforward. When considering queries involving such a path expression, the following 'naive' algorithm suggests itself, which is based on the well-known technique of pushing selections and projections through a join.

A naive algorithm Let Ξ_{C_Q} be the conditions of Ξ that refer to C_Q; let Π_{C_Q} be the attributes retrieved from C_Q. Define Ξ_{D_Q}, Π_{D_Q} analogously (recall that D_Q is the class that is referred to through the path expression). Apply `RelevantExtensions` to both $Q1 = (\Pi_{C_Q}, C_Q, \Xi_{C_Q})$ and $Q2 = (\Pi_{D_Q}, D_Q, \Xi_{D_Q})$. Then construct subqueries that search all combinations of relevant extensions (C_Q, D_Q) thus found.

Restrictions on extension combinations However, we can further restrict the set of possible combinations of relevant extensions, by exploiting some prop-

erties of the OO-style of modelling object relationships by references instead of joins.

Consider the following general case of an integrated class C_Q with a referential property d that has a domain class D_Q, which is an integrated class as well. Consider an *answer instance* $(\dot{O}_1, \dot{O}_2)$ for a path query referencing both C_Q and D_Q, where $\dot{O}_1 \in C_Q$ and $\dot{O}_2 \in D_Q$.

In restricting the possible relevant (C_Q, D_Q) extension combinations, we use the following principles (and their respective mirror cases):

1. Remotely exclusive objects do not have locally exclusive (referential) attributes.
2. A locally exclusive referential attribute cannot refer to a remotely exclusive object.
3. A locally exclusive object cannot refer to a remotely exclusive object.

Note that the latter two restrictions are valid for OO-style references, in contrast to relationships established through joins. Note furthermore that overlapping referential attributes of overlapping objects may refer to both local and remote objects, depending on the decision function. Table 1 now lists relevant cases and restrictions on possible answer tuples. (df_d denotes the decision function for the referential attribute d).

Condition	*Restriction on* $(\dot{O}_1, \dot{O}_2)$
$d \in A_{LE}$	$\dot{O}_1 \notin C_{RE} \wedge \dot{O}_2 \notin D_{RE}$
$d \in A_{RE}$	$\dot{O}_1 \notin C_{LE} \wedge \dot{O}_2 \notin D_{LE}$
$d \in A_O$	$\dot{O}_1 \in C_{LE} \Rightarrow \dot{O}_2 \notin D_{RE}$
$d \in A_O$	$\dot{O}_1 \in C_{RE} \Rightarrow \dot{O}_2 \notin D_{LE}$
$d \in A_O \wedge df_d = trust(C)$	$\dot{O}_1 \in C_O \Rightarrow \dot{O}_2 \notin D_{RE}$
$d \in A_O \wedge df_d = trust(C')$	$\dot{O}_1 \in C_O \Rightarrow \dot{O}_2 \notin D_{LE}$

Table 1. Restricting extension combinations for path queries.

Example Our example query contains a condition involving the locally exclusive attribute 'owner' and the remotely exclusive attribute 'diam' of the integrated class `OffShoreProductionWell` (C_Q). We assume the definition of the integrated class `Company` (D_Q) is $D_{LE} \cup D_O \cup D_{RE}$. D_Q is restricted based on its overlapping attribute 'est-turnov'. `RelevantExtensions` hence returns $\{C_O\}$ when applied to C_Q. When applied to D_Q, it returns $\{D_{LE}, D_O, D_{RE}\}$, but since the referential attribute 'owner' (d) is locally exclusive, from Table 1 we can infer that D_{RE} cannot contain objects satisfying Q. The modified version of Q is now as follows.

```
unnest
 collect
  collect
   if (not exists d2 in Company2 |
            x.owner.name=d2.name)
   and x.owner.est-turnov> 1000
   then ⟨ x.cost, x.owner.est-turnov⟩  % DLE part
   else  % DO part
    if exists d2 in Company2 | x.owner.name=d2.name
    and avg(x.owner.est-turnov, d2.est-turnov)> 1000
    then ⟨ x.cost, avg(x.owner.est-turnov, d2.est-turnov)⟩
    else ⟨⟩
    endif
   endif
  for y in OffShoreWell
  iff x.well-id=y.well-id and y.diam<1    % CO  part
 for x in ProductionWell
```

4.2 Nested queries

If we allow referential attributes to be set-valued, we must be able to cope with nested queries. Moreover, for set-valued referential attributes, decision functions such as *union* and *intersect* are relevant.

Example Consider the nested query

```
unnest
  collect
    collect ⟨ x.area, y.depth⟩
    for y in x.wells
    iff y.diam>1
  for x in Field
  iff x.est-prod<100
```

where `Field` is the enriched version of `Field1` (i.e. it is defined as $C_{LE} \cup C_O \cup C_{SS}$), 'est-prod' is a locally exclusive attribute, 'depth' and 'area' are overlapping attributes with decision functions *avg* and *trust*(`Field2`), respectively. Furthermore, 'wells' is a set-valued referential attribute with domain class `Well` (the enriched version of `Well1`), with decision function *union*. That is, the wells that are associated with a particular field f in the integrated view are those wells associated to f through its 'wells' property in either `WELLS` or `FIELDS`. Finally, 'diam' is a remotely exclusive attribute of `Well`.

Principles of modifying nested queries The naive algorithm suggested for modifying path queries is equally applicable to nested queries. Analogous to the restrictions defined for path queries, we can restrict the (C_Q, D_Q) extension combinations to be considered, however.

Consider the following general case of a nested query Q against an integrated class C_Q with a set-valued referential property d that has a domain class D_Q, which is an integrated class as well. Consider a possible answer instance $(\dot{O}_1, \{\dot{O}_{21}, \dot{O}_{22}, \ldots \dot{O}_{2n}\})$ to a nested query Q on C_Q, where $\dot{O}_1 \in C_Q$ and $\dot{O}_{2i} \in D_Q, i = 1 \ldots n$. Note that $\{\dot{O}_{21}, \dot{O}_{22}, \ldots \dot{O}_{2n}\} \subseteq \dot{O}_1.d$. Based on the principles listed in the previous subsection, we find the restrictions on possible $(\dot{O}_1, \dot{O}_1.d)$ combinations listed in Table 2.

Condition	*Restriction on* $(\dot{O}_1, \dot{O}_1.d)$
$d \in A_{LE}$	$\dot{O}_1 \notin C_{RE} \wedge \dot{O}_1.d \cap D_{RE} = \emptyset$
$d \in A_{RE}$	$\dot{O}_1 \notin C_{LE} \wedge \dot{O}_1.d \cap D_{LE} = \emptyset$
$d \in A_O$	$\dot{O}_1 \in C_{LE} \Rightarrow \dot{O}_1.d \cap D_{RE} = \emptyset$
$d \in A_O$	$\dot{O}_1 \in C_{RE} \Rightarrow \dot{O}_1.d \cap D_{LE} = \emptyset$
$d \in A_O \wedge df_d = trust(C)$	$\dot{O}_1 \in C_O \Rightarrow \dot{O}_1.d \cap D_{RE} = \emptyset$
$d \in A_O \wedge df_d = trust(C')$	$\dot{O}_1 \in C_O \Rightarrow \dot{O}_1.d \cap D_{LE} = \emptyset$
$d \in A_O \wedge df_d = intersect$	$\dot{O}_1 \in C_O \Rightarrow \dot{O}_1.d \subseteq D_O$

Table 2. Restricting extension combinations for nested queries.

Example Our example query involves an overlapping attribute ('area'), and restricts a locally exclusive attribute ('est-prod') of `Fields` (C_Q); hence `RelevantExtensions` returns $\{C_{LE}, C_O\}$. It further involves an overlapping attribute ('depth'), and restricts a remotely exclusive attribute ('diam') of `Wells` (D_Q); hence `RelevantExtensions` returns $\{D_O, D_{RE}\}$. Since the referencing attribute 'wells' is overlapping, with decision function *union*, Table 2 indicates that we can eliminate the $\{C_{LE}, D_{RE}\}$ combination.

The obtained modification of Q is shown in Figure 4. The modification in case the decision function *intersect* is used for 'fields' is easily obtained from the listed modification by substituting **and** for **or** in the (C_O, D_O) part and eliminating the (C_O, D_{RE}) part, due to the last rule of Table 2.

Query execution In this paper, we focus on global query modification. That is, we have defined syntactical transformations of (OO-)queries referencing integrated classes to queries referencing only local and remote classes. The actual execution of such queries further requires query decomposition and optimisation, and local query processing. The issues arising from OO-queries in these phases are not different from those arising in relational query processing. In particular, we may use optimisation techniques such as the ones in [11] to determine subsets such as C_O, C_{LE} and C_{RE} in a more efficient way. We do not discuss these issues here.

```
unnest unnest
 collect
  collect
   collect ⟨ x.area, avg(z.depth, y.depth)⟩
   for y in Well2
   iff y.diam > 1
   and z.well-id=y.well-id% DO part
  for z in x.wells
 for x in Field1          % CLE part
 iff x.est-prod<100
 and not exists u in Field2 | x.field-id=u.field-id
union
unnest unnest
 collect
  collect
     unnest
      collect
       collect ⟨ x2.area, avg(y1.depth, y2.depth)⟩
       for y2 in Well2
       iff (y1 in x1.wells or y2 in x2.wells)
       and y1.well-id=y2.well-id and y2.diam >1
      for y1 in Well1        % DO part
    union
     collect ⟨ x2.area, y2.depth⟩
     for y2 in x2.wells          % DRE part
     iff y2.diam > 1
     and not exists y1 in Well1 | y1.well-id=y2.well-id
  for x2 in Field2
  iff x1.field-id=x2.field-id          % CO part
 for x1 in Field1
 iff x1.est-prod< 100
```

Figure 4. Example modification of nested query

5 Conclusion

This paper discussed query modification in database federations that use the object-oriented model as a canonical data model. We showed that earlier approaches to processing of relatively simple queries can be extended to deal with typical object-oriented features such as queries containing path expressions and nested queries. We showed how the specification of global complex objects can be exploited to reduce the number of subqueries necessary to evaluate such queries. Moreover, we showed that global query modification is independent of the definition of global classes by introducing a generalised global query modification algorithm.

As to extensions to our approach, note that in [17] we discussed the definition of integrity constraints on the global view from the constraints defined on the local and remote database. This opens the possibility of combining the techniques presented in this paper with those of *semantic query optimisation*, as in e.g. [7,10], for queries against the integrated view. Furthermore, some global queries may directly be answered using retrieval *methods* that have been implemented with local databases. We have discussed issues concerning the applicability of local methods to the integrated view in [19].

References

[1] H. Balsters, R. A. de By & R. Zicari, "Typed sets as a basis for object-oriented database schemas," in *Proceedings Seventh European Conference on Object-Oriented Programming, July 26–30, 1993, Kaiserslautern, Germany,*, Springer–Verlag, New York–Heidelberg–Berlin, 1993, 161–184, See also `http://wwwis.cs.utwente.nl:8080/oodm.html`.

[2] C. Batini, M. Lenzerini & S. B. Navathe, "A comparative analysis of methodologies for database schema integration," *ACM Computing Surveys* 18 (December 1986).

[3] A. L. P. Chen, "Outerjoin optimization in multidatabase systems," in *Proceedings Sixth International Conference on Data Engineering, Los Angeles, CA, February 5–9, 1990*, IEEE Computer Society Press, Washington, DC, 1990, 211–218.

[4] U. Dayal, "Query processing in a multidatabase system," in *Query Processing in Database Systems*, W. Kim, D. S. Reiner & D. S. Batory, eds., Topics in Information Systems, Springer-Verlag, New York–Heidelberg–Berlin, 1985, 81–108.

[5] U. Dayal & H-Y. Hwang, "View definition and generalization for database integration in a multidatabase system," *IEEE Transactions on Software Engineering* 10 (November 1984), 628–645.

[6] D. Fang, S. Ghandeharizadeh & D. McLeod, "An experimental object-based sharing system for networked databases," *The VLDB Journal* 5 (1996), 151–165.

[7] D. Florescu, L. Raschid & P. Valduriez, "Query reformulation in multidatabase systems using semantic knowledge," *International Journal of Cooperative Information Systems* 5 (1996), 431–468.

[8] D. Heimbigner & D. McLeod, "A federated architecture for information management," *ACM Transactions on Office Information Systems* 3 (July 1985), 253–278.

[9] J-L. Koh & A. L. P. Chen, "A mapping strategy for querying multiple object databases with a global object schema," in *IEEE RIDE-DOM Taiwan, April 1995*, IEEE Computer Society Press, Los Alamitos, CA, 1995, 50–57.

[10] E. van Kuijk, F. Pijpers & P. M. G. Apers, "Semantic query optimization in distributed databases," in *ICCI'90, Niagara Falls, Canada*, Springer-Verlag, New York–Heidelberg–Berlin, 1990, 295–303.

[11] E-P. Lim, J. Srivastava & S-Y. Hwang, "An algebraic transformation framework for multidatabase queries," *Distributed and Parallel Databases* 3 (1995), 273–307.

[12] E-P. Lim, J. Srivastava, S. Prabhakar & J. Richardson, "Entity identification in database integration," in *Proceedings Ninth International Conference on Data Engineering, Vienna, Austria, April 19–23, 1993*, IEEE Computer Society Press, Washington, DC, 1993, 294–301.

[13] W. Meng & C. Yu, "Query processing in multidatabase systems," in *Modern Database Systems*, W. Kim, ed., ACM Press, New York, NY, 1995, 551–572.

[14] A. Motro, "Superviews: Virtual integration of multiple databases," *IEEE Transactions on Software Engineering* 13 (July 1987), 785–798.

[15] E. Pitoura, O. Bukhres & A. Elmagarmid, "Object orientation in multidatabase systems," *Computing Surveys* 27 (June 1995), 141–195.

[16] A. P. Sheth & J. A. Larson, "Federated database systems for managing distributed, heterogeneous and autonomous databases," *ACM Computing Surveys* 22 (September 1990), 183–236.

[17] M. W. W. Vermeer & P. M. G. Apers, "The role of integrity constraints in database interoperation," in *Proceedings 22nd International Conference on Very Large Databases (VLDB'96), Bombay, India*, Morgan Kaufmann Publishers, San Mateo, CA, 1996, 425–435.

[18] M. W. W. Vermeer & P. M. G. Apers, "On the applicability of schema integration techniques to database interoperation," in *Proceedings Fifteenth International Conference on Conceptual Modelling (ER'96), Cottbus, Germany*, Springer-Verlag, Berlin, 1996.

[19] M. W. W. Vermeer & P. M. G. Apers, "Behaviour specification in database interoperation," in *Proc. 9th International Conference on Advanced Information Systems Engineering (CAiSE'97), Barcelona, Spain*, Springer-Verlag, New York–Heidelberg–Berlin, 1997.