Providing Views and Closure for the ODMG Object Model

Mark Roantree

School of Computer Applications, Dublin City University, Ireland.
mark.roantree@compapp.dcu.ie

Jessie B. Kennedy and Peter J. Barclay

School of Computing, Napier University, Scotland.

Abstract

The ODMG Object Model offers a standard for object-oriented database designers, while attempting to address some issues of interoperability. This research is focused on the viability of using the ODMG data model as a canonical data model in a multidatabase environment, and where weaknesses are identified we have proposed amendments to enable the model to suit the specific needs of this type of distributed database system. This paper describes our efforts to extend its relational style algebra, and to provide query closure and a viewing mechanism for OQL to construct multidatabase schemas.

Key words: Object Model, Multidatabases, ODMG, Views, OQL

1 Introduction

The term *federated database* system is used to refer to a collection of autonomous heterogenous software systems. A Federated Database architecture was introduced in [20], with a more detailed classification described in [2].

The term multidatabase has been used to refer to all types of distributed database systems [13] and not just those with an architecture where participating systems use different data models. A more specialised form of multidatabase system has emerged in the form of object-oriented multidatabase systems [1] where these systems use an object model to represent the global schema. As a federated system contains a quite specific architecture, we will use the term *multidatabase system* as it covers all types of heterogenous distributed database systems which do not have a single global schema. This paper employs the terminology in [20] when discussing the architecture of multidatabase systems. The paper is structured as follows: the remainder of §1 will provide a brief description of multidatabase systems; §2 describes our evaluation process for the Object Data Management Group (ODMG) model and the results of this evaluation; §3 describes Multidatabase Object Query Language (MOQL) which amends the use of the ODMG Object Model for multidatabase systems; finally, §4 offers some conclusions.

1.1 Multidatabase Systems

A multidatabase system allows local databases (LDB) to interoperate with other LDBs even though they may use a different data model. This is achieved by converting each LDB data model into a canonical data model (CDM) representation. We have chosen an object model as our CDM in agreement with the conclusions reached in [19] which advocate a model with strong expressive qualities, at least as expressive as each local data model; moreover in the ODMG object model an attempt has been made to provide a standard object model. The conversion process involves the creation of a new schema called the *component schema* [20] which is modelled using the CDM and contains mappings to the local schema. A separate conversion process for each type of data model is required involving many issues which are discussed elsewhere [4,14]. Once the component schema has been generated, various *export schemata* are derived on top of each component schema in the same way as we derive views on traditional data models. This helps to provide a layer of security for the architecture. *Federated schemata* are then constructed using export schemata. In the architecture of [20] they also provide an external layer where the federated schema (which uses the canonical model) is transformed to the model of choice for the end-user.

2 Evaluation of ODMG and OQL

The ODMG group has proposed a standard object model in an attempt to address the incompatability problem of each object-oriented database using its own object model. As a result of this standard, many multidatabase groups [7,8] have chosen to adopt the ODMG model as their canonical data model. As the ODMG object model was originally designed for object-oriented database systems, part of our research focused on identifying enhancements required for multidatabase systems [16]. In this evaluation we chose to modify an existing framework and compare the ODMG object model with other object models in terms of base model functionality and any possible multidatabase functionality. Our goal is that a potential canonical model should support all the necessary operators to construct not only the component schema, but also export and federated schemata. In [22] the authors proposed a framework for evaluating some of the first generation object models such as ENCORE and IRIS. The purpose of that particular research was to evaluate object query models. Five metrics were employed as part of their evaluation framework: object-orientedness, expressiveness, formalness, performance and database issues. We chose to use this framework as a basis for our evaluation process for canonical data models while modifying some aspects to support multidatabase systems. Some of the characteristics of these metrics are modified and the *performance* metric was replaced with a *multidatabase* metric to suit the requirements of this particular area. In *table 1* each individual evaluation property is listed together with the result for the ODMG OM.

Property	Code	Description	ODMG
Object-Oriented	OO1	Class/collections	Not Satisfied
Object-Oriented	OO2	Heterogenous sets	Partial
Expressive	E1	Extension to relational algebra	Partial
Expressive	E2	Dynamic type creation	Not Satisfied
Expressive	E3	Manipulation of Behaviour	Not Satisfied
Formal	F1	Formal Semantics	Not Satisfied
Formal	F2	Closed Algebra	Not Satisfied
Database	D1	Object Calculus	Not Satisfied
Database	D2	Database Operators	Satisfied
Multidatabase	M1	Views (virtual schemata)	Partial
Multidatabase	M1	Multidatabase Operators	Not Satisfied
Multidatabase	M1	Mapping Language	Not Satisfied

Table 1. Evaluation Framework and ODMG Model Results

The first of the object-oriented metrics, the class/collection metric (OO_1) provides a facility for managing collections of objects. It requires that some mechanism be available to split instances of the same type into separate collections. For example, an attribute *Dept* may have different values and it may be unsuitable to place each instance of the *Lecturer* type into one single collection. Instead, it may be desirable to group all *Lecturer* types by *Department.* A new instance of a *Lecturer* type becomes a member of one of these collections when created, based on its value for *Dept.* The second metric (OO_2) advocates support for heterogenous sets and sequences. This allows subtypes to be grouped as long as they have a common supertype.

The first of the expressive properties (E_1) is essential to the object model i.e. the ability to be able to perform operations on the data. The data model should supply an algebra with operators for selection, projection, join, union and difference. Property E_2 requires support for dynamic type creation. The third expressive property (E_3) advocates support for the manipulation of behaviour. This property is also useful in multidatabase systems where behaviour needs to be modified as part of the integration process.

The formal properties require a formal semantics (F_1) and closed algebra (F_2) for the object model.

The database properties require the inclusion of an *object calculus* and we have included a new property D_2, which tests for operators specifically designed for database functionality. For example, integrity constraints and primary key functions, and some mechanism for building relationships between types should form part of a data model for database design.

The multidatabase properties are *support for views* (M_1), *a multidatabase algebra* (M_2), and a *mapping language* (M_3). The *support for views* property could have been placed under the database properties as it is regarded as a fundamental part of relational databases. However, since multidatabase systems require a special form of view (export and federated schema) they are included here. Conventional views provide a *1-1* mapping between the attribute name in the view and the attribute value in the base relation or class. However, some federated schema attributes will require a *1-n* mapping. An example of this in a healthcare environment is where an attribute such a *patient id* is used to combine results subsets from different participating databases. The federated schema will contain an attribute called *patient id* which will map to a similar attribute in multiple participating databases. A good example of a multidatabase algebra is found in [12] where it is necessary to create a virtual schema (federated schema in our architecture) which maps to various local schemata through a series of schema editing operations. When a query is passed to a federated schema, it can be decomposed and passed to the relevant local schemata by examining the operations which constructed the federated schema (multiview in some literature [12]). A mapping language is used to aid in the conversion of schemata to the canonical data model. This language is part of the interface between component schemata and local schemata described briefly in §2.3. All three properties are required for construction of an integrated (federated) schema and should all be incorporated in a suitable view mechanism. However, we deliberately identified these properties separately as it is possible that some models could support mapping (to create derived attribtes) but not not a full viewing mechanism. Additionaly, models could support integration operators but again, not support a full view mechanism where integrated schemata could be stored and subsequently reused to form new federated schemata.

2.2 ODMG Object Model Performance

The Object Data Management Group's Object Model (ODMG-OM) uses its Object Definition Language (ODL) to define schemata and its Object Query Language (OQL) to perform retrieval and update[1] operations. The basic unit for modelling in the ODMG OM is the *object* which itself is categorised into *types*. The state of an object is modelled by its *attributes* and its *relationships* with other objects.

2.2.1 Object-Oriented Properties

The ODMG OM does not support OO_2 as all collections must be of the same type. A collection such as a *list* is specifically defined using type generators and a collection is an instance of a *collection type*. It is then necessary to maintain the group by adding or deleting objects. This implies that property OO_1 is not satisfied either. The ODMG standard [6] states that even if new objects are instantiated which satisfy the predicate for a collection they are not automatically added to the collection. It also goes on to state that objects which no longer satisfy the collection remain in the collection: they must be removed by some direct operation. Although research is ongoing in this area, a solution to satisfy the OO_1 property is mentioned in §3.

[1] Although data retrieval operators are defined as part of OQL, no update operators are defined. OQL relies upon *update methods* defined in the schema to handle these operations.

2.2.2 Expressive Properties

OQL does not fully satisfy property E_1 as not all relational operators are supported. Queries can return either objects, values or sets. However, no new objects can be defined as a result of a query. This implies that project and join operators are only supported through the retrieval of structures. This demonstrates that the closure property is not satisfied. Only the select operator is present in the examples given. Even minimal query languages such as NOODL [3] can be used to construct views because the closure property is satisfied. To illustrate this, *example 1* requires a projection of the Patient type which is valid in SQL (for relational databases), but not permitted in OQL(for o-o databases).

Example 1 *An OQL query which returns a set of structs and not objects*

select name,dob,bloodtype

from Patients P

Property E_2 is not supported either as there is no provision for the dynamic creation of virtual types. Instead ODL is used to predefine all types in the schema. In the previous example, we may not wish to create a supertype in the schema for *Patient* that contains the selected attributes. Instead we may prefer to create a virtual type which acts as a view of the *Patient* base type.

No provision exists for manipulation of behaviour. Behaviour is defined using ODL and is not modifiable. For example, an operation to derive an attribute to calculate a person's net salary in *Irish Pounds* may need to be modifed to derive the salary in *Euros*. This is not possible in ODL, and thus, property E_3 is not satisfied either.

2.2.3 Formal Properties

No formal semantics exist for the object model so property F_1 is not satisfied. This is not surprising as the model is still evolving and is thus difficult to formalise. Since the algebra allows the retrieval of objects, sets and values, it is not closed and property F_2 is not satisfied. This is a more serious problem. While it is possible to infer the output from an OQL query by examining the operator (*select* or *select struct*), it requires a degree of preprocessing which is cumbersome, and it leads to a loss of independence in the sense that query result types are bound to specific query operators. This is contrary to the relational algebra where all operators generate a single structure: a relation. Furthermore, as it does not appear to be possible to query a structure, a query which returns a structure cannot form the inner part of a nested query.

2.2.4 Database Properties

No object calculus has been defined and thus property D_1 is not satisfied. The ODMG OM was designed specifically to model databases and is equipped with operators to support this. One of the constructs used to model state is the *relationship* type. Relationships are bi-directional denoted by the `relationship` and `inverse` keywords.

ODMG OM supports a nested transaction model by providing a *Transaction* type which may be reused. There is also *Database*, *Schema* and *Subschema* types which are part of the object model. Metadata access is also part of the OM. It uses the fact that ODL definitions are predefined and available as objects of type *Type*. Thus, property D_2 has been satisfied.

2.2.5 Multidatabase Properties

There is no mechanism for views, no multidatabase algebra nor any mapping language in the ODMG OM so properties M_1, M_2 and M_3 are not satisfied. This is understandable since the model was not intended for multidatabase systems. Although views are 'supported' in the latest version of the model, they exist only as 'named defined queries' and do not appear to be reusable as input to new queries.

2.3 ODMG Object Model in a multidatabase environment

The role of the ODMG OM is demonstrated in *figure 1* where the canonical model schemata are represented by component schemata (C1, C2, C3), export schemata (E1, E2, E3, E4, E5, E6, E7), and federated schemata (F1). The component schemata are defined using ODL although functions rather than attributes are used to model data. We take the view that the component schema is an ODMG schema where attributes are represented as return values of functions. These functions serve to encapsulate the inter-model mappings and schema transformations. This has been done in the past using mediators [15] or agents [10] which reside at the local database. These agents are local applications at each local database, and may hold the component schema of the local database, which acts as a client (when the local site wishes to make global queries) and as a server (where the local database supplies results to external global queries). At an implementational level, the ODL schema is no different from a centralised ODMG schema where retrieval and update operations are performed using functions to read/write the actual data values. This is a common practice where it is desirable to preserve encapsulation. However, the retrieval functions are more complex in the multidatabase model as they will form the interface between the multidatabase system and the local

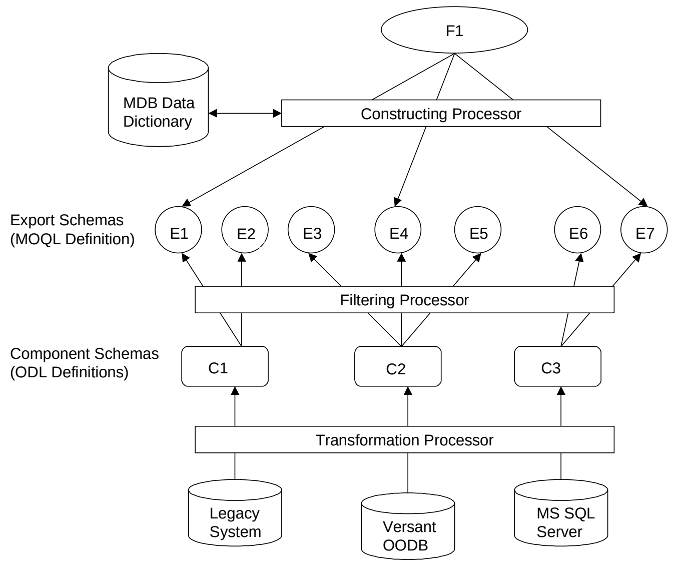

Fig. 1. *The ODMG model used as a canonical model in a federated database architecture. A Transformation Processor is used to convert local schemata to component schemata; A filtering processors are used to filter data into export schemata; and a Construction Processor is used to build the federated schemata.*

database system. In earlier work [17] the construction of such interfaces for four different healthcare software systems was described. Although the ODMG model was not employed, the definition and construction of interfaces between the common model and local models is similar.

Once the component schemata have been defined, OQL is used to derive views which can be used as export schemata. This is made possible through the use of query definitions in the existing ODMG model but it will not be possible to reuse these views which make them useless for definitions of federated schemata. A federated schema must reuse the existing export schemata in its definition. The extended version of OQL proposed in this paper can be used to define both export and federated schemata.

3 MOQL: An Multidatabase Extension to OQL

In this section we discuss Multidatabase Object Query Language (MOQL) which is a result of our efforts to provide closure for OQL, to make the algebra at least as expressive as the relational algebra, thereby providing the facility to define views for ODMG databases. Before we can define our view semantics it is necessary to have a closed algebra similar to the relational algebra where the output is always known (i.e. a relation). It is then necessary to ensure that our algebra is sufficiently expressive to (appear to) create the new types required. Finally, we require the facility to define and store views which provide

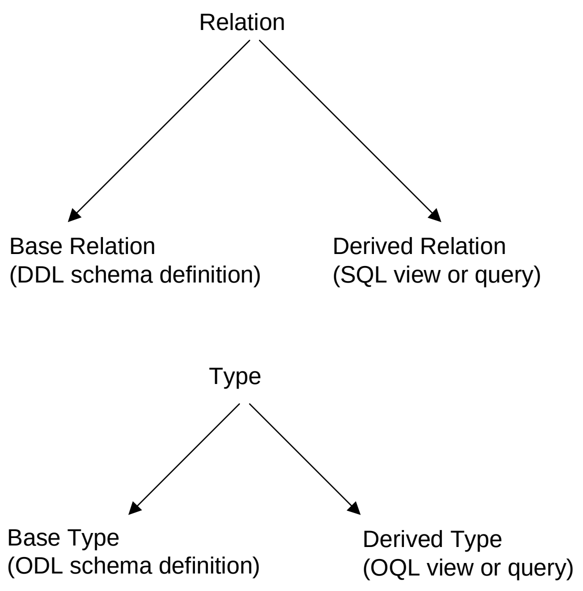

Fig. 2. *Database Objects for Queries. A relational query results in a set of tuples of a base relation or some derived relation. An object-oriented query results in a set of objects of a base type or some derived type.*

mappings to data values in the object schema.

In our extended model we use ODMG types defined by ODL and classes as implementations of types. However, we permit both base classes (based on ODL type definitions) and virtual classes defined in MOQL. A *type* can have any number of *classes* which are used to manage collections of objects. This is similar to the architecture employed in the COCOON model [21]. A simple example is where a type called *TaxablePerson* which represents people who pay income tax and are grouped by an attribute called *SalaryBand*. A predicate based on the *SalaryBand* type is used to create collections of *TaxablePerson* objects. We may have three collections based on three values, $SalaryBand \leq$ 20k, $SalaryBand \geq$ 20k and $\leq$ 50k, and $SalaryBand \geq$50. Thus, ODL is used to define types and OQL is used to define a predicate for each type which leads to the separation of n classes for each type. for these examples, we are still dealing with concrete types which are present in the type hierarchy

3.1 Query Closure for OQL

The OQL language has potential problems in that queries can return objects, sets or values, and thus it is difficult to nest queries or create views when the return type is not consistent. Our flavour of OQL has been modified slightly to restrict queries to return only a set of objects. We have modelled our design on the relational model which returns a *relation*, i.e. a set of tuples.

To examine the analogy, we first take the relational model which deals with *Relation* objects. There are two types of these objects: *Base* and *Derived* Relation objects. *Figure 2* demonstrates a simple inheritance hierarchy where the type *Relation* has two sub-types, *BaseRelation* and *DerivedRelation.* A query will always return the supertype and the database management system determines whether it is a base or derived structure before calculating the results.

In the ODMG model, a *type* is the main modelling primitive in much the same way as a *relation* is the modelling primitive in the relational model. Whereas a relation is a simpler structure containing a set of attributes, a type contains a set of attributes (some of which may be other types), some behavioural characteristics, and it may inherit some of its structure and behaviour from an existing type.

3.2 Extensions to the OQL Algebra

OQL does not create new types as a result of a query but returns existing types. Although MOQL obeys the same rules, it can return *derived* types in the same way as the relational model returns derived relations. This is contrary to the ODMG 2.0 specification where OQL returns "atoms, structures, collections and literals" [5]. In the following examples for selection and projection the difference between OQL and MOQL is demonstrated. In §3.4 an example of a join is provided.

3.2.1 Selection

The following example taken from [5] demonstrates a selection query using OQL. The query demonstrates the power of object-oriented databases where objects can navigate to other objects following a particular relationship. The query in *example 2* retrieves the address for the children of each *Person* in the database.

Example 2 *Retrieval of base objects*

select c.address

from Persons p, p.children c

The MOQL format for the query is identical. The difference is in the returned values: OQL returns a set of structs containing addresses whereas MOQL returns a set of objects of a derived type. A derived type results from the need to filter (or hide) data before exporting. Our derived types can be used as input to subsequent queries in order to filter data further for different types of users. As it stands the MOQL query can be nested as the result is a set of objects. It can also be reused if it is contained inside a view definition (see §3.3).

3.2.2 Projection

The projection query in *example 3* is also taken from [5] where a predicate is used to restrict the output of the previous query. Once again the syntax for OQL and MOQL queries are identical.

Example 3 *Projection Query*

```
select c.address

from Persons p,p.children c

 where p.address.street = "Main Street" and

 count(p.children) >= 2 and

 c.address.city != p.address.city
```

This particular example highlights a potential problem for views in object-oriented databases where updates are required, and also for future view definitions which reuse this view. The problem is that the attribute *Person.children* which is part of the predicate is not present in the view. Subsequent view definitions cannot *see* this attribute and additions to the database through this view are impossible. In [11] they employ a *materialisability rule* which states that a view which does not contain attributes used in the condition part of the view definition are not updatable. We choose to include the attribute in the display part of the view. OQL returns a set of structs for this modified query whereas MOQL returns a set of derived objects

3.2.3 Preserving Encapsulation

In §2.3 we discussed the fact that attributes exist as functions at the component schema level. Multidatabase schemata are virtual schemata with mappings between higher level schemata and attributes at the local database level. This type of mapping supports encapsulation as access to local attributes is available only through an ODL interface. When defining views on ODL schemata or OQL query results, MOQL accesses functions which return attribute values. Component schemata are defined using ODL. Each attribute is encapsulated with `Get` and `Set` functions. In a conventional OODB, the `Get`

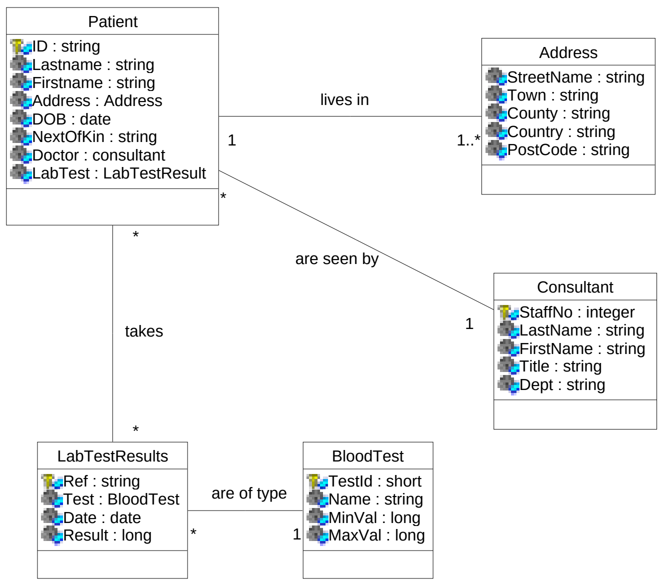

Fig. 3. *A simple healthcare database modelling using the Unified Modeling Language.*

function returns or computes an attribute value and returns the value to the retrieval query. In our proposed multidatabase architecture, each `Get` function must interface with the local schema to retrieve the required value from the physical database.

3.3 Providing View Support for the ODMG OM

In this section we discuss how MOQL can be used to provide a view mechanism similar to that of the relational model. For simplicity the sample views are created at a single site and represent export schemata. The sample views are intended to demonstrate the difference between OQL and the extensions provided by MOQL. Research which is focused on the view mechanism itself is presented in [18] and contains a series of complex view samples.

The example in *figure 3* demonstrates a subsection of the Component Schema for a Laboratory Information System (LIS) using the Unified Modelling Language [9]. There are five objects that contain information to be shared with another participating database. We have a requirement to export patient data to senior administration (*ManagementView*), a subset of this view to medical staff (*StaffView*), and a different subset of the same view for a particular consultant (*WardView*).

An OQL user will have no problem creating the first query or view. Using the named query definition feature a query can be created and stored persistently which also serves as a view.

The management query is for patient id, name, address, dob, consultant, blood test reference, blood test name, blood test result for all patients in the 'HIV' department.

Using OQL we can define a query as illistrated in *example 4*.

Example 4 *OQL Query*

```
define ManagementView as
  select D.id, D.lastname, D.firstname, D.address.StreetName,
  D.address.Town, D.address.County, D.address.Postcode, D.dob,
  D.doctor.name, B.Ref, B.test.name, B.result
  from Demographics D, BloodTest B
    where D.doctor.dept = "HIV"
```

Using MOQL the query is similar.

Example 5 *MOQL Query*

```
CreateView ManagementView as
  select D.id, D.lastname, D.firstname, D.address.StreetName,
  D.address.Town, D.address.County, D.address.Postcode, D.dob,
  D.doctor.name, B.Ref, B.test.name, B.result
  as PatientID, lname, fname, addr1, addr2, addr3, addr4, dob,
  consultant, testref, testname, testresult
  from Demographics D, BloodTest B
    where D.doctor.dept = "HIV"
```

The above OQL query definition cannot be reused as an input into a second query. In OQL it will be necessary to define three separate queries, although subsequent queries (examples 6 and 7) should be able to reuse the first query (illustrated in example 5). Additionally, it is not clear how the first query can be used as the inner query in a nested structure as it will return a set of structs which OQL cannot use as an input to a query.

Using MOQL the first query returns a derived object which we can use to create the remaining views shown in examples 6 and 7.

Example 6 *Reuse of ManagementView*

```
CreateView StaffView as
 select M.lname, M.fname, M.dob, M.testresult
 from ManagementView M
   where M.testresult > 40
```

This provides a list of patients who have a positive result to their blood test.

Example 7 *Reuse of ManagementView*

```
CreateView Wardview as
 select M.lname, M.fname
 from ManagementView M
   where M.consultant = "Ward"
```

This provides Doctor Ward with his list of HIV patients. Note that there is no reason why a view cannot use more than one view, or a combination of a views and base schema objects.

4 Conclusions and Future Research

In this paper we attempted to use the ODMG object model in a multidatabase environment. As it stands, it falls short of meeting the needs of this type of database system. Its shortcomings are due to the fact that it was mainly designed for object-oriented databases and not for multidatabase systems. Additionally, the failure of OQL to return a single modelling primitive leads to difficulty in constructing reusable views and nested queries. We have shown how to overcome this problem by returning ODMG derived types as a result of all OQL queries. This has also provided us with a mechanism for defining reusable views.

One of our objectives has been to retain compatibility with the ODMG-93 OM specification. Our enhancements have been for the purpose of providing multidatabase functionality in the form of reusable views which can be used to define export and federated schemata. To that extent, we have not compromised compatibility with the ODMG standard. MOQL is a multidatabase extension for OQL with the capability to create and manipulate derived types. OQL and MOQL queries return the same results.

Although the ODMG model has fared badly in the evaluation process [16] and was not intended for usage in a multidatabase system, we have decided to continue to use it for multidatabase design and implementation. Our reasons are mainly of a practical nature. A standard for object database modelling and design is necessary for the future of object oriented databases and multidatabases. In addition, there are some planned features which are beneficial to multidatabase systems. For example, versioning will enable us to manage schema evolution at local database level. Integrity constraints can be declared on attribute types; transactions may access objects in more than one database; and metadata is available as a predefined schema. These are all characteristics required of multidatabase systems.

Although our initial appraisal of the model, and our first definition of views and MOQL were all based on the ODMG 1.2 specification, we have recently updated to the ODMG 2.0 [6] specification. There are many similarities between the versions which meant that the major thrust of ur work is unchanged. Improvements in the model (for example the specification of an access interface for the schema repository) has helped in the design of an implementation for our view mechanism. This implementation extends the ODMG schema repository and is breifly described in [18], with a fuller description due for submission shortly.

Finally, we are specifying a process for transferring export schemata between information servers and the multidatabase kernel. This is based on using a CORBA object to browse the schema repository of the component schema and retrieve export schemata before passing them to the kernel where they will undergo an integration process.

The authors would like to acknowledge the helpful comments of the annonynous referees which clarified some of the issues discussed in the paper.

References

[1] O. Bukhres and A. Elmagarmid (eds). *Object-Oriented Multidatabase Systems.* Prentice Hall, 1996.

[2] M. Bright, A. Hurson and S. Pakzad. A Taxonomy and Current Issues in Multidatabase Systems. *IEEE Computer*, December 1991.

[3] P. Barclay and J. Kennedy. Viewing Objects. *Proceedings of 11th British Conference on Database Systems*, Springer, 1993.

[4] C. Batini, M. Lenzerini and S. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, vol. 18, no. 4, December 1986.

[5] R. Cattell (editor). *The Object Database Standard: ODMG-93.* Morgan Kaufmann, 1996.

[6] R. Catell and D. Barry (eds). *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann, 1997.

[7] S. Conrad, B. Eaglestone, W. Hasselbring, M. Roantree, F. Saltor, M. Schonhoff, M. Strassler, M. Vermeer. Research Issues in Federated Database Systems. *ACM SIGMOD Record*, Vol. 26, No. 4, December 1997.

[8] S. Conrad, W. Hasselbring, A. Heuer, G. Saake. Engineering Federated Database Systems. *Proceedings of the CAISE 97 Workshop*, Preprint 6/1997, University of Magdeburg, 1997.

[9] H. Eriksson and M. Penker. *UML Toolkit*, Wiley, 1998.

[10] W. Hasselbring. Federated integration of replicated information within hospitals. *Intl. Journal on Digital Libraries*, Vol. 1, No. 3, Nov. 1997.

[11] W. Kim and W. Kelley. On View Support in Object-Oriented Database Systems in *Modern Database Systems: The Object Model, Interoperability, and Beyond*, Won Kim (ed), Addison-Wesley, 1995.

[12] A. Motro. Superviews: Visual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, vol. se-13. no. 7, 1987.

[13] M. Tamer Özsu and P. Valduriez. Distributed Database Systems: Where are we now? *IEEE Computer* vol. 24, no. 8, 1991.

[14] E. Pitoura, O. Bukhres and A. Elmagarmid. Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, Vol. 27, No. 2, June 1995.

[15] M. Papazoglou, Z. Tari and N. Russell. Object-Oriented Technology for Interschema and Language Mappings. In [1].

[16] M. Roantree. Evaluating the ODMG Object Model for Usage in a Multidatabase Environment. *Technical Report No. CA-2597*, Dublin City University, 1997.

[17] M. Roantree, P. Hickey, A. Crilly, J. Cardiff, J. Murphy. Metadata Modelling for Healthcare Applications in a Federated Database System. *Intl. Workshop on Trends in Distributed Systems*, LNCS No. 1161, 1996.

[18] M. Roantree, J. Kennedy and P. Barclay. A Multidatabase Layer for the ODMG Object Model. *Proceedings of 5th International Conference on Object-Oriented Information Systems*, Springer, 1998.

[19] F. Saltor, M. Castellanos and M. Garcia-Solaco. Suitability of Data Models as Canonical Models for Federated Databases. *SIGMOD Record* vol. 20, no. 4, 1991.

[20] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, vol. 22, no. 3, September 1990.

[21] M. Scholl, H. Schek and M. Tresch. Object Algebra and Views for Multi-Objectbases. *Proceedings of Workshop on Distributed Object Management*, 1992.

[22] An Evaluation Framework for Algebraic Object-Oriented Query Models. *Proceedings of 7th International Conference on Data Engineering,* IEEE Computer Society Press, 1991.