Interoperable Services for Federations of Database Systems[⋆]

Mark Roantree[1], Jessie B. Kennedy[2], Peter J. Barclay[2]

[1] School of Computer Applications, Dublin City University, Dublin, Ireland.
{mark.roantree@compapp.dcu.ie}
[2] School of Computing, Napier University, Edinburgh, Scotland.

Abstract. *The problems associated with defining interoperable protocols for heterogenous information systems has been the subject of researchers for many years. While numerous solutions have been offered, the problem remains unsolved, mainly due to the distinctive nature of each target environment, and the uniqueness of each solution. This is not helped by the use of proprietary common data models or the use of non-open or non-standard technologies to provide an infrastructure and services. In this research, services and processors were defined and implemented using a platform of existing standards, in order to provide a more open platform for system integration, and to enhance forward interoperability with new systems over time.*

1 Introduction

The concept of a federation of databases [24] is one in where heterogenous databases (or information systems) can communicate with each other through an interface provided by a common data model. In this research, the common data model is the ODMG model, the standard model for object-oriented databases since 1993 [6]. The most common architecture for these systems is as follows: data resides in (generally heterogenous) information systems or databases; the schema of each Information System (IS) is generally translated to an O-O format, and this new schema is called the component schema; view schemata are defined as subsets of the component schema, and these are shared with other systems; the view schemata are exported to a global or federated server where they are integrated to form multiple global or federated schemata.

These requirements necessitate wrapper and view services for an ODMG database: to provide wrapper specifications to non-ODMG information systems; and view specifications for ODMG databases. In the case of the view specifications, they must be capable of retaining as much semantic information as possible, and thus, a view which results in a single virtual class, is not sufficient. Instead, the view should resemble a subschema that retains information regarding inheritance and relationships between classes. Furthermore, this view

[⋆] Supported by Forbairt Strategic Research Programme ST/98/014

mechanism must be powerful enough to support the restructuring of the subschema specification for subsequent schema integration operations. The architecture must also include a transport service containing processors that display different levels of genericity. The contribution of this research is to enhance the interoperable 'power' of previous federated architectures by eliminating or minimizing the use of proprietary models and support services. In doing so, we introduce federated services for ODMG databases which provide view and wrapper languages, and transport services for moving metadata and data. This paper is structured as follows: in the remainder of this section, we provide a motivation for our research, describe the contribution offered, and discuss related work; in §2, the integration system architecture and integration method is described; in §3, integration services which are formed using open standards are presented; and finally in §4, some conclusions are offered. An extended version of this paper [20] includes details of integration operators and implementation.

1.1 Background and Motivation

Information systems integration is as relevant now as it was ten or more years ago when federated database architectures were first proposed. However, unlike the era when researchers first tackled the problem, issues such as *interoperability*, *open systems* and *distribution standards* are more prominent. The use of Open Systems design technologies has been recommended for a number of years in an attempt to make future integration of previously autonomous systems, less complex. While it must be accepted that the possibility of building systems which are fully interoperable with all other systems is not possible, the usage of published information technology standards enhances the type of integration which is not always obvious at the time of systems development.

The OASIS project [16] focused on the research and implementation of a federation of large healthcare systems. A major aspect of this research was the definition and implementation of services based on standard technologies. This research sought to define integration services and languages using open and modern underlying technologies. While it is necessary to 'extend' certain standards in order to provide new features, we sought to keep these extensions both minimal, and in the 'flavor' of the original standard. This research required the identification of suitable software and database standards; a subsequent specification of what was required to extend these standards to design services for federated information systems [17]; and finally the development of a prototype suitable for testing the practical implications of using the services provided in this research.

1.2 Related Work

In terms of architecture and processors, this research is based upon the standard presented in [24], and is thus, similar to federated architectures such as that employed in the AQUA project [11], which contains services similar to those offered here. Apart from AQUA's use of a proprietary data model, the key difference is that most of the effort in AQUA was employed in building an architecture to support global transactions (eg. federated schema updates) whereas in our research we concentrate on the power of the view mechanism. There are some subtle architectural differences in that the AQUA model supports a form of encapsulation whereas the ODMG does not[1]. Although this certainly provides security, it has the drawback in that some client software and all view services must be built (compiled and linked) with the data access functions in order to retrieve data and display views. When our Display Service receives a request to materialize a view, a general query is passed to local information systems to retrieve information, and subsequent Object Query Language (OQL) queries are used to filter and format the data.

In terms of object-oriented view services, most concentrate on supporting a single information server, and have no need for the integration operators required in a federated architecture. Although view mechanisms such as [22][7][25] do provide a means of defining virtual schemata (rather than virtual classes), they use proprietary object models (which hinder interoperability) and do not include valuable restructuring operators such as those described in [10]. Similarly in [8] [15], they do not supply restructuring operations, although in these cases, they do employ the standard object database model [6].

Although this research has chosen to employ the ODMG database model as standard on which to base higher level (federated) operations, and thus, as the pivotal design model for the architecture, it is accepted that the standard is still evolving, and in some cases it needs to address design issues which contains potential problems for database engineers. For example, there are issues concerning the semantics of the model, the adoption of an SQL-type query language, and its 'views' on encapsulation [27]; in a deeper analysis of its Object Query Language, there are concerns for the support of compositionality, guaranteed access to all portions of an object' state (due to lack of identification support for arrays of structures); and examples of unsafe query side effects [26]; and in [1] there are numerous examples including the persistence model, parametric polymorphism and reflection. These are important issues which have been highlighted by researchers, and have not all been addressed in the latest version of the standard [5]. However, we must limit the scope of our research to the provision of federation extensions, and trust that these base model concerns are addressed in subsequent revisions of the standard.

2 System Architecture

The system architecture is based on an earlier architecture [24] (now regarded as the standard federated architecture) which employed a series of layers to facilitate the integration of two or more heterogenous information systems. It employs the ODMG model as the common model for sharing and exchanging data between participating systems. Early research into federated database systems [23] indicated that object models are best candidates as common models, and we

[1] In ODMG the keyword `private` is ignored during class definition.

extend this requirement by stating that the use of proprietary object models should be avoided as it hinders interoperability. Specifically, we differ from the standard devised in [24] in a number of key areas.

– Firstly, we propose the ODMG model as the common model. Theoretically, this provides a uniform interface to which other federations can interact with federated and local schemata in our architecture. The standard API for the schema repository also provides a uniform interface for constructing, querying, and manipulating view schemata, a necessary feature for query transformation and optimization. Since the ODMG model provides an access gateway for the this architecture, the external layer in the original architecture [24] is not required.
– Secondly, we introduce a transport layer to manage the movement of data between local information servers and the federated kernel. This provides a controlled mechanism for moving view definitions to the federated kernel, and ultimately the movement of data (if requested).
– Thirdly, we have replaced the accessing processors with an object manager with a set of very specific functions. Each of these layers is discussed in detail shortly.

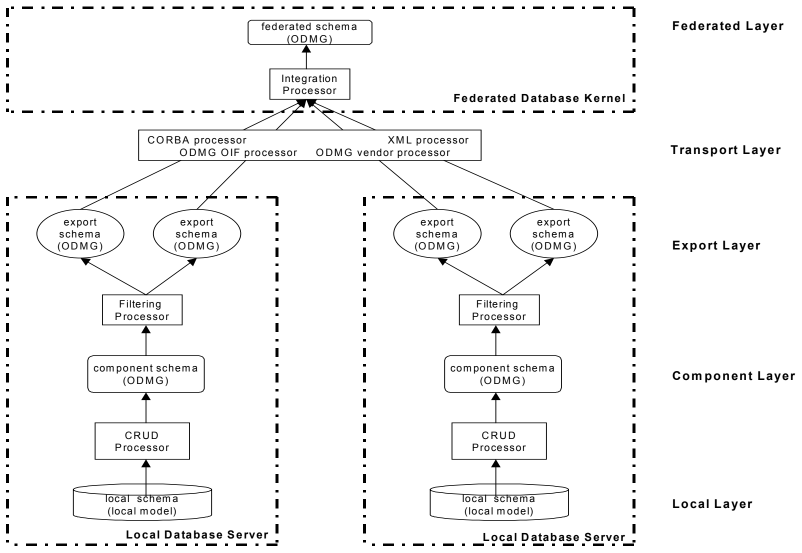

Fig. 1. *Integration Architecture in the ODMG Federation.*

The architecture in *figure 1* comprises five layers and four processor types for the integration of non-ODMG information systems. ODMG databases begin at the second layer (component layer) and undergo three integration processes.

Any number of local systems may be integrated to form a federated schema, with integration using a binary ladder strategy as described in [3]. This means that processors permit two schemata to be (firstly defined and then) integrated to form a first pass of the federated schema. Subsequent transported (local) schemata can then be added, one at a time, to make the final version of the federated schema. Although *figure 1* shows only one federated schema (and thus one federation), any number of these global schemata may be constructed, facilitating the operation of multiple concurrent federations of data. The roles of the system processors in schema integration are now described in more detail.

2.1 CRUD Object Manager Processor

An object manager is required to interact between the bottom two layers in the architecture. This provides *Creation*, *Retrieval*, *Updating* and *Deletion* (CRUD) of objects in the local information system, and is thus labelled, the CRUD processor. Not all information systems will be capable of providing all four features, but one must permit the retrieval process for a minimum level of participation. The CRUD processor can be seen as one local application which runs concurrently with all remaining local applications. Modern systems such as Oracle or Informix should support the creation, updating and deletion of local objects, subject to the local rules of transaction management. However, some legacy systems may not be able to guarantee write operations, and may thus participate in *read-only* federations. For those systems which can support write operations, it is incorrect to assume that this provides an obvious support for global transactions such as those supported in traditional distributed database systems. Each federation will require its own unique analysis to determine if global transactions are possible, and then a purpose-built transaction protocol to guarantee integrity of data. Such protocols are outside the scope of an abstract architecture such as ours, but typical examples are provided in [11] and [14].

This step in the integration process requires further services, some of which are external to this research. For example, an ODMG representation of the local data model must be compiled. It is generally accepted that this process can never be fully automated although for some conversions (relational to object-oriented), automated processes can perform much of this task. Our research assumes that this task has been completed, and that the input for our architecture is an ODMG component schema, or in practical terms, an ODMG database (known as the component database) with a schema defined, but with no physical data. The CRUD processor requires a mapping language to 'bind' the local database schema with the ODMG representation in order that it may extract data from, or update data inside, the local information system. This mapping language is part of the *wrapper service* described in the following section on integration services.

2.2 Filtering Processor

Between the component and export layers, a filtering processor is used to define local filters or view schemata. This processor requires an ODMG view mechanism to define what are termed export schemata in [24]. The view mechanism itself is outlined in the following section and described fully in [19]. The requirements of this architecture illustrate three main differences with existing federated architectures. Firstly, it requires a view mechanism based on the ODMG standard in order to *improve* interoperability. Secondly, view definitions must be stored in the schema repository as specified in the ODMG 2.0 [6] and ODMG 3.0 [5] standards. Providing a common interface exists for retrieving and processing view definitions, the Object Query Language can be used to extract views for export, and subsequently play a part in displaying or materializing views. Finally, an object view should not simply be a class, but a subset of the database schema. This requirement states that a view should retain details of abstraction (in the form of inheritance information), associations between objects (in the form of ODMG bi-directional relationships), and behavior, as the schema integration process requires as much semantic information as possible.

2.3 Transport Processor

The Transport Processor is used to move data between local ISs (at the export layer) and the federated database kernel (at the federated layer). In practical terms metadata is moved when export schemata are extracted from local ISs and transferred to the federated kernel; query data is passed from the federated kernel to local ISs; and when requested, data is moved to the federated kernel, or to local ISs which may request data. The Transport Layer contains different implementations of transport processors depending on the required function and the constituent parts which comprise the federation. In *figure 1* there are four possible transport processors identified. Each processor type from the simplest (thin application) through to more complex processors (CORBA), is now explained briefly.

- **ODMG Vendor.** The extraction of view schemata from local ISs to the federated kernel where all parties use the same vendor implementation of ODMG database represents the simplest form of transport processor. This extraction process can migrate a collection of meta-objects (the view definition) between ODMG databases using a basic connection between two homogenous databases. Since only (ODMG standard) meta-objects are moved, there are no issues with regard to the importing of unknown (user-defined) types. View meta-objects are discussed in [18].
- **ODMG OIF.** In theory, the previous processor should suffice, even where different ODMG vendor databases are used across similar platforms. However, the heterogenous nature of different ODMG implementations means that a processor is required to connect to target ODMG systems, convert the source ODMG objects into a standard transportation format (known

as the Object Interchange Format), transport them, and convert from the OIF format back to the ODMG representation in the target system. Each processor will require a 'thin' adaptor for connecting to, reading from, and writing to each vendor implementation.

- **XML.** One of the problems with the previous processor is that it requires a significant amount of proprietary software code and is not really suited to multi-platform federations. Furthermore, the processor is confined solely to moving data between ODMG systems in a rigid and predefined fashion. A processor which can achieve the same function, but can also be used for more complex negotiations between object stores (see [4]) can be implemented using the eXtensible Markup Language (XML). Each ODMG vendor supplies an adaptor or driver which converts their ODMG objects to and from an XML representation. Different negotiation protocols can be used [4] depending on the nature of (transport) transaction.
- **CORBA.** In many cases it will be necessary to transport[2] behavior between the participating ODMG systems, and between the local systems and the federated kernel. None of the previous processors can manage this requirement as the design structure of ODMG databases and applications means that data is stored in the (shareable) database, and behaviour is implemented and stored in (proprietary) client applications. Consequently, a 'generic' application such as a data transport processor cannot access behaviour which is not stored in source databases. The Object Management Group's CORBA technology [13] can be used to extend ODMG databases to make their behaviour shareable to the transport processor.

2.4 Integration Processor

The role of this processor is to integrate imported local schemata to form federated schemata. It uses the *View Service* to define global or federated schemata on top of imported schemata using different forms of joining operations. These integration operations require a view mechanism with the rich set of operators originally specified in [10]. Thus, the same view mechanism discussed earlier must also incorporate complex restructuring operators.

Each of the processors described in this section requires key services to bind appropriate layers in the architecture. For example, the CRUD processor is dependent on a wrapper language to bind its ODMG schema representation with the schema of the local participating information system. In the following section, the integration services which support these processors are described.

3 Integration Services

In the previous section we outlined the integration processors employed in this architecture. An outside service (outside the scope of this research) is used to

[2] The term *transport* is used in a broad sense. In reality, what is required is the sharing of behaviour, as it belongs to a class definition and not to the object instances which are moved between information systems.

construct an ODMG representation of each local database schema. One output from this process will be an Object Definition Language (ODL) schema specification. When this specification is passed through the ODMG databases services (see ODL Database Process in *figure 2*), the database will contain a schema definition (in the form of metaclass instances). This is the standard schema definition step for ODMG databases and to understand the structure and type of metaclasses, please refer to [9]. To design and implement the services required in this architecture it was necessary to extend the ODMG metamodel (and its implementation, the schema repository interface) to include the concept of virtual types and virtual subschemata. Only a brief discussion of the repository extensions will follow shortly, but they are described in detail in [19][21].

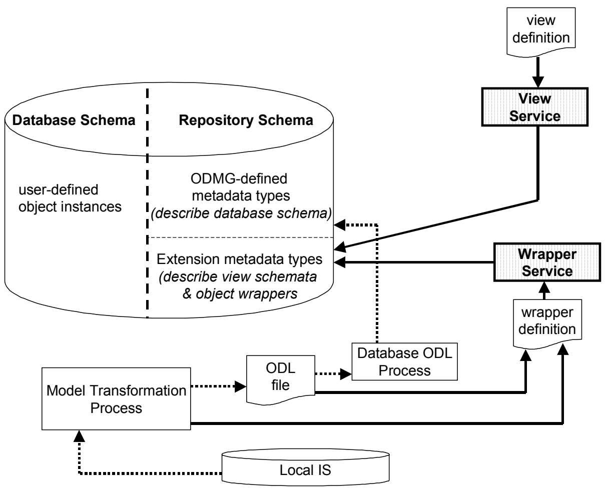

Fig. 2. *Service Deployment and Operations in the ODMG Federated Database Architecture.*

There are two key services which are discussed in the following sections: the *Wrapper Service* which stores wrapper definitions and the *View Service* which stores view schemata, both in the form of ODMG meta-objects in the extended schema repository. In *figure 2* the operations and services which take place at the local information server are illustrated. The broken line indicates operations which are not part of this research. Specifically, the Model Transformation Process is performed manually with the help of CASE tools, and generates both a set of mappings between local and ODMG entities, and an Object Definition Language (ODL) specification of the component (ODMG) schema. Both the mappings and ODL file are used as inputs to the Wrapper Service. A description of the *View* and *Wrapper* services is now provided, with the help of the service deployment diagram in *figure 2*.

3.1 Wrapper Service

Essentially, the Wrapper Service comprises a specification language for mapping ODMG to non-ODMG entities, and a language parser combined with a storage mechanism. A full description of the language can be found in [21], with a brief example provided here. Briefly, a *wrapper* may contain any number of entities specified in the form of a *class*; a *class* has a *name* and is comprised of *attributes* and *relationships*; and a *class* may have *relationships* with other *classes*. In other words, the wrapper language permits an ODMG-style specification[3]. Due to its close association with the ODMG model, the wrapper specification language is known as ODL$_W$, the *Object Definition Language for Wrappers*.

The wrapper language parser processes the wrapper specification and the result is the construction and storage of a collection of meta-objects in the schema repository extension. The Display Service utilizes this metadata when generating view results.

3.2 View Service

Similar to the Wrapper Service, the *View Service* comprises a specification language for defining local and global views, and a language parser combined with a storage mechanism. The view language, ODL$_V$ (*Object Definition Language for Views*) is more complex than ODL$_W$ due to the nature of the operations involved. The purpose of the language is to define subschemata which contain as much semantic information as possible, and to provide an operator set which supports the integration of subschemata. In *figure 2* the view definition (an ODL$_V$ file) is processed by the View Service and stored as a collection of metaclass instances. A standard interface to the schema repository is crucial to both the view and display services. Both services represent a form of *generic*[4] application which must operate on any ODMG database. In ODMG version 2.0 the schema repository interface standard was published which theoretically[5] provides a mechanism for developing generic applications which can generate dynamic queries or construct view classes using existing base classes. During the design of ODL$_V$ it was necessary to extend this specification to include the concept of virtual types. These extensions maintained the structure and naming scheme of the original repository interface for the purpose of interoperability. The full extensions are described in [21], with a brief account available in [18].

Although the view language itself is described in [19], we will now provide a brief overview in order to demonstrate how schema segments can contain more semantic information than simple *class* views, and how these subschemata can

[3] The difference is that the wrapper language does not specify behaviour as it is unclear how a function which has been re-implemented in ODMG can map to, or execute a function in a remote database. This forms part of on-going research.

[4] A *generic* application is assumed to be one which has no prior knowledge of the structure of the database.

[5] Unfortunately, ODMG vendors continue to use proprietary interfaces to schema repositories which provide an obstacle to interoperability among databases.

be integrated. A *view* comprises one or more sub-schemata called *view segments*. A view segment comprises a collection of connected classes, where connections form either hierarchical or association relationships. At the class level, classes may drop or gain attributes or relationships, and at the schema level, classes may drop or gain both hierarchical or association relationships. These operations are typical of local view definitions, but may also take place during the specification of federated view definitions.

In addition to these basic operations, there are also a series of integration operators which are generally used at the federated level, but may also be used for local view definitions. The operators can be divided into two broad categories: *view restructuring* and *view integration* operators. The syntax, grammar and working examples are provided in [21] as space restrictions prevent a more detailed discussion here.

3.3 Display Service

The *Display Service* generates extents for all classes in the view. Each view segment must have a single OQL-style query defined for a *pivotal* view class, that is used to generate extents for all classes in the segment. Consider the view sample in *figure 3* which has three distinct segments: the first is a list of consultants together with patient and duty information; the second is a list of nurses and details of the wards in which they work; and the final segment is a list of all patient visit details called *episodes*, which are specialized as *in-patient* and *out-patient* records. The pivotal classes are *Consultant*, *Nurse* and *Episode* respectively.

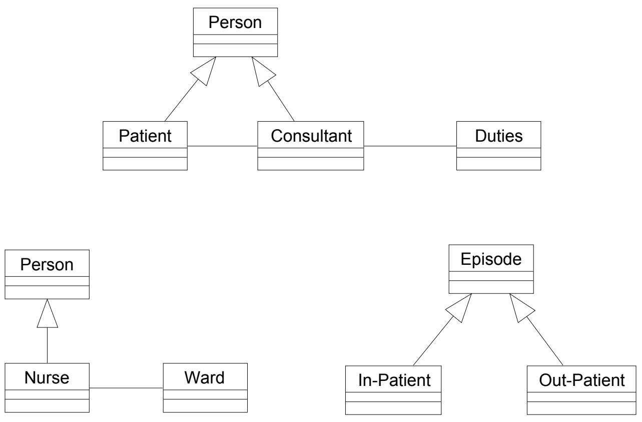

Fig. 3. *Sample healthcare view schema.*

Assume that the extent queries are:

- select C from Consultant where C.Speciality = ‘blood disorders’;
- select N from Nurse where N.shift = ‘nights’;
- select E from Episode where E.date = ‘10.10.99’;

In the first schema segment, the extent for the *Consultant* class will be determined directly by the pivotal query; the extent for the *Patient* class will be selected as those patients who are treated by *Consultants* who have a *speciality* in ‘blood disorders’; the extent for the *Person* class will be the union of the two previous extents; and finally the extent for the *Duty* class will contain only *Duty* objects for *Consultants* with a *speciality* in ‘blood disorders’.

In the second schema segment the extent for the *Nurse* class will contain only nurses who work the ‘night shift’; the extent for the *Person* class will be identical[6]; and the extent for the *Ward* class will contain only those wards in which night-shift nurses work.

In the third schema segment, the extents for the *In-Patient* and *Out-Patient* classes will contain only those *In-Patient Episodes* and *Out-Patient Epsiodes* for ‘10.10.99’ respectively; and the extent for *Episodes* will be the union of the two previous extents.

4 Conclusions and Future Research

In this paper we described our efforts to employ modern standards such as the ODMG database model, CORBA, ODBC and XML to build services to support the integration of heterogenous information systems. In a larger version of this paper [20], implementation details are provided.

- Participating information systems must provide an ODL representation of that segment of the schema they wish to share.
- The architecture supplies a wrapper specification language to bind the ODMG to local schema representation and through the CRUD processor, extract data from local ISs and present the results as ODMG objects.
- An ODMG-style view language creates virtual subschemata.
- Heterogenous processors can move view schemata (and data if requested) between local and federated servers.
- A rich set of operators are present which can manipulate ODMG view schemata.
- A display system has been specified to query and display views.

All of the above have been implemented in a generic fashion so that they can operate with any ODMG database which supports the schema repository interface standard, and has incorporated the schema extensions specified in our earlier research. All of the services described in this publication can be downloaded from the OASIS web site[12], together with instructions for building a

[6] In ODMG databases each Nurse object requires the construction of a {Person, Nurse} pair, where both have the same identifier.

federation of legacy systems with ODBC interfaces. The project is now at a review stage with the focus on improving the current prototype system. Current research is threefold:

- While previous tests have used all features (and operators) specified in the view language, it is felt that as views are (continually) defined on existing views, some combinations of operators may have performance implications. This element of research is focused on emulating the full datasets in six healthcare information systems, generating an exhaustive list of combination views, and identifying problematic view types.
- Since behavior of ODMG classes is not stored in databases, it is impossible to include it in view specifications. However, we have proposed some extensions to the implementation of ODMG databases where behavior resides at the server and not in client software. While this work is ongoing, a description of the revised ODMG architecture, together with a method for sharing has been completed.
- Finally, although our CRUD processors can provide updates, there is no global transaction manager (such as that supported in AQUA) specified. Current efforts involve the classification of participating systems to determine the types of transactions (read-only and different levels of write transactions) which can be supported.

Issues concerning legacy system migration, interoperability among heterogenous software systems, and information sharing are common among software architects and engineers. Where possible, solutions must be based upon agreed standards, with a clear description of all extensions in order that today's integration solutions do not provide tomorrow's interoperability problems.

Acknowledgments. This research was carried out jointly between the Interoperable Systems Group (ISG) at Dublin City University, and the Object Systems Group at Napier University in Edinburgh. The authors wish to acknowledge the support of Robert Byrne, Dalen Kambur and Damir Becarevic in the ISG.

References

1. Alagić S. The ODMG Model: Does it make sense? *Proceedings of OOPSLA '97*, 1997.
2. ANTLR Reference Manual. *http://www.antlr.org/doc/* 1999.
3. Batini C., Lenzerini M. and Navathe S. A comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18:4, December 1986.
4. Byrne R. and Roantree M. An Object Transport Architecture for ODMG Databases. *Proceedings of the 34th International HICSS Conference*, IEEE Press, 2001.
5. Cattel R. et. al. (eds.) (2000). *The Object Data Standard: ODMG 3.0*, Morgan Kaufmann.
6. Cattell R. and Barry D. (eds), *The Object Database Standard: ODMG 2.0.* Morgan Kaufmann, 1997.

7. Dos Santos C., Abiteboul S. and Delobel C. Virtual schemas and bases. *Advances in Database Technology* (EDBT94), pp. 81-94, Springer, 1994.
8. Dobrovnik M. and Eder J. Adding View Support to ODMG-93. *Advances in Databases and Information Systems: Proceedings of the International Workshop of the Moscow ACM SIGMOD Chapter*, ACM Press, 1994.
9. Jordan D. *C++ Object Databases: Programming with the ODMG Standard.* Addison Wesley, 1998.
10. Motro A. Superviews: Visual Integration of Multiple Databases. *IEEE Transactions on Software Engineering*, 13:7, 1987.
11. Nodine M. and Zdonik S. The Impact of Transaction Management in Object-Oriented Multidatabase Views. In Bukres O. and Elmagarmid A. (eds.), *Object-Oriented Multidatabase Systems*, *pp 57-104,* 1996.
12. OASIS Web Site. *http://www.compapp.dcu.ie/˜oasis/.* 2000.
13. Orfali R. and Harkey D. (1998) *Client/Server Programming with Java and CORBA*, Wiley.
14. Pitoura E., Bukhres O. and Elmagarmid A. Object Orientation in Multidatabase Systems, *ACM Computing Surveys*, 27:2, pp 141-195, 1995.
15. Radeke E. Extending ODMG for Federated Database Systems, *Proceedings of DEXA* 1996.
16. Roantree M., Murphy J. and Hasselbring W. The OASIS Multidatabase Prototype. *ACM Sigmod Record*, 28:1, March 1999.
17. Roantree M., Kennedy J., and Barclay P. Providing views and closure for the ODMG object model. *Information and Software Technology*, 41:15, Elsevier Science, 1999.
18. Roantree M., Kennedy J., and Barclay P. Using a Metadata Software Layer in Information Systems Integration. To appear in *13th Conference on Advanced Information Systems Engineering (CAiSE 2001)*, June 2001.
19. Roantree M., Kennedy J., and Barclay P. Constructing View Schemata Using an Extended Object Definition Language. To appear in *6th International IFCIS Conference on Cooperative Information Systems* (CoopIS 2001), Italy, 2001.
20. Roantree M., Kennedy J., and Barclay P. Interoperable Services for Federations of Database Systems. *OASIS Technical Report OAS-10*, November 2000.
21. Roantree M. *Constructing View Schemata Using an Extended Object Definition Language* (PhD Thesis). Napier University, November 2000.
22. Rundensteiner E. Multiview: A Methodology for Supporting Multiple Views in Object-Oriented Databases. *Proceedings on the 18th International Conference on Very Large Databases (VLDB'92)*, pp 187-198, 1992.
23. Saltor F., Castellanos M. and Garcia-Solaco M. Suitability of Data models as Canonical Models for Federated Databases. *ACM SIGMOD Record,* 20:4, 1991.
24. Sheth A. and Larson J. Federated Database Systems for Managing Distributed, Heterogenous, and Autonomous Databases. *ACM Computing Surveys*, 22:3, pp 183-236, ACM Press, 1990.
25. Scholl M., Schek H. and Tresch M. Object Algebra and Views for Multi-Objectbases. In *Distributed Object Management*, Özsu, Dayel & Valdiurez (eds), pp. 353-374, Morgan Kaufmann, 1994.
26. Subieta K. Object-Oriented Standards: Can ODMG OQL be extended to a Programming Language? *Proceedings of the International Symposium on Cooperative Database Systems for Advanced Applications*, pp. 546-555, Japan, 1996.
27. Subieta K. *Remarks on the ODMG Standard.* Technical Report. Kyoto University, March 1996.