Advances in Databases

Carole Goble John Keane (Eds.)

Advances in Databases

13th British National Conference on Databases
BNCOD 13
Manchester, United Kingdom, July 12-14, 1995
Proceedings

Springer

Volume Editors

Carole Goble
Department of Computer Science, University of Manchester
Oxford Road, Manchester M13 9PL, United Kingdom

John Keane
Department of Computation, UMIST
P.O.Box 88, Manchester M60 1QD, United Kingdom

CIP data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Directions in databases : proceedings / 13th British National Conference on Databases, BNCOD 13, Manchester, United Kingdom, July 12 - 14, 1995. Carole Goble ; John Keane (ed.). - Berlin ; Heidelberg ; New York : Springer, 1995
(Lecture notes in computer science ; Vol. 940)
ISBN 3-540-60100-7
NE: Goble, Carole [Hrsg.]; BNCOD <13, 1995, Manchester>; GT

CR Subject Classification (1991): H.2-5

ISBN 3-540-60100-7 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
Printed on acid-free paper SPIN 10486347 06/3142 - 5 4 3 2 1 0

FOREWORD

This volume continues the theme of directions in database research established by the British National Conference on Databases, containing the proceedings of the thirteenth conference (BNCOD 13) held in Manchester, UK in July 1995.

The conference enhanced its record of excellence and internationalism: in all 64 technical paper submissions were received from 18 countries including Australia, Brazil, Korea, New Zealand, Singapore and USA. Each paper received at least three reviews. Of the 64 papers, 14 were accepted for presentation at the conference, together with two internationally respected invited speakers.

The database field is now an established one with regard to conventional applications and structured data types, yet a progressive and exciting one for new applications, such as multimedia, document management, and CAD. These applications demand sophisticated and powerful models, and new operational approaches.

Although the relational model continues to dominate the commercial sector, object-oriented data databases (OODB) are maturing and are increasingly influential in both the commercial and research sectors. This is reflected by the first invited speaker, *Rick Cattell* of SunSoft, USA, and a leader in the field of OODB standardisation. He discusses *object databases and standards*, exploring object database technology and product market directions, comparing object and relational technology, and presenting the ODMG-93 standard for object DBMSs.

One of the chief new application areas that requires the expressivity and sophistication of the OODB is multimedia. The second invited speaker, *Arif Ghafoor* of Purdue University, USA, considers *multimedia database management*. He argues that multimedia database systems will need data models both more powerful and more versatile than the relational model. He suggests that two key requirements for multi-media databases are the process of spatio-temporal modelling and the computational needs for automatic-indexing of spatio-temporal data.

The first group of technical papers continues the exploration of new modelling formalisms through functional database languages. *Courtenage & Poulovassilis* consider extending a functional database language to support subtyping, inheritance and method overloading, whilst *Sutton & Small* discuss the update operations implemented in the functional database programming language PFL and the linear type system which regulates their use.

Not only have database applications become more demanding, but so have database users. The first point of contact for a user is the database interface, and surprisingly little attention has been paid to this essential area. *Nordbotten & Crosby* investigate user understanding of graphic models: their study in graphic data perception indicates that many details are seen by less than half of the readers and that graphic style is an influence. *Haw et al.* analyse the communicative process of enquiry and present GUIDANCE, a system based on ideas derived from the human communicative practices and conventions. Finally *Mitchell et al.* propose a conceptual approach to defining interfaces which uses the features of a fully object-oriented data language

to specify interface objects combined with database objects.

Work remains to be done at the system-level of databases: an area addressed in the third group of papers. *Sieg et al.* describe and analyse query scheduling policies that use knowledge of the number of available system buffers and the various hot points of the queries to provide more efficient processing. *Veenhof et al.* consider the optimisation of n-way spatial joins using filters, showing that a filter sequence can reduce the number of calls to spatial operations. Finally, *Gukal et al.* present a dynamic transient-versioning method which both increases concurrency among transactions and reduces storage overhead.

Concurrency, distributed environments, and data types such as text provide challenges to, and support for, retrieval and transactions. *Hussak & Keane* discuss how transactions that access different types (*tiers*) of data offer greater scope for concurrent execution by allowing the standard serializability condition to be weakened, whilst *Kim et al.* consider the problem of finding an optimal global plan for a tree query in a distributed database, the aim being the minimisation of total processing time. At a different level, *Kaufmann & Schek* describe the realisation of a preprocessor for simple text retrieval on top of a relational database.

The ubiquity and evolutionary growth of databases have altered the database environment to one of federated systems rather than centralised stand-alone systems. *Alzahrani et al.* present a software tool to help resolve conflicts between local integrity specifications in a heterogeneous federated system. Databases that have accumulated, and continue to accumulate, terabytes of data are commonplace; new applications such as satellite information and scientific data collect information on an immense scale. Parallel machines offer some solutions to the scale and performance issues. *Watson & Catlow* discuss the requirements on such machines of commercial database processing, and consider how the ICL GOLDRUSH MegaSERVER meets these requirements. Finally, *Kerridge et al.* present an interface between the relational processing part and the storage system of a parallel machine, the aim being to perform low-level SQL processing as close to the data storage as possible.

Acknowledgements

Many people have assisted in the staging of BNCOD 13: the other members of the Organising Committee – Ferzana Butt, Mary Garvey, Babis Theodoulidis and Brian Warboys – have all made invaluable contributions to the organisation; the members of the Programme Committee who ensured that all papers had at least three referees; the members of the Steering Committee; the authors who responded on time to the deadline; the British Computer Society for their support; and Alfred Hofmann and Springer-Verlag for continued interest in publishing these proceedings in the *Lecture Notes in Computer Science* series. Finally, thanks to our respective departments – Computer Science at the University of Manchester and Computation at UMIST – for use of their facilities, and to all our colleagues.

Manchester, UK, May 1995 *Carole Goble & John Keane*

CONFERENCE COMITTEES

ORGANISING COMMITTEE

Carole Goble	(Chair)	University of Manchester
Ferzana Butt	(Administrator)	University of Manchester
Mary Garvey	(Publicity)	University of Wolverhampton
John Keane	(Proceedings)	UMIST
Babis Theodoulidis	(Exhibition)	UMIST
Brian Warboys		University of Manchester

PROGRAMME COMMITTEE

Brian Warboys	(Chair)	University of Manchester
Andy Bailey		Oracle
Tim Bourne		SIAM Ltd
David Bowers		University of Surrey
Richard Cooper		University of Glasgow
Barry Eaglestone		University of Bradford
Bill Edisbury		TSB Bank
Carole Goble		University of Manchester
Alex Gray		University of Wales, Cardiff
Peter Gray		University of Aberdeen
Frances Grundy		University of Keele
Mike Jackson		University of Wolverhampton
Keith Jeffrey		DRAL
Mike Kay		ICL
John Keane		UMIST
Jessie Kennedy		Napier University
Jon Kerridge		University of Sheffield
Mark Levene		University College London
Rob Lucas		Keylink Computers
Sue Malaika		IBM (UK)
Simon Monk		University of Central Lancashire
Ken Moody		University of Cambridge
Bill Olle		T.William Olle Associates
Norman Paton		Heriot-Watt University
Alex Poulovassilis		Kings College, London
Norman Revell		Middlesex University
Phill Robinson		Sybase
Mike Shave		University of Liverpool
Babis Theodoulidis		UMIST
Sarah Wilkinson		Integrated Computer Technologies Ltd
Geoff Young		NATWEST Bank

STEERING COMMITTEE

Alex Gray	(Chair)	University of Wales, Cardiff
Tim. Bourne		SIAM Ltd
David Bowers		University of Surrey
Carole Goble		University of Manchester
Peter Gray		University of Aberdeen
Mike Jackson		University of Wolverhampton
Mike Worboys		University of Keele

CONTENTS

Queries and Transactions

Parallel and Federated Systems

Object Databases and Standards

R.G.G. Cattell, SunSoft, Inc
1500 Salado Drive, Mountain View, CA 94043 USA

Object DBMSs are an interesting new technology now reaching some degree of maturity. This paper explores object database technology, product market directions, comparisons to object-relational technology, and the ODMG standard for object DBMSs.

1. Introduction

For the purposes of this paper, an object-oriented DBMS, sometimes called an object DBMS, is a DBMS that adds database capability to an existing object-oriented programming language such as C++ or Smalltalk. An object DBMS differs in several ways from the other most popular way to incorporate "object" capabilities in databases, *object-relational DBMSs*, in which SQL-based DBMSs are extended with object programming language capabilities.

In the commercial arena, examples of object DBMSs include GemStone from Servio, O_2 from O_2 Technologies, Objectivity/DB for Objectivity, ObjectStore from Object Design, ONTOS from Ontos Corporation, POET from POET Software, and VERSANT from Versant Object Technology. In contrast, Oracle, Sybase and other major relational vendors are evolving towards object-relational DBMSs, and new start-ups such as Illustra and UniSQL have also introduced products in the object-relational market.[1]

Object DBMSs generally provide the following features:

- An object-oriented data model, including object identifiers, attributes, methods, and type inheritance.
- Integration with an object-oriented programming language, with transparent or semi-transparent fetch and store of objects.
- A declarative query language similar to the ones provided by other DBMSs, usually a SQL derivative.
- Advanced data sharing mechanisms, including long transactions, optimistic concurrency control, multiple versions of data, and private data check-out.

The object DBMS products differ in a variety of specific details. Also, simple object managers, database system generators, semantic/functional DBMSs, and other approaches have been taken in addition to object-relational and object DBMSs. A number of good sources are available comparing the approaches [3,4].

---

[1] Yes, these product names are all trademarks of their respective companies.

2. Contrast of Approaches

There is some debate about the future market directions for object DBMSs, particularly about the likely success of object DBMSs versus object-relational DBMSs, however a fair amount of this debate may simply result from financial investments of the parties involved.

There are certainly compelling arguments for the object-relational approach. Most notably, there is substantial investment in SQL-based relational DBMSs, so an evolutionary approach that preserves that investment is very attractive. Object-relational DBMSs based on existing products such as Oracle or Sybase boast many years investment in the robustness, application development tools, data security, transaction processing performance, and day-to-day business requirements such as online backup.

Nevertheless, there are reasons to believe that there is a substantial market for object DBMSs over the next decade, even as object modeling capabilities are added relational systems:

- Object DBMSs provide a simpler way for programmers familiar with an object-oriented programming language to use databases; it is not necessary to split applications into parts written in the programming language and parts written in the database language (SQL), and it is not necessary to explicitly translate data from the database to programming language data structures and vice versa.
- In addition to supporting conventional DBMS functionality, object DBMSs deal with complex data with substantially higher performance; these benefits stem primarily from client-side caching and seamless integration with the programming language environment [2].
- Projections from market firms such as IDC indicate that the object database market is experiencing revenue growth very closely parallel to that of the early relational industry. As with relational systems a decade earlier, object DBMSs now offer revolutionary advantages that are not likely to be achieved through evolutionary development of their predecessors.

Despite these advantages, it is unlikely that object DBMSs will completely overcome the evolutionary advantages that extensions to relational systems have for business applications in the near future. Thus, I believe object DBMSs and object-relational DBMSs will co-exist for the remainder of this decade. They will likely be used for different kinds of applications, just as different programming languages are popular for different kinds of applications. As an example, SunSoft has incorporated both object DBMS and relational DBMS access into its DOE distributed object environment.

3. ODMG Standard

The importance of a standard for new technology is often underestimated. The success of relational database systems did not result simply from a better level of data independence and a simpler data model than previous systems. Much of their success came from the standardization that they offered. The acceptance of the SQL standard allowed a high degree of portability and interoperability between systems, simplified learning new relational DBMSs, and probably most importantly, represented a wide endorsement of the relational approach.

All of these factors are as important for object DBMSs today as they were for relational systems a decade ago. In fact, these factors are even more important, because most of the products in this area are offered by young companies: portability and endorsement of the approach are essential to a customer investing in applications on these DBMSs. In the case of object DBMSs the scope of the customer's investment is even more far-reaching than with relational DBMSs, because one environment encompasses all of an application's operations and data. A standard is critical to making applications practical.

In addition to the benefits of standardization, the introduction of ODMG-93 has relieved another impediment to the use of object DBMSs, namely the existence of a powerful query language. The OQL language in ODMG-93 is more powerful than SQL, as we shall see, while the query languages in many of the object DBMS products had been substantially weaker than SQL, or non-existent. Object DBMS vendors committed to much more query capability in conjunction with ODMG-93.

In 1993 the Object Database Management Group (ODMG) defined a standards specification, ODMG-93, designed for object DBMSs [1, 5]. Consistent with our definition of object DBMSs, the ODMG architecture includes bindings to provide transparent persistence and database capability in object-oriented programming languages, specifically C++ and Smalltalk.

On the object-relational front, the ANSI X3H2 (SQL) group has been working on SQL3, which is not yet to "draft standard" stage this year. The thrust of the SQL3 work is aimed at extending the SQL type system and adding procedural capability to the SQL language. There is little overlap between the ODMG and SQL3 work, except in the query language portion (Chapter 4 of the ODMG-93 specification, and the SELECT statement of SQL3).

The ODMG and X3H2 groups have had several ad hoc meetings to decrease differences in the query language syntax and semantics in their respective standards. Two actions have resulted from these meetings:

1. The ODMG OQL language has been revised to make it as compatible as possible with SQL2 (with ODMG object extensions).
2. Change proposals are being made to X3H2 to reduce differences in the SQL3 and OQL object extensions. The most fundamental remaining difference is that

SQL3 still treats tables as the only "top level" type, while OQL treats all types as equivalent (the result of a query or a named top level entity can be of any type).

ODMG was founded because no progress had been made towards standards for object DBMSs several years after their successful deployment. OMG had formed a database special interest group and had begun work toward database-related standards in the Object Services Task Force, ANSI had formed an Object-Oriented Database Task Force which resulted in ANSI X3H7, and various ad hoc attempts were made between vendors, but nothing resulted in standards for object DBMS products.

ODMG was formed in late 1991, at my invitation to a meeting at SunSoft. The ODMG work was done by a small group of five vendor employees who committed one week per month to the ODMG work over two years. As a result, the work progressed very quickly compared to traditional standards groups. A first draft for all the major components was produced during 1992, and the accepted version was published in 1993. The intense ODMG effort gave the object database industry a "jump start" toward standards that would otherwise have taken many years.

Since the introduction of the standard, the ODMG group has expanded significantly; it now includes Object Design, Objectivity, Ontos, O$_2$ Technologies, POET Software, Servio, Versant, American Management Systems, Anderson Consulting, EDS, Fujitsu, Hewlett-Packard, Intellitic, MITRE, Persistence Software, Sybase, UniData, and Texas Instruments. The voting members of ODMG (the first seven in the list) are committed to support the ODMG-93 standard this year, and as of this writing several of the vendors have already released partial implementations. Thus, ODMG-93 is likely to become a de facto standard for the object DBMS industry.

There are some lessons to be learned about technology and the standards process from the ODMG history. It is very difficult to do substantial creative work within a large standards group, given the number of people and the politics involved. It is generally necessary to choose a de facto standard as a starting point, and then make incremental modifications. Unfortunately, unlike SQL in the relational DBMS world, no accepted starting point existed for object DBMSs. Instead, the ODMG work is derived by creatively combining the strongest components of a number of products currently available. These products provided demonstrated implementations of the standards components that had been tried in the field.

4. ODMG Architecture

ODMG defines a common architecture for object DBMS products. The programmer writes declarations for the application schema (both data and operations) plus a source program for the application implementation.

The source program is written in a programming language such as C++, which has been extended to provide a full database manipulation language including transactions and object query. The schema declarations may be written in an extension of the programming language syntax, called the programming language

ODL (object definition language), or may be written in a programming language-independent ODL that ODMG defines. The latter ODL might be used as a higher-level design language, or to allow schema definition independent of programming language.

The programmer's declarations and source program are compiled and linked with the DBMS runtime to produce the running application. The application accesses a new or existing database, whose types must conform to the declarations. Databases may be shared with other applications on a network; the DBMS provides a shared service for transaction and lock management, allowing data to be cached in the application.

The chapters of the ODMG-93 specification correspond to the main components of the standard:

- Architecture: The first chapter defines a common architecture for an object DBMS as just described. This agreement on the architecture and approach to object DBMSs was essential to making this work possible.
- Object Model: ODMG defines a common data model to be supported by object DBMSs. A subset of the object model provides interoperability across programming languages, e.g. allowing the same database to be shared by a C++ and Smalltalk program.
- Object Definition Language: ODMG defines an object definition language (ODL) as a syntax for the object model. ODL may be used to define an application schema; the schema can subsequently be translated into declarations in the desired programming language.
- Object Query Language: ODMG defines a declarative object query language (OQL) for querying database objects. OQL can be used by end-users or from within a programming language. OQL is based on SQL syntax wherever possible.
- C++ and Smalltalk Bindings: The remaining chapters of the ODMG specification define programming language bindings, also known as the object manipulation language (OML). Currently bindings have been defined for C++ and Smalltalk. OML binding chapters for SQL3, C, LISP, and IDL are being considered.

5. Object Model and Definition Language

Much of the ODMG work is based on Object Management Group (OMG) specifications [6]. In particular, the ODMG object model is designed as a superset of the OMG object model.

The ODMG model is based on objects, with object identifiers. Objects can be categorized into types. All objects of a given type exhibit common behavior and a common range of states. The behavior of objects is defined by a set of operations

that can be executed on an object of the type. The state of objects is defined by the values they carry for a set of properties. These properties may be either attributes of the object itself or relationships between the object and one or more other objects.

As with variables in programming languages, human-meaningful names may be given to ODMG objects. A name must refer uniquely to a single object within the scope of the definition of the name; currently the only name scope defined is a database. Note that these names differ from primary keys in a relational DBMS; they are more like relation names, except that they can refer to objects of any type (not just tables).

Operation signatures define the operations that objects of a given type support. As in most programming languages, each signature defines the name of the operation, the name and type of any arguments, the name and type of any returned values, and the names of any exceptions (error conditions) the operation can raise.

Attributes of object types are similarly specified with attribute signatures. Each signature defines the name of the attribute and the type of its legal values. Attributes take literals as values, e.g., strings, numbers, etc.

Relationship signatures specify the relationships in which objects of a given type can participate. Each signature defines the type of the other object or set of objects involved in the relationship and the name of a traversal function (an *inverse attribute*) used to refer to the related object or set of objects. Relationships are binary and are defined between two types of objects (as opposed to attributes that are defined between an object and a literal). The cardinality of the relationship can be one-to-one, one-to-many, and many-to-many.

ODMG defines a number of collection types: sets, bags, lists, and arrays. Named instances of these types can be used to group objects; for example, hourly_employees might be the employees who are paid by the hour. Thus, there may be many pre-defined collections of each type, not just the extent of the type (e.g., the employee table).

ODMG supports inheritance between types in a subtype/supertype graph. All of the attributes, relationships, and operations defined on a supertype are inherited by a subtype. The subtype may add additional properties and operations to introduce behavior or state unique to instances of a subtype. Multiple inheritance is supported.

The extent of a type can automatically be maintained by the object DBMS, as in relational DBMSs. The type definer can request that the system automatically maintain an index to the members of this set. Keys can also be defined on type extents, in which case the DBMS guarantees the uniqueness of the key attributes within the type extent.

ODMG defines an object definition language (ODL) that is the syntax for the object model. ODL is intended to define object types that can be implemented in a variety of programming languages; it is not tied to the syntax or semantics of one programming language.

There are a number of benefits to having a programming language-independent ODL. ODL allows the same database to be shared across multiple programming languages, and allows an application to be ported to a new programming language without rewriting the data schema description. ODL can also be used by design and analysis tools that must describe an application's data and operations independently of programming language. The resulting design can then be used directly or translated into a data description language of the programmer's choice. Also, a schema specified in ODL will be supported by any ODMG-compliant DBMS.

In addition to the programming language-independent ODL, the ODMG programming language bindings (currently C++ and Smalltalk) describe optional ODL syntaxes designed to fit smoothly into the declarative syntax of their host programming language. Due to the differences inherent in the object models native to these programming languages, it is not always possible to achieve 100% consistent semantics across the programming-language specific versions of ODL. ODMG's goal has been to maximize database schema portability across programming languages. In the ODMG specification, each programming language binding documents any extensions or shortfalls with respect to the common ODMG model.

ODL's syntax is based on OMG's Interface Definition Language (IDL) developed as part of the Common Object Request Broker Architecture (CORBA). ODMG used IDL rather than invent yet another language syntax. ODL adds to IDL the constructs required to specify the complete semantics of the ODMG object model, in particular, referential integrity and collections. An example ODL definition looks something like this:

```
interface Professor: Employee {
    extent professors;
    attribute String office_number;
    attribute enum rank {full, associate, assistant};
    relationship Set<Course> teaches inverse is_taught_by;
    grant_tenure() raises (ineligible);
};
```

This declaration defines professors to be a subtype of employee with a named extent, two attributes, a relationship to courses named teaches (the inverse attribute of courses is named is_taught_by), and an operation to grant tenure.

6. Object Query Language

There are two ways to retrieve objects out of an ODMG database:

1. Objects may be retrieved implicitly by navigating relationships in OML (the programming language), or

2. Objects may be retrieved through the ODMG object query language (OQL), by identifying objects through predicates defined on their characteristics.

Simple OQL queries are based on a predicate applied to a collection, selecting the members of a collection that satisfy the predicate. However, more complex queries may be performed: OQL can perform the equivalent of relational joins, and more. OQL need not produce an object or a table — it may result in an integer, a list of object references, a structure, a set of sets of real numbers, or any data structure that can be defined in the ODMG object model.

OQL queries objects starting with their names, which act as entry points into a database. In this sense, OQL is like SQL; however, in OQL a name may denote any kind of object: atomic, structure, collection, or literal. As an embedded language, OQL allows you to query objects which are supported by the native language through expressions yielding atoms, structures, collections, and literals.

The ODMG object model is used as the basis for OQL. The semantics of OQL are formally defined in the ODMG specification.

OQL differs from SQL2 and the planned SQL3 in some important respects. In keeping with the object paradigm of encapsulation, OQL does not provide explicit update operators; it relies on methods defined on objects for this purpose. Also, OQL is a declarative query language; it contains no procedural operations as in SQL3. OQL can be easily optimized by virtue of its declarative nature.

OQL's syntax is based on SQL, because of the prevalence of this language in the DBMS world. However, ODMG did not feel constrained to SQL syntax or semantics in cases where it would compromise the simplicity or power of OQL. Other concrete OQL syntaxes will be defined in order to merge the query language into programming languages. For example, ODMG plans to define a syntax for preprocessed C++ that is a natural extension of the language as opposed to an embedded foreign syntax.

OQL provides high-level primitives to deal with collections of objects, but OQL is not exclusively centered on the set construct as is SQL. OQL provides primitives to deal with structures and lists, and treats all such constructs with the same efficiency and convenience. As an illustration of the scope and nature of OQL, the following are valid queries:

```
2+3

president.subordinates

list ( joe, harry ) union
    select x from x in employees where x.salary > 50000

exists x in professors : x.spouse in x.advisees
```

7. Programming Language Bindings

A "programming language binding" in ODMG is quite different than in SQL. The ODMG binding is based on extending a programming language's syntax and semantics in order to provide database capabilities rather than embedding statements in SQL or another language. It is the goal of an ODMG programming language binding that the programmer feel there is one language, not two separate languages with arbitrary boundaries between them.

ODMG's goal of programming language integration results in several general principles. There is a single unified type system across the programming language and the database; individual instances of these common types can be persistent or transient. The programming language-specific binding respects the syntax and semantics of the base programming language into which it is being inserted. The binding is structured as a small set of additions to the base programming language; it does not introduce sublanguage-specific constructions that duplicate functionality already present within the base language. Expressions in the OML and OQL can be composed freely with expressions from the base programming language and vice versa.

An ODMG programming language binding has three components: OML, ODL, and OQL. In the C++ binding, ODL is expressed as a class library and an extension to the standard C++ class definition grammar. The class library provides classes and functions to implement the concepts defined in the ODMG object model. The OML, or object manipulation language, is used for retrieving and operating upon objects from the database. The C++ OML syntax and semantics are those of standard C++ in the context of the standard class library. The C++ OQL provides a way to retrieve data based on predicates.

In the case of C++, a completely seamless interface between the programming language and the ODL, OML, and OQL technically requires a preprocessor, due to constraints of the current C++ language. Because some customers dislike a preprocessor, ODMG chose to limit the ODMG-93 standard in some minor ways. ODMG-93 requires only an ODL preprocessor. A nearly seamless OML solution is possible without C++ changes, and a short-term OQL solution is possible with procedure calls. This allows vendors to get a standard API out quickly to customers, and to get some experience with it.

In addition to the query facilities and data modelling extensions such as collections and inverse attributes, ODMG extends programming language operations with database and transaction operations. All access, creation, modification, and deletion of persistent objects must be done within a transaction on an open database. Operations are defined on pre-defined database and transaction types. A database application generally will begin by opening a database and accessing one or more named objects, proceeding from there. These objects are in some sense "root" objects, in that they lead to interconnected webs of other objects.