AN OBJECT MODEL OF DATA, BASED ON THE ODMG INDUSTRY STANDARD FOR DATABASE APPLICATIONS

K R Sujithan*

Abstract
The computational requirements of modern information systems, coupled with the inherent limitations of the relational databases, are stimulating research into post-relational database systems. The recent advances in object databases resulting from the integration of object-oriented programming capabilities with database capabilities, seem highly promising. The Object Database Management Group (ODMG) as part of the Object Management Group (OMG) has proposed a de facto standard known as ODMG-93 for object databases, which is rapidly gaining acceptance. In this paper, we introduce the relevant aspects of the object technology within the OMG framework, their Common Object Request Broker Architecture (CORBA), and discuss how this technology integrates with, and extends the client-server model. We survey the main elements of ODMG data model, particularly the datatypes supported, and introduce the elements of both, ODL (Object Definition Language) for defining the structure of the database, and OQL (Object Query Language) for defining queries over the database. Finally, we highlight some results from our work on a calculus for the declarative representation of the OQL queries, suitable for an efficient implementation.

1 Introduction
Since the early days of the digital computer, the computational requirements have changed from the execution of arithmetic operations, to processing simple data structures (e.g. arrays or records), to simulating highly complex objects that represent real world concepts or entities. These applications are characterised by the requirement for the creation, classification, storage, retrieval and dissemination of very large collections of complex structured objects. In addition to the mainstream commercial distributed databases and transaction processing systems, a wide range of examples include VLSI Design, CAD Databases, Protein Structure Modelling and Software Engineering Databases (i.e. databases that manage multiple versions of large programs).

The seemingly insatiable end-user demand for improved throughput (e.g. database transactions per second) for such systems, led to the natural division of services provided by database servers and client computers, taking into consideration the relative strength of each resource (e.g. processing power, storage capacity). This client-server organisation is essentially a combination of three major technologies: Databases, Networks, and Client Interfaces (usually Graphical User Interfaces). Since the database server automatically handles many of the difficult and complex data processing tasks, including concurrency control (locking) and transaction processing, the application’s responsibility is to issue requests for data and to handle returning data.

The ability to manage collections of persistent data, and the ability to access the data efficiently, are two fundamental qualities expected of the database server. These fundamental qualities need to be supported by at least one data model, defined as a mathematical formalism with a notation for describing the data, and a set of operations used to manipulate the data [1]. The data model, in essence, provides an abstract view of data, and allows the users to see information in a way that would be meaningful to them. The data model is made available via interfaces provided by database languages that are embedded into common programming languages (such as COBOL, PL/1 or C). Data Definition Languages (DDL) allow the programming

*Author’s address: Programming Research Group, Oxford University Computing Laboratory, Wolfson Building, Parks Road, Oxford, OX1 3QD. Email: Ronald.Sujithan@comlab.ox.ac.uk
of the structure of the database, and Data Manipulation or Query Languages (DML) allow the high-level expression of queries over the stored data objects.

The relational model, originally proposed by E.F. Codd in 1970 [2], with a firm basis in relational calculus and classical first-order logic, provided the foundation for database research and development for the past 25 years. The elegance and simplicity of the "tables" (i.e. relations) structure of this model, the very reasons underlying its success, are now considered to be the limitation for the class of applications outlined earlier [3]. For instance, the simple but limited set of query operations leads to the so called "transitive closure" problem [4], and the separation of database languages (DDL and DML) from the host language leads to the so called "impedance mismatch" problem [5], studied in detail elsewhere.

The recent advances in object database, resulting from the integration of object-oriented programming capabilities with database capabilities, as a promising alternative has stimulated a very high level of interest within the database community [6]. Object-oriented programming is a disciplined programming style that incorporates four main Software Engineering principles: abstraction, encapsulation, inheritance and polymorphism. Adding a rich set of type constructors, and a persistent storage mechanism, to an object-oriented language produces a database system that is superior (in terms of overcoming the limitations outlined previously) to the relational database [5]. The basic idea of an object database is to view the system as a collection of identifiable, interacting objects, and offers a radically different way to analyze, design and implement database applications.

Fundamental to this approach is the concept of an object model, and until recently, the variety of research activities resulted in a diversity of techniques and the lack of a "standard" model was considered an impediment. In an attempt to remedy this situation, the Object Database Management Group (ODMG), as part of the OMG Organisation – an industrial consortium established to promote object technology – has proposed an industry standard for object databases1 [7]. Currently, ODMG is the closest to a convergence of various object database techniques2, which incorporates all major facilities from existing object database products. In this paper, we first introduce the relevant aspects of the object technology within the OMG framework, their Common Object Request Broker Architecture (CORBA), and discuss how this technology integrates with, and extends the client-server model [9].

The ODMG-93 standard defines an object model of data, object definition (ODL) and query languages (OQL), and bindings for common object-oriented programming languages (e.g. C++, Smalltalk). We survey the main elements of the data model, particularly the data types supported - the generalised collections data type and the specialised data types sets, bags, lists and arrays - for grouping the database objects. Using a simple example we show how these predefined types can be composed to produce complex object structures. We also introduce the elements of both, ODL for defining the structure (i.e. schemas) of the database, and OQL for defining queries over the database.

Finally, we highlight some results from our work on a calculus for the declarative representation of the OQL queries (based on the work done by [10, 11]) suitable for an efficient implementation. The datatypes are coded as monoids (an algebraic structure) and the queries are represented as monoid comprehensions, and query processing is captured as monoid homomorphisms. This framework, which we refer to as object calculus, generalises the relational calculus.

2 The Object Model

This section provides a necessarily brief introduction to some fundamental concepts of object technology, that are useful to understand the ODMG standard described in the sequel. Detailed accounts can be found in [12, 13, 6] and the references provided therein.

1Essentially, all major database vendors have committed to supporting this model, in one form or another, their in future products.

2In parallel to the ODMG-93 development, the ANSI group X3H2 is working on an object database extension to the SQL2 standard (SQL3, or informally, "SQL with Objects"). As [8] note, ODMG-93 represents more concrete progress than the "currently confused state of SQL3", and provides a good specification of aggregate datatypes and facilities for handling them.
2.1 Objects and Object Identifiers

An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain [12]. An object is described in terms of its state, behaviour and identity. The State of an object is defined by the properties (or attributes) of the object and the current values of each of these properties. The Behaviour of an object is the description of how the object acts and reacts in its environment. Behaviour is based on accepting stimuli from the environment and producing internal state changes and external responses to the environment. Identity is the inherent property of an object that distinguishes it from all other objects in the environment.

The identity of objects are implemented as Object Identifiers (OIDs), that are similar to "keys" in the relational scheme since they provide a unique ID for each object, yet different from keys since users cannot directly access it. OIDs are implemented by the system and their existence is transparent to the language interface. Logically, the language interface sees the object by a user-declared name, and access is through the interface, in terms of the signatures defined for that object.

2.2 Classification and Association

Classification is the process of generalising a set of objects according to some common attributes or common behaviour patterns, and a type (or a class) is the general term denoting the generalised concept. In practice, the real world entities are identified and represented as objects, and these objects are grouped into types based upon common state structures or common behaviour. Then, low-level types (or subtypes) can be grouped into higher-level types (or supertypes), using inheritance relationships. In the single inheritance case, the types form a strictly hierarchical or tree structure. In the multiple inheritance case, the types form a rooted and connected directed acyclic graph (DAG), sometimes called a type lattice.

Links and associations are the means for establishing relationships among objects and types, and are orthogonal to the inheritance relationship. A link is a physical (or conceptual) connection between object instances (mathematically a link is a tuple, that is an ordered list of object instances). An association describes a group of links with common structure and common semantics, and all links in an association connect objects from the same type. Sometimes it is useful to model association as a type, in which case, each link becomes one instance of the association type, and links can participate in association with other objects or links themselves can be subject to operations [13].

In practice the system is decomposed into groups of concurrent objects, as clients and servers, and communication and synchronisation are achieved by explicit message exchanges between clients and servers. Objects address other objects by means of a reference (or capability), that defines the permission of the invoking object with respect to the invoked object, and at systems level, OIDs allow the implementation of references between objects.

3 The OMG Framework

The Object Management Group (OMG) is a non-profit international software industry consortium which aims to promote an object-oriented approach to Software Engineering, and in particular to develop common models and interfaces for the design and use of large-scale distributed applications using object-oriented technology. OMG's basic framework is called the Common Object Request Broker Architecture (CORBA) that contains three main components: an Object Request Broker (ORB), Object Services and Object Facilities [9].

This sophisticated CORBA base provides much more that a conventional client-server, as described in the next section. Figure 1 shows the components of this architecture. The Object Services, which will provide the main functions for realising basic object functionality using the ORB - the logical modelling and physical storage of the system objects. The Common Facilities will comprise functions which are useful in many application domains and will be made available through OMG compliant class interfaces.
3.1 The Object Request Broker

The Object Request Broker, the key inter-object communications element, handles the distribution of requests (i.e. messages) between application objects in a system independent manner. The ORB provides the mechanisms by which objects transparently make requests and receive responses, and seamlessly interconnect multiple object systems. A request mainly consists of (1) target object reference, (2) operation to be performed, (3) input/output parameters, and (4) optional request context. The object interface definitions are manifested as objects in the interface repository, as client stubs and server skeletons. Each client stub represents an object operation (a possible request) which a client invokes in a language independent manner (e.g. by calling a subroutine which represents the operation). Each server skeleton provides the interface through which a method receives a request (without any knowledge of the invocation approach).

Alternatively, using the dynamic invocation interface, a client may construct and invoke request objects which can represent any object operation, and the major feature of this form of request is that, they appear as objects themselves (i.e. requests become “first-class objects”). The dynamic invocation requests allow clients to dynamically discover objects, object interfaces, create requests, invoke requests and receive responses.

In summary, clients perform requests using object references, through the static object interface stubs, or the dynamic invocation interface. Clients may access general ORB services via the object adaptor, which handles the vagaries of object implementation schemes. The basic object adaptor is intended for an implementation that separates the client programs from the ORB services and server programs that provide ORB services such as method invocation, authentication, implementation registration, activation and deactivation. Specialised adaptors, such as the object database adaptor, provide services tuned for specific applications and integrate with the generic ORB services.

4 The ODMG-93 Standard

The first release of this proposed standard, ODMG-93, Release 1.1, contains the following components: (1) Object Model, (2) An Object Data Language, (3) An Object Query Language, (4) C++ (and Smalltalk) Language Bindings [7]. ODMG is mainly an interface standard, designed to work with the CORBA specification. The Object Definition Language part allows the definition of database schemas and the Object Query Language (OQL) part is a declarative language for querying and updating database objects. Language Bindings provide specific common programming languages (C++ and Smalltalk) to enable query processing from these languages – collectively called an Object Manipulation language (OML).

The ODMG object model provides a rich set of type constructors for defining both immutable objects (literals) and mutable objects (objects, as described in section 2.1), and both can be either atomic or structured. Application entities for most parts be modelled using either literals or object types, defined in the model. In addition, mutable and immutable objects can be freely mixed to produce complex object struc-
tures. The main component of the object model is the type structure supported, which is described next.

4.1 Type Structure

An ODMG built-in datatype has an interface and one or more implementation, where the interface defines the external behaviour supported by the instance of the type and an implementation defines data structures that support the external behaviour. The set of all instances of a given type is termed its extent, which represents all current instantiations of this type in the database. In some cases the individual instances of a type can be uniquely identified using the keys declaration.

Each datatype is declared in terms of type signatures. Operation signatures define the name, the type of arguments and the type of returned values. Attribute signatures define the name of the attributes and the type of its legal values. Relationship signatures define the relationships in which objects (i.e. instances of this type) participate, thus, it is defined between two objects.

The ODMG built-in basic type hierarchy has two components (1) characteristics, and (2) denotable objects. Characteristics refer to the operations and the properties of an object collectively, where properties are given by the attributes (internal states) and relationships (external links) of that object. Figure 2 shows the type hierarchy for Object Characteristics.

![Figure 2: Characteristics](image)

The hierarchy of object types is rooted as the type Denotable.Object, and there are two orthogonal lines along which the set of denotable objects can be decomposed (1) mutable (object) versus immutable (literal), and (2) atomic versus structured. All denotable objects have identity (OID), however, the representation for objects is different from literals. Figure 3 shows the Denotable Object type hierarchy.

Instances of type Object are mutable (i.e. values of their attributes may change, and the relationships in which they participate may change), and represents objects in the usual sense\(^3\). Literals are objects whose instances are immutable, and are conceptually similar to the datatypes found in the conventional programming languages. Two types of literals are defined by the model, the simplest is the Atomic.literal that are numbers (Integer, Float), characters (Character) and boolean (Boolean). The type Structured.Literal has two subtypes, Immutable.Collection and Immutable.Structure, and figure 3 shows these two built-in subtypes. Note that Date, Time, Timestamp and Interval are defined as in the ANSI SQL specification.

The type Structured.Object has two subtypes, Structure and Collection, and offer ways of defining aggregate datatypes. Structures (or tuples) have a fixed number of named slots, each of which contains an object or a literal. Collections by contrast, contain an arbitrary number of elements, do not have named slots, and contain elements that are all instances of the same type. Insertion of elements is based on either absolute position within the collection, or at a point established by the cursor. Retrieval is based on either absolute position, cursor-relative position or a predicate that uniquely selects an element from the collection based on the value(s) that object carries for one or more of its properties.

The abstractions available in the collection type are essentially Abstract Data Types (ADTs) with support for inheritance (i.e. sub-typing). The resulting substructure contains Sets, Bags, Lists, Arrays, as the elementary collection types. Structured objects may be freely composed to model application data, for example sets of structures, structures of sets, lists of structures etc. are all possible. Since ODMG supports the set structure, the basis of Relational Databases, along with other commonly used abstract types, ODMG is essentially a generalisation of the relational database.

---

\(^3\)The Object type represents the dynamic objects found in the object-oriented programming languages (see section 2.1). This confusing ODMG terminology – using the term “object” for a type as well as its instances – is rather unfortunate.
4.1.1 Collections

A collection is defined as an abstract object that groups other objects, and all of the elements of a collection must be of the same type. Collections may be defined over any instanciable subtype of Denotable.Object, and the model supports both ordered and unordered collections (e.g. based on the sequence of objects present), and collections with or without duplication of elements. Individual collections are instances of a parameterised abstract collection type. The ODMG Object Model defines a corresponding standard set of built-in collection type generators: Set<T> (unordered collections that do not allow duplicates), Bag<T> (unordered collections that allow duplicates), List<T> (ordered collections that allow duplicates), Array<T> (one dimensional arrays), are all subtypes of the type generator Collection<T>, where T is the element type.

For example, type generator for List<T> can be instantiated to produce List<Account> by supplying the element type Account. The model defines several abstract operations over the generalised collection types, and the specific instances specialise these operations to provide the required semantics. The standard document describes these types and operations in detail [7].

4.2 Object Definition Language

The Object Definition Language is a specification language for defining object types and the interfaces to object types that conform to the ODMG object model (i.e. the database schemas). The primary objective of ODL is to facilitate portability of database schemas across a range of conforming databases. ODL is designed to be programming-language independent and supports all semantic constructs of the ODMG model (ODL is essentially a DDL, as described in section 1, for object databases). ODL provides a degree of insulation for applications against the variations in programming languages and the underlying database products. Details of the abstract and concrete syntax of ODL can be found in the standard document.
4.3 Object Query Language

The Object Query Language supports the ODMG data model, and capable of dealing with complex objects and collections. OQL allows to query denotable objects starting from their names, where a name may denote any atomic, structure, collection, or literal objects, and acts as entry points into the database. OQL provides high-level primitives to process collection constructs (i.e. sets, bags, lists and arrays). Further, OQL provides declarative access to objects, in a way similar to the relational calculus, and a query consists of a set of query definition expressions followed by the query specification (in terms of predicates defined over the data elements). Details of the abstract and concrete syntax of OQL can be found in the standard document.

An OQL query is a function which, when applied to this input, delivers an object whose type may be inferred from the operator contributing to the query expression. A Query Evaluator is that part of the database system which processes high-level queries to produce an execution plan. The execution of queries involve calling the operators that manipulate the stored data and other system calls, to retrieve the queried object from the database.

5 A Calculus for Query Processing

In this section, we highlight some results from our work on the efficient implementation of OQL queries. Hopefully, this will indicate possible design choices for prospective implementors as well as expose some work currently in progress. The objective is to develop a mathematical framework for the expression and manipulation of OQL queries, that can lead to efficient implementations. The primary focus is the bulk manipulation of collection types, and implementations on shared-memory and distributed memory parallel machines.

5.1 Elements of the Calculus

Our calculus is based on monoids, a general template for datatypes. Consider how lists and sets are constructed, for example. Constructing a list \([1] \oplus [2] \oplus [3] = [1, 2, 3]\), and constructing a set \(\{a\} \cup \{b\} \cup \{c\} = \{a, b, c\}\), where \([\ ]\) (empty list) and \(\{}\) (empty set) are identity elements, and \(\oplus\) and \(\cup\) are associative operators. Using terminology from abstract algebra \((\text{list}, [\ ], \oplus)\) and \((\text{set}, \{\}, \cup)\) are monoids.

Primitive types such as integer and boolean can be represented as primitive monoid, for example \((\text{int}, 0, +)\) and \((\text{int}, 1, \times)\) are integer monoids, and \((\text{bool}, F, \wedge)\) and \((\text{bool}, T, \vee)\) are all boolean monoids. Each collection monoid has unit function \(f\) that takes an element of some type as input and constructs a singleton value of the collection type. Since all datatypes are coded as monoids, a query is a map from some monoid to another monoid. These maps are termed monoid homomorphisms. For example a \(\text{list} \rightarrow \text{set}\) mapping is given as \(\text{hom}_{\text{list} \rightarrow \text{set}}(f)A\) which returns a set of elements \(f(x)\) where each \(x\) is drawn from the list \(A\) (the function \(f\) is referred as the map function). In this calculus comprehension is the only form of bulk manipulation of collections, and the above transformation in comprehension syntax is \(\text{set}\ \{f(x) \mid x \leftarrow A\}\).

The OQL expressions have direct translation into the monoid calculus. The main OQL statement has a \select\-from\-where format (similar to the SQL \select\ statement) such as,

\[
\text{select } e \text{ from } (x_1 \text{ in } e_1) \ldots (x_n \text{ in } e_n) \text{ where } \text{pred};
\]

which directly translates to the following equivalent comprehension in our calculus,

\[
\text{set } \{e \mid x_1 \leftarrow e_1 \ldots , x_n \leftarrow e_n; \ \text{pred}\}\}
\]

Thus, OQL expressions can be coded as monoids and monoid homomorphisms and query processing can be implemented as transformations from one type of monoid that represents an OQL statement, to another form of monoid suitable for efficient execution by the query evaluator. Defining suitable “normal forms” and designing normalisation algorithms for this formal framework is currently being researched [10].
5.2 Implementation considerations

Our approach to efficient implementation of OQL expressions is based on the observation that comprehensions are also considered as parallel programming constructs [14]. Statement comprehensions are sufficiently abstract to allow the programmer or compiler to perform the various transformations to convert statements into a form mathematically equivalent but computationally more efficient. The GPL language is a parallel programming language that makes use of this form of constructs extensively and produces architecture-independent parallel code [15] (for the Oxford BSP library). The level of abstraction provided by GPL matches closely to the OQL constructs, and a prototype parallel OQL translator based on the GPL language is currently being developed by the author at Oxford.

6 Summary

In this paper, after highlighting the limitations of relational databases, we introduced the concepts of object databases, the OMG CORBA framework, and the proposed ODMG industry standard. The ORB base provides a sophisticated interface for building object-oriented, client-server systems. After discussing the type structure of the ODMG object model, we also introduced the interface languages ODL and OQL for manipulating the object databases. Finally, we highlighted some implementation issues based on our work for a calculus for query processing and efficient parallel implementations. We believe these techniques show promise for future database systems.

References


© 1995 The Institution of Electrical Engineers.
Printed and published by the IEE, Savoy Place, London WC2R 0BL, UK.