Observations on the ODMG–93 Proposal for an Object–Oriented Database Language

Won Kim

UniSQL, Inc.
9390 Research Blvd.
Austin, Texas 78759

1. What It Is and What It Is Not

Although it has many problems, ODMG–93 is an important and positive contribution as a starting point in industry–wide efforts to define a standard object–oriented database language.

The ODMG–93 specification may be summarized simply as follows:

- Despite its hasty claim, it is NOT a "standard". Rather, it is a work–in–progress proposal for an object–oriented database language and language bindings to it for C++ and Smalltalk. ODMG (Object–Oriented Database Management Group) is not a formal "standards" body. It is a committee formed by five vendors of first–generation object–oriented database systems (OODB). (For several years, some of these vendors have offered products that are not much more than persistent storage managers for object–oriented programming languages, but the misleading label "Object–Oriented Database System" has been stuck on such products in the market.) And now the misleading label "standard" seems to be attached to the ODMG–93 work–in–progress proposals.
- It espouses a database architecture that consists of a database management system that supports an object–oriented database language, and language binding layers on top of it for specific object–oriented programming languages. In particular, the specification consists of proposals for C++ language binding and Smalltalk language binding to a database language' and the database language is specified in proposals for a data definition language, a data manipulation language, and a query language. Of these proposals, the C++ language binding is the most mature, as these vendors have a lot of expertise in providing persistent storage for C++. However, in my view, the query language (at least as it is currently presented) is woefully deficient; and the data definition language and data manipulation language have problems, but are a reasonable basis for further work.
- In essence, the database language proposal is an Object SQL. It uses the familiar SELECT FROM WHERE clauses of SQL. It includes the ORDER BY and GROUP BY clauses; the aggregation functions MIN, MAX, COUNT, SUM, and AVG; the set UNION, INTERSECTION, and DIFFERENCE queries, the existential quantifier predicate EXISTS, etc. It even has the SQL–2 facility called "derived table" to allow the SELECT clause of a query to contain another query. Besides these, the language includes facilities for defining and manipulating compound data structures (i.e., sets, bags, lists, arrays, and structs) that SQL does not support.
- The major problems and deficiencies in the ODMG–93 database language are due to the fact that the database language does not subsume the facilities in SQL (despite the fact that the data model on which the database language is based, the Core Object Model espoused by the Object Management Group (OMG), fully subsumes the relational model). The database language is missing some important elements of SQL, including views, dynamic schema changes, and access authorization. Further, the current language includes many little features that will add up to a "meta–data management" nightmare, such as naming each individual object, maintaining multiple overlapping subsets of all objects in a type, etc.

The goal of ODMG, with respect to the database language (but not the language bindings for C++ and Smalltalk), is largely identical to that of the X3H2 Database Standards Committee (i.e., the SQL–3 Committee); namely, the development of a database language for an object–oriented database as a post–relational database. SQL–3 is envisioned in essence as an Object SQL which extends SQL–2 with facilities for defining, manipulating, and querying an object database as a superset of a relational database. The SQL–3 committee includes all major relational database vendors, and some OODB vendors.

Although members of ODMG are supposedly to implement the ODMG–93 proposal within 18 months, at this point it is not clear how many of the members will actually do so, and how much of the proposal they will implement. The technical challenge of implementing the full ODMG language, especially the automatic query optimizer and query processor, is one that is unlikely to be met in 18 months. ODMG had five original members: Object Design, Inc., O2 Technology, Versant Object Technology, Objectivity, Inc., and Ontos. However, Versant Object Technology has opted for a joint development and marketing of an Object SQL product based on UniSQL's SQL/X object–oriented SQL. Objectivity, Inc. has announced a plan to deliver an object–oriented SQL product by early 1994 which extends a pure SQL language processor they licensed from an SQL vendor. (Objectivity claims that they will offer both the SQL interface product and an ODMG–based query language product.) Ontos has had their own (limited) Object SQL for some time. O2 Technology claims that it already supports ODMG–93. (Although it is not a member of ODMG, UniSQL also supports most of the features found in ODMG–93, albeit in somewhat different syntax.)

The ODMG–93 database language specification, from an academic perspective, represents a much more concrete progress than the current confused state of SQL–3. In fact, despite some problems, its treatment of compound data (sets, bags, arrays, structs) — including the definition and creation of a compound data, and retrieval of the elements in a compound data — is a significant accomplishment. The ODMG–93 database language also accounts for nested data (e.g., parts explosion), methods, and inheritance in the language to some extent. However, the ODMG–93 database language has many major problems and deficiencies, and is premature to be considered a "standard" for "anything".

The biggest problem with the ODMG–93 database language is that it fails to recognize that the OMG Core Object Model on which it is based to is a superset of the relational model of data, and as such the ODMG database language should be a superset of ANSI SQL. For example, a type may have attributes, relationships, and methods; whereas a table in a relational database may have only attributes — in other words, a type subsumes a table. Further, the domain of an attribute under the OMG Core Object Model may be a primitive type (e.g., integer, float, char, date, time, money) or a compound type (e.g., set, bag, list, array, struct, enumerated type); whereas the domain of an attribute of a table in a relational database may only be a primitive type — in other words, the set of types supported under the Core Object Model subsumes the set of types supported in relational databases. The failure to recognize this simple fact has led to the current language which is similar to SQL but not compatible with SQL.

Given that ODMG does not include any relational database vendors as members, and that relational database vendors are working to develop the SQL–3 object–oriented SQL, ODMG needs to address the major technical problems in the database language parts of ODMG–93 and work with the ANSI X3H2 SQL–3 Committee to arrive at a single standard for a database language. The C++ and Smalltalk bindings, as specified in ODMG–93, should still be able to work with the database language.

2. ODMG–93 Database Language

2.1 What a Database Language Is

Before we proceed, it is important to briefly review what a database language is. A complete database language, such as SQL, consists of three sublanguages: data definition language (DDL), query and data manipulation language (DML), and data control language (DCL).

The DDL is used for specifying the structure and integrity conditions on the database schema. In a relational database, the DDL is used to define tables, attributes in a table, the domain of an attribute, and constraints on an attribute or a table. In an OODB, the DDL should be used to define types, attributes in a type, the domain of an attribute, and constraints on an attribute or a type. The DDL for an OODB, however, must be richer than the DDL for a relational database. This is because an OODB is supposed to admit additional types of information which relational databases do not; namely, methods, compound data, nested data (e.g., parts explosion), and inheritance and IS–A relationships among types.

The DML is used for creating (inserting), updating, and deleting data that populates the database schema. The query language is used for nonprocedurally retrieving a subset of the database that satisfies user–specified search conditions. In a relational database, the DML is used to populate the database based on the database schema defined using the DDL. Once the database has been populated, the query language and DML are used to retrieve data from the database and to update and delete the contents of the database. In an OODB, the DML and query language must serve the same purposes. Because the DDL of an OODB allows additional types of information to be represented (and therefore stored in the database), the DML and query language must necessarily be more powerful than SQL in order to access and manipulate the additional information stored in the database.

The data control language is used for managing transactions (i.e., commit work, rollback work, restart), for controlling the access of the database by multiple users (i.e., grant or revoke access authorization), and for managing database resources (create index, drop index), for enforcing database integrity based on user–specified conditions (create trigger), etc.

Of the sublanguages, the query language is the most involved and difficult to design and implement. Efficient evaluation of a query is a major determinant of the performance of a database system. Unfortunately, this is the part of the ODMG–93 database language that is given the most skimpy treatment.

2.2 Technical Contributions of ODMG–93 Database Language

In my view, the most important contribution of the ODMG–93 database language is the specification of compound data types in the ODL, OML, and OQL. The ODMG–93 database language provides a broad set of facilities for defining and creating each type of compound data, facilities for accessing elements of a compound data (get first, get last, get i–th element), and facilities for manipulating more than one compound data (e.g., flatten a list, concatenate lists, compute the union of bags, compute the intersection of bags, etc.). SQL has no such facilities. I note, however, that compound data is not one of the primary object–oriented concepts, namely, encapsulation and inheritance. In my view, the ODMG–93 database language proposal on compound data management addresses the technical challenges in making programming languages persistent and also one of the longstanding deficiencies in relational database systems.

A second important contribution of the ODMG–93 database language is the introduction of strong typing in the OQL. The user must indicate the type of the result of a query or the database language processor will infer the type of the query result. A strongly typed language makes it possible for invalid operations to be detected at compile time, rather than at run–time.

(The ODMG–93 database language proposes a simplified syntax for creating new objects (rather than using the SQL INSERT statement) and accessing named objects (rather than using the SELECT FROM WHERE query). In my view, however, this is not a significant contribution, since relational database systems also offer call–level interfaces to create and access records in much simpler ways than using the SQL INSERT or SELECT statements.)

2.3 Technical Problems of ODMG–93 Database Language

The ODMG–93 database language has the following three types of technical problems that need to be addressed.

The OQL in its current form is missing almost all of the SQL extensions that are required to account for the additional types of information that the ODL and OML allow to be defined and stored in the database. In particular, the OQL has virtually no mention of queries that involve methods or nested data; and it does not define queries that involve a type hierarchy. (The France–based O2 Technology, who authored the OQL part of the ODMG–93 "specification", claim that they already support all these features. However, the current OQL "specification" merely includes one simple example that shows the use of a method in a query, and one simple example that shows a path expression — and no additional semantic or syntactic specifications. A specification for each such feature is much more involved than can be described in such a skimpy manner. Below, I provide a brief review of the motivation and issues for these features.)

- It proposes a different syntax and terminology from SQL even for those facilities whose semantics are identical or compatible with SQL. For example, it uses different keywords for the set difference operation (it calls it EXCEPT), GROUP BY HAVING (it calls it GROUP BY WITH), ORDER BY (it calls it SORT BY), etc.
- It has not adequately accounted for the semantic consequences of object–oriented concepts on a query language. In particular, the semantics of queries involving methods and nested data are woefully deficient, and queries involving an inheritance hierarchy of types are not even included.
- It is simply missing some major features found in SQL and relational database systems: views, access authorization, triggers, and dynamic schema changes. It also has no SQL–like statements for creating new objects, updating a set of objects based on query search conditions, etc.

The first two problems above arise because the OQL of ODMG–93 is an object–oriented SQL; and because the object–oriented data model on which the ODMG–93 database language is based fully subsumes the relational model, the OQL should simply subsume SQL. In other words, the OQL should simply be designed such that if the users will not use any object–oriented extensions to SQL, it should naturally degenerate into the standard SQL. The second problem above, the seemingly gratuitous syntax and terminology change, may be

Although members of ODMG are supposedly to implement the ODMG–93 proposal within 18 months, at this point it is not clear how many of the members will actually do so, and how much of the proposal they will implement. The technical challenge of implementing the full ODMG language, especially the automatic query optimizer and query processor, is one that is unlikely to be met in 18 months. ODMG had five original members: Object Design, Inc., O2 Technology, Versant Object Technology, Objectivity, Inc., and Ontos. However, Versant Object Technology has opted for a joint development and marketing of an Object SQL product based on UniSQL's SQL/X object–oriented SQL. Objectivity, Inc. has announced a plan to deliver an object–oriented SQL product by early 1994 which extends a pure SQL language processor they licensed from an SQL vendor. (Objectivity claims that they will offer both the SQL interface product and an ODMG–based query language product.) Ontos has had their own (limited) Object SQL for some time. O2 Technology claims that it already supports ODMG–93. (Although it is not a member of ODMG, UniSQL also supports most of the features found in ODMG–93, albeit in somewhat different syntax.)

The ODMG–93 database language specification, from an academic perspective, represents a much more concrete progress than the current confused state of SQL–3. In fact, despite some problems, its treatment of compound data (sets, bags, arrays, structs) — including the definition and creation of a compound data, and retrieval of the elements in a compound data — is a significant accomplishment. The ODMG–93 database language also accounts for nested data (e.g., parts explosion), methods, and inheritance in the language to some extent. However, the ODMG–93 database language has many major problems and deficiencies, and is premature to be considered a "standard" for "anything".

The biggest problem with the ODMG–93 database language is that it fails to recognize that the OMG Core Object Model on which it is based to is a superset of the relational model of data, and as such the ODMG database language should be a superset of ANSI SQL. For example, a type may have attributes, relationships, and methods; whereas a table in a relational database may have only attributes — in other words, a type subsumes a table. Further, the domain of an attribute under the OMG Core Object Model may be a primitive type (e.g., integer, float, char, date, time, money) or a compound type (e.g., set, bag, list, array, struct, enumerated type); whereas the domain of an attribute of a table in a relational database may only be a primitive type — in other words, the set of types supported under the Core Object Model subsumes the set of types supported in relational databases. The failure to recognize this simple fact has led to the current language which is similar to SQL but not compatible with SQL.

Given that ODMG does not include any relational database vendors as members, and that relational database vendors are working to develop the SQL–3 object–oriented SQL, ODMG needs to address the major technical problems in the database language parts of ODMG–93 and work with the ANSI X3H2 SQL–3 Committee to arrive at a single standard for a database language. The C++ and Smalltalk bindings, as specified in ODMG–93, should still be able to work with the database language.

2. ODMG–93 Database Language

2.1 What a Database Language Is

Before we proceed, it is important to briefly review what a database language is. A complete database language, such as SQL, consists of three sublanguages: data definition language (DDL), query and data manipulation language (DML), and data control language (DCL).

The DDL is used for specifying the structure and integrity conditions on the database schema. In a relational database, the DDL is used to define tables, attributes in a table, the domain of an attribute, and constraints on an attribute or a table. In an OODB, the DDL should be used to define types, attributes in a type, the domain of an attribute, and constraints on an attribute or a type. The DDL for an OODB, however, must be richer than the DDL for a relational database. This is because an OODB is supposed to admit additional types of information which relational databases do not; namely, methods, compound data, nested data (e.g., parts explosion), and inheritance and IS–A relationships among types.

The DML is used for creating (inserting), updating, and deleting data that populates the database schema. The query language is used for nonprocedurally retrieving a subset of the database that satisfies user–specified search conditions. In a relational database, the DML is used to populate the database based on the database schema defined using the DDL. Once the database has been populated, the query language and DML are used to retrieve data from the database and to update and delete the contents of the database. In an OODB, the DML and query language must serve the same purposes. Because the DDL of an OODB allows additional types of information to be represented (and therefore stored in the database), the DML and query language must necessarily be more powerful than SQL in order to access and manipulate the additional information stored in the database.

eliminated by simply accepting the SQL standard. In the remainder of this section, I will discuss the major problems and omissions in the ODMG–93 database language in some detail.

Deficiencies of Query and Data Manipulation Language

The OML does not support inserts, updates, and deletes that are based on query specifications; that is, OML offers no means of inserting, updating, or deleting more than one objects based on their satisfying arbitrary search conditions. Standard SQL supports such facilities.

An attribute that holds a value is a special case of a method; that is, an attribute has a method for reading the value and a method for updating the value. An attribute is an essential element in a search condition (predicate of the form "attribute_name comparison_operator value_expression; e.g., Salary > 50000). It should be possible to allow a method anywhere in a search condition where an attribute name may be used (e.g., RetirementBenefits > 30000, where RetirementBenefits is a method). But it is very difficult to support queries that involve arbitrary user–supplied methods. There are such issues as whether methods will reside on the client or server in a client/server environment, and whether even unsafe methods (methods with unpredictable side effects) should be allowed in queries. Further, arbitrary methods are not amenable to automatic query optimization. These considerations impose practical limitations on allowing queries that involve methods. The OQL provides no such considerations.

The notion of a "path query" has been well–developed by database researchers during the past decade. A path query is a query written against nested data, by specifying search conditions against nested data. A path query contains, instead of just an attribute name, a sequence of attribute names (called a path expression). For example, a Person type may have an attribute named Hobby; the domain of Hobby may be an Activity type; and the Activity type may have an attribute named Number_of_Participants. Then it should be possible to issue a single query that says "find all persons whose hobby includes an activity that involve four or more participants." The WHERE clause of the query may contain a predicate
"Person.Hobby.Number_of_Participants > 4".
The OQL specification, except for one short paragraph in which one simple path query is illustrated, does not define the semantics of a path query. A path expression contains a sequence of attributes. When any of the attributes in a path expression has a compound data as its domain, the meaning of the path expression becomes complicated. Further, a path expression should allow methods as well as attributes. A path expression also makes it difficult for automatic query optimization and processing. The OQL provides no such considerations.

The OQL allows a query on only a single type, but does not allow an inheritance–hierarchy query (i.e. a query that is targeted to an entire type hierarchy). An inheritance hierarchy of types implies an IS–A relationship; that is, an entity represented by a subtype "is a kind of" an entity represented by a supertype. For example, an Employee type "is a kind of" a Person type, where Employee is a subtype of Person (and conversely, Person is a supertype of Employee). Sometimes it makes sense to enquire about Person objects; and other times it is useful to be able to enquire about all types of Person (i.e., Person objects and Employee objects collectively).

Deficiencies of Data Definition Language

The ODL has an important omission, namely view support. Relational database systems all support views as the external schema over the conceptual schema of an integrated database, and are used as a unit of access authorization. A user may define a view over a table by, for example, omitting certain attributes, and authorize another user to access the database only through the view. The utility of views remains the same for object–oriented databases; it is not true at all that somehow object–oriented databases obviate the need for views. In fact, without views, access authorization can only be supported partially. I note that the semantics of views for object–oriented databases are much more involved than their counterpart for relational databases. This is because a view is almost like a table in a relational database. If this semi–equivalence of a view and a table is to be transferred to an object–oriented database, a view should be almost like a type; which means that views may form an inheritance hierarchy, a view may be used as the domain of an attribute, and a view may have methods as well as attributes.

The ODL also is missing facilities for making dynamic changes to the database schema, beyond just adding a type as a new subtype of some existing types. Even relational database systems allow a table to be dynamically added or dropped, and an attribute to be added or dropped. Since an object–oriented data model admits additional information, its DDL should include facilities for making such schema changes as adding a method or attribute to an existing type, dropping a method or attribute from a type, adding a new supertype to an existing type, dropping a supertype from a type, etc.

Deficiencies of Data Control Language

The ODMG–93 database language has (almost) no data control language; in particular, it has no provision for access authorization (granting and revoking authorizations), and no provision for triggers (for automating the enforcement of data integrity with user–specified actions). Again, the facility for granting authorization should include consideration of the object–oriented data model; for example, authorization to run methods, and authorization to access not just one type, but a type hierarchy.

The ODMG–93 database language proposes a nested transaction model (a transaction recursively consisting of other transactions) for transaction management. A transaction is simply a sequence of reads and updates against a database (any database — relational or object–oriented). The purpose of bundling a sequence of reads and updates into a single transaction is simply to define the sequence as an atomic database access. Either all the reads and updates in a transaction finish (i.e., the effects get recorded permanently in the database), or none finish. In this way, the database is not corrupted with data as a result of a partial work. A nested transaction model is sometimes desirable, since it naturally models a situation in which the work of a transaction may be split into multiple transactions (subtransactions) which may be executed in parallel. However, this has nothing whatever to do with object–orientation (i.e., encapsulation, methods, inheritance); that is, object–orientation does not require that transactions be nested. In my view, ODMG should adopt the standard non–nested transaction model as default, and suggest a nested transaction model as an option.

Potential Meta–Data Management Nightmare

The ODMG–93 database language has various little facilities that will add up to a "meta–data management" nightmare. For example, it proposes to allow a user to specify multiple names for a single object; to specify the lifetime of a single object; to create an index on a subset of an extent of a type (i.e., a subset of all instances of a type — equivalent to a subset of all records of a table in a relational database); to create multiple such subsets of an extent of a type, such that some objects may belong to more than one subset; to name a query statement and use it in other queries, etc.

I note that it is of course desirable, conceptually, to allow the user to manage the database at the object level, that is, to use a single object as the smallest unit of database access and control. For example, in a document management application, it is certainly desirable to be able to name each document object, attach an authorization on the document (although ODMG–93 does not include any consideration of authorization), to fetch the document by its name, etc. However, it must also be possible to manage the database at the type level, that is, to use a type (and its extent) as a unit of database access and control — relational database systems allow this, but do not allow each tuple (record or row) of a table as a unit of database access and control. Although it is desirable to have a single object as a unit of database access and control, it may sometimes lead to significant additional difficulties with meta data management. Further, it can lead to a performance degradation; a larger system catalog to maintain the status of each named object, a larger catalog to keep track of access authorizations on individual objects, a more complex logic for authorization checking, a more complex logic to screen redundant objects when processing queries against multiple subsets of the extent of the same type, etc. In summary, it should be recognized that an object–level database access and control is merely the next logical step from the traditional type–level access and control, rather than a replacement of the type–level access and control. (Although I do not regard this as a "major" problem, the ODMG–93 database language requires the user to explicitly name the extent of a type. This seems to merely add to the meta–data management problem. It is not clear at all why the extent of a type cannot be implicit. In a relational database, the extent of a table is implicitly all the records that are inserted into the table — the user need not declare the extent of a table explicitly and separately.)

C++ and Smalltalk Bindings

Both the C++ and Smalltalk bindings force all objects to be persistent; and this is regarded as totally unacceptable to some users and prospective users of C++ and Smalltalk. It will be necessary to allow C++ or Smalltalk applications to deal with both persistent objects and nonpersistent objects. To do this, it will be necessary to allow the users to declare persistent classes and persistent attributes, to differentiate them from nonpersistent classes and nonpersistent attributes, respectively. Further, the issue of storing persistent objects that reference nonpersistent objects needs to be addressed.

3. Future Course for ODMG

In order to understand the relative merit of ODMG and ODMG–93 and the future course of action that ODMG should take, we must bear in mind the fact that relational database vendors are planning to add object management facilities to their relational database products, that is, extend SQL to object–oriented SQL and that they are all working within the ANSI X3H2 (SQL–3) Standards

Committee. Given that OODB vendors that are members of ODMG have elected to adopt SQL as the basis of their object–oriented database language, it is clear that ODMG should really join forces with ANSI X3H2 to arrive at a single standard. To this end, ODMG may do the following.

1. ODMG should start developing levels of conformance and certification process if they hope to have their proposals become a standard. The current proposal is "all or nothing"; either a vendor must implement all aspects of all of the proposals or they do not conform at all.
2. ODMG should quickly proceed to implement a certification process for the C++ and Smalltalk language bindings, as these are the most mature parts of its current proposal.
3. ODMG should influence SQL–3 with the two primary technical contributions of the ODMG–93 database language, namely management of compound data and strong typing of query results.
4. ODMG should make a revised proposal that will fully subsume SQL, while preserving its two primary technical contributions. In particular, it should simply accept and extend the major missing features from SQL, namely views, dynamic schema changes, and access authorization; and drop gratuitous differences in syntax and terminology from SQL.

In my view, the first step ODMG may take is to reorganize its data model and database language into three tiers. The first tier should consist only of those data modeling concepts and language constructs that are identical to their counterparts in relational databases. The second tier may consist of major new concepts and constructs, such as the concepts of relationships between two types, inheritance hierarchy, methods, and compound data types. The third tier may consist of the "little" things that can lead to meta–data management problems. In practice, the first two tiers should be combined into a single tier; but the artificial separate presentations may make it clear that an object–oriented model subsumes the relational model. In any case, the first two tiers should be the baseline for compliance certification for vendors. The third tier should strictly be an option for compliance.

Finally, ODMG proposes to expand ODMG–93 with proposals for additional database features. Before they proceed, they should first seek to broaden its membership to add more experiences in discussions involving some of the really thorny issues ODMG claims to have on its agenda. The small number of members whose primary expertise is in providing persistent storage for C++ should not be so presumptuous as to believe that they will define standards for such far–reaching and thorny issues as versioning, work–group transactions, multimedia data management, etc. These issues are very important to a far wider area of computing industry than just persistent storage for C++ or Smalltalk.