OPM Data Management Tools for CORBA Compliant Environments*

(Working Document)

Anthony Kosky†

Ernest Szeto‡

I-Min A. Chen§

Victor M. Markowitz¶

Data Management Research and Development Group
Information and Computing Sciences Division
Lawrence Berkeley National Laboratory
1 Cyclotron Road, Berkeley, CA 94720

June 1996

*Issued as Technical Report LBNL–38975. This work is supported by the Office of Health and Environmental Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

†Author's e-mail address: *Anthony_Kosky@lbl.gov*, phone: (510) 486-5471, fax: (510) 486-4004
‡Author's e-mail address: *E_Szeto@lbl.gov*, phone: (510) 486-7565, fax: (510) 486-4004
§Author's e-mail address: *IAChen@lbl.gov*, phone: (510) 486-7264, fax: (510) 486-4004
¶Author's e-mail address: *VMMarkowitz@lbl.gov*, phone: (510) 486-6835, fax: (510) 486-4004

Contents

1 Introduction

In this document we describe the development of an OPM-CORBA interface supporting the employment of OPM data management tools in a CORBA environment.

The Object-Protocol Model (OPM) is a an object data model that allows specifying database schemas in terms of objects and protocols [1]. OPM follows loosely the ODMG-93 standard [10], and supports additional constructs for modeling scientific experiments (protocols) and versions.

CORBA supports an Interface Definition Language (IDL) that has OPM's object-oriented flavor. However, IDL is not appropriate for defining database schemas, mainly because it provides an open *syntactic* framework for defining object interfaces in terms of methods, where code implementing these methods is transparent to the interfaces defined in IDL. This open syntactic framework is key to CORBA's support for object interfaces, where the semantic definition (i.e. interpretation of object representations) is filled in by the underlying systems, such as databases or application programs. Conversely, an integral part of object data models such as OPM, is a *semantic* interpretation for object structures (i.e., object state definition). Such a semantic interpretation of object structures is an essential component of the semantic definition of a database query language, such as OPM's query language. Furthermore, the granularity of database (e.g., OPM) objects whose type is predefined as part of the database design, may be different than that of interface (IDL) objects, which are often ad-hoc queries and query results whose type is not known ahead of time.

OPM is valuable because of its data management tools providing facilities for constructing databases with commercial relational database management systems (DBMSs), such as Sybase and Oracle, for constructing OPM views for existing relational and ASN.1 databases, and for querying such databases using the OPM query language, via OPM interfaces based on the OPM query translator (OPM-QLT). Furthermore, OPM provides an open framework for adding new constructs deemed important for a large class of applications and for developing new tools that, in combination with the core OPM tools, can be used for achieving new goals. For example, the currently developed OPM multidatabase toolkit (OPM*MT) provides facilities for constructing and exploring multidatabase systems involving heterogeneous databases.

We describe the development of an OPM-CORBA interface that allows client objects to access OPM tools in a CORBA environment, and discuss other ways in which OPM tools could interact with CORBA compliant systems.

Note that in discussing CORBA interfaces and tools, one must be careful in determining the boundaries between the vision of the CORBA standard, the speculations around this continuosly evolving standard, and the reality of what existing CORBA products support or are likely to support in the future.

The rest of this document is organized as follows. Section 2 contains a brief overview of the CORBA standard. OPM and the OPM data management tools are reviewed in Section 3. A CORBA compliant OPM interface for interacting with OPM data management tools in a CORBA environment is described in Section 4. Section 5 contains concluding remarks.

2 CORBA: A Brief Overview

CORBA is a system of standards describing facilities for client-server communication and interaction between distributed software objects [5, 6, 8]. The CORBA standards have been devised by *OMG*, a large consortium of software and hardware vendors. A number of vendors are producing systems compliant with the CORBA standards for communication between objects, so it seems likely that CORBA will become the industry standard for client-server systems and distributed software.

2.1 The CORBA ORB

The *Object Request Broker* (ORB) specification is the part of CORBA which describes a "software bus": a mechanism which handles communication between distributed objects. The ORB allows for client-server interaction between heterogeneous objects distributed over a wide-area network, and makes meta information describing the objects in a system and their interfaces available to any object in the system, so that it may access other objects as a client without prior knowledge of their existence. Any object connected to the ORB can play the role of both a client and server object: that is it can initiate calls to other objects and respond to requests for services from other objects on the ORB.

The ORB specification is programming language, operating system and architecture independent. It allows vendors considerable flexibility in their choice of implementation methods. CORBA compliant ORBs are currently available from a number of different vendors based on mechanisms such as RPC, TCP/IP and sockets. Further the ORB allows for transparent communication between objects implemented using a variety of programming languages and operating systems.

The CORBA specification describes protocols for communication between ORBs, which should allow for ORBs provided by different vendors to communicate in a federation. Several vendors are now offering CORBA 2.0 compliant ORBs, though it is not clear that compatibility issues between ORBs from different vendors have yet been resolved.

2.2 The Interface Definition Language

The *Interface Definition Language* (IDL) defined by OMG is a language for describing the *interfaces* of software objects. According to the CORBA 2.0 specification "an *interface* is a description of the set of possible operations a client may request of an object" [8]. It is important to note that an interface does not specify the internal data-representation or executable code used to implement an object. In practice an IDL interface specification may also contain declarations of types, exceptions and constants, and IDL supports multiple inheritance among interface definitions, in order to facilitate re-use and extensibility of classes. An IDL specification may contain declarations for *public* attributes of an object, but in reality these are a short-hand for the declaration of a pair of accessor functions for that attribute. The IDL syntax is based on, but is not a strict subset of, C++ declarations. IDL is independent of programming languages, and may be used to describe objects implemented using a variety of programming languages, compilers or operating systems.

The role of an IDL specification is analogous to that of a header (.h) file in a conventional programming language such as C or C++. An IDL specification describes how to use an object: that is, it describes the object's publically accessible attributes and the usage signatures of its methods, but it does not define the semantics of the data or methods of an object, or any other parts of an object that are hidden and not part of the interface. Consequently IDL *does not*, in and of itself, constitute a data-model. It would be possible to base a data model on a subset of the IDL *syntax*, but doing so would require providing a different *semantics* for the language. Since IDL does not have support for many concepts that are important in data modeling, such as various integrity constraints, it is doubtful that such a re-interpretation of the language would be beneficial.

The IDL specification for an object is used to automatically generate *"stub"* and *"skeleton"* programs for the object. The stub provides an interface for other client objects to request services from the object via an ORB. The stub performs tasks such as converting parameters and returned values into a form which allows them to be transmitted via the ORB. The skeleton acts as an interface between an object in its server role and the ORB. IDL specifications can also be used to generate header files in a variety of programming languages, including C++ and Smalltalk, on which implementations can then be based.

In addition the information represented by the IDL specification for any objects connected to an ORB is compiled and stored in the *Interface Repository* service which the ORB must provide. The interface repository can be examined by objects on the ORB in order to ascertain what other objects are connected to the ORB and what interfaces they provide. This allows an object to request services from other objects on the ORB without having prior knowledge of the other objects or their interfaces.

2.3 CORBA Services

OMG has adopted, or is in the process of adopting, specifications for a number of *"services"* as part of the CORBA standard. These services consist of descriptions and IDL specifications for modules to perform various common software engineering tasks such as transaction management, concurrency, security and so on. It is hoped that software vendors will provide modules implementing these descriptions, and that software developers will use them, thus allowing users to mix and match software modules and reducing software development time by avoiding duplication. Currently few of the CORBA services have commercial implementations available, and it seems likely that commercial implementations of many of the services will not be available for some time.

2.4 CORBA Object Query Service

The CORBA Object Query Service Specification is a joint proposal by IBM, Objectivity, O2, SunSoft, Sybase, Taligent and others. The proposal consists of specifications of modules for implementing collections of objects, and queries over collections [7]. Note that the Query Service is not a query language but rather a framework for query interfaces.

The *Collections* module of the Query Service specification describes classes for handling arbitary collections of CORBA objects, and includes support for adding and deleting objects and for iterating over collections.

The *Query* module specifies interfaces for *query evaluators* and *query management classes*. Query evaluators are required to support queries either in the SQL-92 standard relational query language or in ODMG-93 OQL standard object-oriented query language (though not necessarily both). SQL is assumed for queries against relational databases, where relations are specified as specializations of collections in which the component objects represent records. Query evaluators may work by performing queries locally, or by translating them into native database query systems. Note that much of the semantics of the Query Service module are described informally in the text accompanying the OMG proposal, but are not formally imposed by the IDL specification. For example the IDL specification requires that a Query Evaluator has a method "evaluate" which takes a query and returns an object, and defines exceptions representing invalid queries or query processing errors, but it does not define how the returned object should relate to the submitted query, or under what circumstances an exception should be raised. Consequently many aspects of how a Query Service module should be implemented, or how it should evaluate a query are left ambiguous: this provides a high degree of flexibility for vendors implementing the Query Service specification, but is also liable to lead to misinterpretation of the intensions of the proposal.

There are a number of features missing from the Query Service specification which are, in the authors' view, important for the support of database queries in CORBA. These include facilities to represent and explore meta-data describing the schema of a database, facilities for providing semantically enriched views of a database or collection, and facilities for performing queries across multiple databases or collections.

There are no implementations of the Object Query Service specifications available at present, and none scheduled for release in the near future of which the authors are aware.

3 The Object-Protocol Model and Tools

In this section, we briefly review the main constructs of the Object-Protocol Model (OPM), the OPM query language (OPM-QL), and the OPM data management tools. OPM is an object data model whose non-versioned object part is closely related to the ODMG-93 standard for object-oriented data models [10]. In addition, OPM supports object versioning and a protocol construct for modeling scientific experiments. The OPM query language (OPM-QL) also follows the ODMG-93 standard for object-oriented query languages. OPM is described in [1]; OPM-QL is described in [3].

3.1 The Object-Protocol Model and Query Language

Objects in OPM are uniquely identified by object identifiers (oids), are qualified by attributes, and are classified into classes. A subset of the attributes associated with a class is specified as the external object identifier. A class can be defined as a subclass of other (super) classes, where a subclass *inherits* the attributes of its superclasses. OPM supports multiple inheritance in class hierarchies.

Attributes can be *simple* or consist of a *tuple* (aggregation) of simple attributes. A simple attribute can have a single value, a set of values, or a list of values, and can be *primitive*, if it is associated with a system-provided data type, or *abstract*, if it takes values from object classes. The attributes of an object class can be partitioned into *non-versioned* and *versioned* attributes. Non-versioned attributes represent stable object properties while versioned attributes represent evolving object properties of an object.

Protocol classes in OPM are used to model scientific experiments. OPM supports the recursive specification (expansion) of protocols, where a protocol can be specified in terms of alternative subprotocols, sequences of subprotocols, and optional protocols. A protocol class can be associated with regular as well as input and output attributes that are used for representing input-output protocol connections.

OPM supports the specification of derived attributes using derivation rules involving arithmetic expressions, aggregate functions, and attribute composition. OPM also supports two types of derived object classes: derived subclasses and derived superclasses. A derived subclass is defined as a subclass of another derived or non-derived object class with an optional derivation condition. A derived superclass is defined as a union of two or more derived or non-derived object classes.

The OPM query language follows the ODMG standard for object-oriented query languages [10]. An OPM query involves local, inherited, derived and system attributes and path expressions starting with these attributes. An OPM query can involve conditions consisting of and-or compositions of atomic comparisons, and can contain an order-by clause, which specifies an attribute whose values are used for sorting (in ascending or descending order) the class instances returned by the query.

3.2 The Core OPM Tools

The core OPM data management toolkit includes tools for constructing, maintaining and querying OPM databases, and for constructing OPM views on top of existing databases [1].

OPM schema editing and browsing tools are provided for specifying and examining an OPM schema for a particular application. The OPM schema translator can then be used for generating automatically complete definitions for the underlying relational (Sybase or Oracle) database, including rules and constraints required for maintaining data integrity. A *mapping dictionary* records the correspondences between the classes and attributes of an OPM schema and the underlying relational tables, and is used in translating OPM queries and updates into their relational (SQL) correspondents.

Pre-existing, non-OPM databases can be retrofitted with an OPM view (schema) using the OPM *retrofitting* tools. Retrofitting involves first generating a canonical OPM schema from the native database schema, and then refining, and thus semantically enhancing, this view via a series of schema restructuring manipulations. The retrofitting tools can be used for constructing multiple OPM views for a single (OPM or non-OPM) database. The retrofitting tools the mapping dictionary that is used in conjunction with the other OPM tools in order to browse or query the underlying databases. Retrofitting tools are currently available for relational and ASN.1 databases, with versions of the tools for other databases planned.

The OPM query translator supports processing OPM-QL queries. Two alternative approaches to OPM-QL processing are supported: (1) using stored procedures, which provides maximum efficiency for fixed-form queries on a native OPM database, and (2) generating SQL queries on the fly, which is suitable for free-form (ad-hoc) queries and/or for querying databases retrofitted with OPM schemas.

In addition the core OPM toolkit includes tools for publishing OPM schemas in a variety of formats, including HTML, PostScript and LaTeX, and for constructing Web form-based interfaces for querying OPM databases.

3.3 Developing Application Programs with the OPM Tools

Application programs can interact with OPM-QLT in two ways: (1) Access OPM-QLT via a C++ wrapper: metadata representing the OPM database is loaded at runtime through dynamic linkage; the OPM-QL query input, specified as a string, is translated into a series of SQL statements that are sent and executed through the DBMS API; the DBMS API returns a relations as query results, which in turn are converted into OPM data objects represented using C++ data-structures. (2) OPM-QLT is employed as a stand-alone application, and temporary ascii files are used for data-exchange; this approach can be used for example with PERL scripts implementing web-based query interfaces for OPM. Other alternatives that are currently considered include embedding OPM QLT inside a scripting language such as PERL, rather than just running it as an external utility as mentioned in (2).

Applications that use the first alternative mentioned above, are required to use C function calls and generic data structures, and must run on the same system as OPM-QLT. The second alternative mentioned above can be inefficient and restrictive if OPM-QLT is accessed frequently or the data returned needs to be interpreted for further processing. In addition handling errors and passing exception information to a client program is difficult under this approach.

Interacting with OPM-QLT could be more flexible and easier to implement via a uniform CORBA interface that would allow accessing metadata and data in an OPM database using a variety of programming languages, independent of the physical location of the OPM tools.

3.4 The OPM Multidatabase Tools

The multidatabase OPM (OPM*) toolkit, provides tools for constructing multidatabase systems consisting of heterogeneous databases, and for exploring (browsing, querying) such multidatabase systems. This toolkit is built on top of the core OPM toolkit and includes tools for: (1) assembling component databases into an OPM-based multidatabase system, while documenting their schemas and inter-database links; (2) processing ad hoc multidatabase queries via uniform OPM interfaces; and (3) assisting scientists in specifying and interpreting multidatabase queries.

Incorporating an MBD into an OPM multidatabase system involves constructing one or more OPM views of the MBD, and entering information about the MBD and its views into a multidatabase directory. The multidatabase directory stores information necessary for accessing and formulating queries over the component MBDs, including: (1) general information describing each MBD accessible in the system, and the information required for accessing the MBD; (2) structural information on the schemas of each MBD, including semantic descriptions of the physical or real world concepts represented by the schemas, synonyms and keywords for identifying differences in terminology and for establishing potential correspondences between components of different MBD schemas, and sample data illustrating how the schema constructs are for representing application data; (3) information on known links between different MBDs, including semantic descriptions of links, the nature of the correspondence represented by links, and data-manipulations, such as reformatting of accession numbers, that need to be performed in order to traverse the link.

Queries in an OPM-based multidatabase system are expressed in the OPM multidatabase query language (OPM*QL). OPM*QL extends the single-database OPM query language, OPM-QL, with constructs needed for querying multiple databases. These extensions include the ability to query multiple classes, possibly from distinct databases; constructs that allow navigation between the classes of multiple databases following inter-database links; and the ability to rename fields of a query in order to resolve potential naming conflicts between multiple databases.

Processing OPM multidatabase queries involves generating OPM-QL queries over individual databases in the multidatabase system, and combining the results of these queries using a local query processor. The stages of generating OPM-QL queries and manipulating data locally may be interleaved depending on the particular query evaluation strategy being pursued.

4 The OPM Tools in a CORBA Environment

The CORBA standard provides a valuable software environment that could increase the modularity of the OPM tools and help in the development of new versions of these tools. Furthermore, a CORBA compliant OPM interface would make the OPM tools and OPM-based databases accessible to any CORBA-compliant application, and hence will provide facilities for CORBA client objects to access and query databases implemented using a wide variety of DBMSs via semantically enhanced OPM views.

A CORBA-compliant OPM interface would have several advantages over a programming language specific API for accessing OPM databases, such as the C++ API mentioned in section 3. First, client applications would be able to transparently access local or remote OPM database servers, without concern for the physical location of the OPM server or the underlying database itself. Further, clients could be programmed using a wide variety of programming languages, such as Smalltalk, Java or C++, and implemented using a variety of platforms and operating systems, while making use of a common and well-defined interface.

For example a schema driven query or browsing system, similar to Genera [4], could retrieve an OPM schema via the CORBA compliant OPM interface, display this schema, and then process OPM queries expressed over this schema by submitting the queries to the OPM query translator and underlying database via the CORBA ORB, as described below (see figure 1). Such a client application could be implemented in any programming language supported by CORBA, and could access OPM tools running on a remote system. The client application could also be implemented on any platform for which a CORBA ORB is available, including platforms which are not directly supported by the OPM tools. Furthermore, it would be transparent to the client application whether it was accessing local or remote OPM databases, or even multiple versions of the OPM tools implemented on different platforms.

Another potential source of client applications for a CORBA compliant OPM interface are "intelligent agents for data mining and extraction" for sequence data annotation. Such intelligent agents would need to access a variety of heterogeneous genome databases distributed over the net, extract, verify, and fuse relevant data, and resolve data conflicts. The OPM tools and CORBA compliant OPM interface could provide support for developing such intelligent agents: they will allow agents to access distributed heterogeneous databases using a common data model and interface, retrieve metadata for these databases, and adapt to changes in the environment, such as changes in the location or accessibility of databases, or the addition of new data sources. Furthermore, such a system would allow agents to be implemented using the most appropriate choice of platform and programming language, rather than being restricted by the implementation of the database access libraries.

Ultimately any OPM CORBA-compliant tools or interfaces should be able to function equally well with any CORBA-compliant ORB. In practice however, differences and incompatibilities between available CORBA implementations require an actual implementation of a CORBA compliant OPM interface to be based on a specific CORBA product. We discuss below a possible architecture and implementation strategy for a CORBA compliant OPM interface, but do not get into the implementation details for a specific CORBA product.

A CORBA compliant OPM interface should provide server access to an OPM database (native or retro-fitted) via a CORBA-compliant ORB. It should allow client objects to access the metadata (schema) for an OPM database and to execute OPM queries over the database.

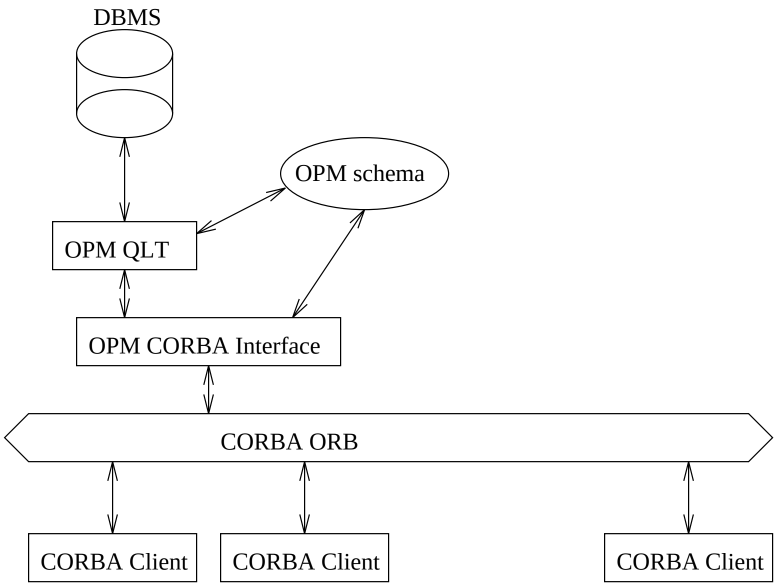

Figure 1: Connecting an OPM database to an ORB using the CORBA OPM Interface

There are two alternative approaches to providing such interfaces, namely generating an individual IDL interface for each OPM database, with IDL classes in a one-to-one correspondence with OPM classes, or providing a single generic IDL interface to access OPM databases. The first approach leads to a stronger correspondence between OPM objects and objects accessed via the CORBA ORB, which may be more natural for programming client applications for a specific database with a relatively static schema. However it would be less appropriate for building clients which require generic access to OPM databases, or which require access to databases with rapidly evolving schemas for the following reasons:

1. A new IDL interface must be generated and compiled for each OPM database.
2. The methods supported by such a CORBA interface would depend on the OPM class definitions. Consequently a client object accessing OPM databases will not be able to tell a priori what the interface to an OPM database will be, and will need to rely on the CORBA *dynamic invocation* facilities in order to access an OPM database. The CORBA *dynamic invocation* facilities are not as efficient as and require more complex programming than the *static invocation* facilities, and are not universally available or compatible for current CORBA ORBs.

3. Since queries must be submitted to a single CORBA object, queries against an OPM database would need to be posed against an individual OPM class or object, rather than an entire OPM database. This means that queries involving multiple classes must involve local computation, making optimization of such queries difficult. In general optimization is facilitated by computing as much of a query as possible on a remote database, and minimizing the additional computations that must be done locally.

Conversely a single generic interface allows for more uniform application code, may be accessed by clients more simply using static invocation, and offers greater flexibility and more scope for optimization in terms of the queries that can be evaluated at the OPM database. Note also that it would be a relatively easy task to construct OPM-class specific wrappers on top of a generic interface, while it is not clear that the reverse would be possible.

For these reasons we have decided to pursue a single generic CORBA interface for OPM databases. This interface will provide access to OPM metadata through generic representations of the data, and will allow for the execution of queries against the database using the OPM Query Language Translator (OPM-QLT) (see figure 1). The module specification will describe classes representing OPM schemas, OPM databases and query results.

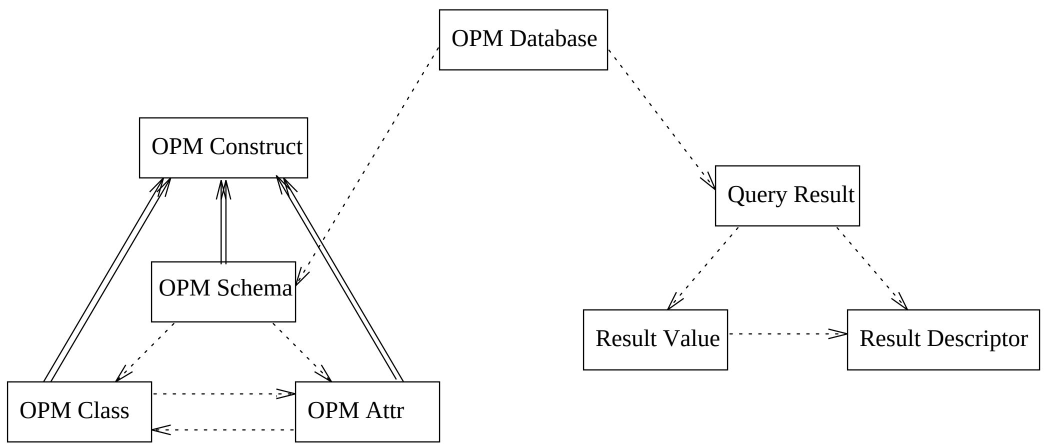

Figure 2: Main classes for the CORBA OPM interface

The main IDL classes of the CORBA OPM interface module are illustrated in figure 2: double arrows represent inheritance and dotted arrows represent dependencies in the IDL definitions:

1. The OPM Schema class has methods for accessing the metadata stored in an OPM schema, including OPM class and attribute definitions, descriptions, sample data and so on.

2. The OPM Database class has methods for accessing the schema of a database (returning an object of the OPM Schema class), and for submitting queries to a database.

3. Queries are submitted as strings representing an OPM-QL query. The result of such a query invocation will be an object of the Query Results class. The Query Results class will be self-describing, in that it will contain type information and metadata in addition to data values, and will provide methods for accessing the types and metadata for the data returned.

The IDL class specifications will be used for generating the appropriate stubs for the client application, while the methods for the IDL classes of the CORBA OPM interface will be provided as part of the OPM database server.

The combination of such an interface with the OPM retrofitting tools is particularly powerful: it allows CORBA applications to access databases implemented with a variety DBMSs using a single CORBA interface and query language, regardless of whether a CORBA compliant interface for a particular DBMS is available. This includes databases implemented using systems such as ASN.1/Entrez or ACeDB, which are in use within the molecular biology community, but which are not sufficiently wide-spread and are too far removed from standard data models for CORBA compliant interfaces to be made available for them in the foreseeable future. Additional benefits of this system include the ability to access metadata and constraints for a particular database, and the ability of OPM to provide a semantically enhanced view of a database, allowing queries to be formulated in terms of the conceptual entities represented by a database rather than the relational or other lower level representation of the database.

Future extensions of these modules will involve the OPM multidatabase tools, and will include support for accessing multidatabase metadata and inter-database links, in addition to allowing for multi-database queries.

5 Concluding Remarks

CORBA represents an important development in the field of distributed computing and is likely to have far reaching consequences. Though the CORBA standard does not, in and of itself, fulfil the need for general data management tools and for accessing heterogeneous distributed databases, we believe that making the OPM toolkits available to client objects via a CORBA compliant interface will fulfil such needs, and will add to the value of the existing OPM tools. Further, CORBA offers the potential for new extensions to the OPM toolkits including the ability to call external data viewing and analysis software from within OPM querying and browsing tools. Some of these extensions are discussed below.

5.1 Retrofitting Tools for CORBA Database Interfaces

Although currently there are no implementations of the CORBA Object Query Services, such implementations may eventually become available. Alternatively CORBA compliant uniform database interfaces such as Visigenic SOftware's ODBC, may become a popular standard in the future. In this case it will be possible to implement versions of the OPM retrofitting tools and query translators for such CORBA interfaces. This will allow the OPM tools to access any new DBMS supporting the CORBA interface without developing versions of the OPM retrofitting and query translator tools on a per-DBMS basis. Then the OPM tools would become available for the widest possible range of databases, and be available to any CORBA compliant clients.

5.2 Connecting External Modules to OPM Data

Some types of data, such as images, sound samples, crystalographic structures and so on, have structure or semantics that cannot be captured using certain DBMSs. Numerous software packages are available for manipulating or displaying such data. Extending OPM with support for such data types and associated methods based on existing software could be facilitated in a CORBA environment. CORBA supports dynamic linking of software objects, and consequently provides a general mechanism for calling external methods from within OPM query and browsing tools. In order to make use of such facilities, the OPM tools would require access to a local directory containing information on these special data-types, the methods they support, and the information necessary to access a CORBA server implementing these methods. This requires further investigation.

References

[1] Chen, I.A., and Markowitz, V.M., An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools, *Information Systems*, Vol 20, No 5 (July 1995), pp. 393-418. Additional documentation is available at http://gizmo.lbl.gov/opm.html.

[2] Chen, I.A., and Markowitz, V.M., OPM Schema Translator 4.0, Reference Manual, Lawrence Berkeley National Laboratory Technical Report LBL-35582 (revised), 1995.

[3] Chen, I.A., Kosky, A., Markowitz, V.M., and Szeto, E., The OPM Query Translator, Lawrence Berkeley National Laboratory Technical Report LBL-33706, 1996. Available together with additional documentation at http://gizmo.lbl.gov/opm.html.

[4] Letovsky, S., Genera: A Specification-Driven Web/Database Gateway Tool, http://gdbdoc.gdb.org/letovsky/wgen.html.

[5] Mowbray, J., Zahavi, R., *The Essential CORBA*, John Wiley and Sons, 1995.

[6] Orfali, R., Harkey, D., Edwards, J., *The Essential Distributed Objects Survival Guide*, John Wiley and Sons, 1996.

[7] *Object Query Service Specification*, OMG TC Document 95.1.1, January 1995; http://www.omg.org/docs/1995/95-01-01.ps.

[8] *The Common Object Request Broker: Architecture and Specification* Revision 2.0, July 1995. OMG TC Document 96.03.04, http://www.omg.org/docs/ptc/96-03-04.ps

[9] Abstracts of the Second Meeting on Interconnection of Molecular Biology Databases. Available at http://www-genome.wi.mit.edu/informatics/abstracts.html.

[10] *The Object Database Standard: ODMG-93*, Release 1.2. Cattell, R. G. G. (ed), Morgan Kaufmann, 1996.