Seamless Integration of Biological Applications within a Database Framework

Thodoros Topaloglou, Anthony Kosky and Victor Markowitz

Data Logic, A Division of Gene Logic Inc
2001 Center Street, Suite 600, Berkeley, CA 94704
Email: {thodoros, anthony, victor}@genelogic.com
Tel: (510) 649 3444, Fax: (510) 649 3449

Abstract

There are more than two hundred biological data repositories available for public access, and a vast number of applications to process and interpret biological data. A major challenge for bioinformaticians is to extract and process data from multiple data sources using a variety of query interfaces and analytical tools.

In this paper, we describe tools that respond to this challenge by providing support for cross-database queries and for integrating analytical tools in a query processing environment. In particular, we describe two alternative methods for integrating biological data processing within traditional database queries: (a) "light-weight" application integration based on Application Specific Data Types (ASDTs) and (b) "heavy-duty" integration of analytical tools based on mediators and wrappers. These methods are supported by the Object-Protocol Model (OPM) suite of tools for managing biological databases.

Introduction

In order to perform high-throughput analysis of biological data, it is necessary to access and process information from a variety of data sources using standard and proprietary query interfaces and analytical tools. These data sources may be heterogeneous, and distributed over intranets or the internet. This problem is compounded by the large number of public biological data repositories and the diversity of applications that are used to access, filter, interpret and combine these data. Although existing database and Web-based technologies offer tools for organizing and searching remote data repositories on a stand-alone basis, they do not properly address the problem of database and application integration. A solution to this problem must also address the semantic heterogeneity of different data sources, as well as the complexity of collaborative and efficient access to multiple data sources in a distributed environment.

To illustrate these problems, let us consider the scenario of a high-throughput sequencing laboratory that produces sequence fragments which need to be analyzed, labeled, identified, and finally assigned a gene association. Automating this process involves: (a) accessing a LIMS database for new sequencing trace objects or fragments, (b) passing each fragment through the various steps of an analysis pipeline for base calling, trimming and labeling, (c) using a homology search (e.g. BLAST) against one or more public databases to identify homologous sequences, and finally (d) accessing public archival databases, such as the Genome Database (GDB) or the Genome Sequence Database (GSDB), to retrieve genes with which its sequence might be associated. Similar data access and application integration requirements are encountered in a gene-based drug discovery process. These requirements cannot be satisfied easily using existing database technologies. For example, a sequence fragment might be represented using a complex data type (binary) that is not supported by traditional relational DBMS technologies, and manipulated with applications such as PHRED and PHRAP for base calling and trimming. The contents of more than one database might need to be queried together in a semantically meaningful manner. The output of a homology search might need to be processed in order, for example, to extract high scoring hits.

In this paper, we describe tools that provide support for seamless integration of biological applications within a database framework. These tools have been developed in the context of the Object Protocol Model (OPM) and involve using advanced query processing methods for integrated data access and biological computation. These tools allow the extension of database query interfaces to support (i) transparent integration of applications within a single database query, (ii) queries across multiple databases, and (iii) query interfaces on top of applications such as homology searches.

The rest of this paper is organized as follows. In the following two sections, we briefly review OPM and the OPM data management tools, and we present the architecture of the OPM database and application integration platform. Then, we describe two different approaches to application integration in databases. Finally, we review related work and present concluding remarks.

Background

Our approach to exploring biological databases is based on the Object-Protocol Model (OPM) [Chen and Markowitz, 1995]. OPM is an object-oriented data model used for specifying the structure of heterogeneous databases and defining queries against these databases in an abstract, uniform and consistent way.

OPM supports the specification of derived attributes using derivation rules involving arithmetic expressions, aggregate functions such as min, max, sum, avg, count, or compositions of attributes and inverse attributes. OPM also supports derived subclasses and derived superclasses. A derived subclass is defined as a subclass of one or more object classes with an optional derivation condition. A derived superclass is defined as a union of two or more object classes.

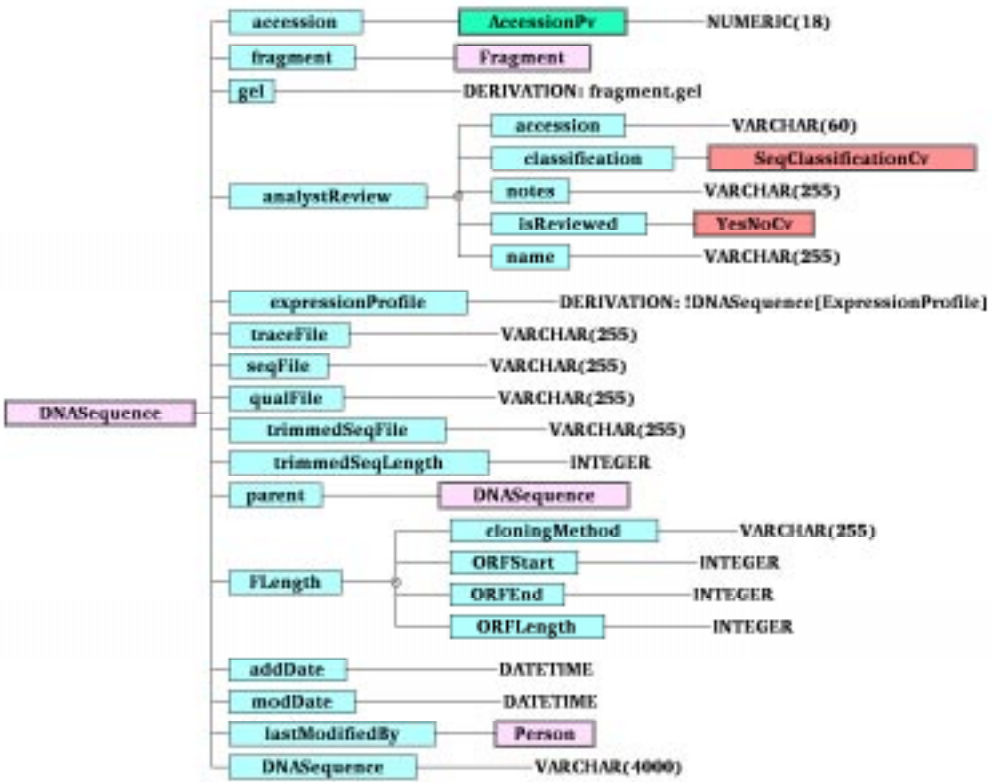


Figure 1: Browsing Classes using the OPM Schema Browser

The "object" part of OPM follows the ODMG standard for object-oriented data models [Cattell, 1996]. Objects in OPM are uniquely identified by object identifiers (oids), are qualified by attributes, and are classified into classes. Classes are organized in subclass-superclass hierarchies and may be grouped into clusters.

Attributes may be simple or consist of a tuple of simple attributes. Attributes can be single-valued, set-valued or list-valued and can be required to have non-null values. If the value class of an attribute is a system-provided data type, or a controlled-value class of enumerated values or ranges, then the attribute is said to be *primitive*. If an attribute takes values from an object class or a union of object classes, then it is said to be *abstract*.

Figure 1 shows part of the OPM schema for a simplified example database, Biotech Laboratory Database (biotechdb for short), underlying the high-throughput sequencing laboratory described earlier. The schema is represented in a diagrammatic notation that may be browsed using the Java-based OPM Schema Browser. This figure shows the object class *DNASequence* with its attributes. For example, *fragment* and *parent* are abstract attributes with value classes *Fragment* and *DNASequence* respectively, while *accession* and *traceFile* are primitive attributes. The attribute *Flength* is a tuple attribute with components including *cloningMethod*, *ORFStart*, *ORFEnd* and *ORFLength*. Attributes *gel* and *expressionProfile* are derived attributes. The class *DNASequence* models

sequence objects that are generated as a result of a sequencing experiment.

In addition to object classes, OPM supports a protocol class construct for modeling scientific experiments. A discussion of the protocol classes is beyond the scope of this paper.

OPM has been extended with Application Specific Data Types (ASDTs). ASDTs are used to model complex, multimedia data types such as DNA sequences, protein structures, genetic maps and gel images. ASDTs are further discussed in Section 4.

The OPM Data Management tools provide facilities for rapidly developing, documenting, and querying biological and other databases [Data Logic, 1998]. The OPM Database Development tools provide facilities for rapidly developing databases using commercial relational database management systems (DBMSs). The OPM Retrofitting tools provide facilities for constructing OPM views for relational and flat-file databases which were not originally developed using the OPM tools. The OPM Database Query tools support querying and exploring databases via OPM views using a high-level query language or Web-based graphical interfaces. The OPM Data Entry tools provide a generic way of entering data into OPM based relational databases. The OPM Database and Application integration tools are discussed in the following sections.

The OPM Query and Application Integration Architecture

The OPM *Multidatabase Query System* (MQS) provides support for querying and exploring multiple heterogeneous databases that have native or retrofitted OPM views. Queries against MQS are expressed in the OPM multidatabase query language, *OPM-MQL*, which is similar to the ODMG standard for object-oriented query languages, OQL. Queries can be submitted to MQS either using a command-line interface, using a CORBA API and wrappers, or using Web-based graphical query tools.

A *Database Directory* is used to store and coordinate information on the databases, DBMSs, inter-database links and Application Specific Data Types (ASDTs) in a multidatabase system. The Database Directory also provides access to meta-data and system information for other applications via a CORBA API.

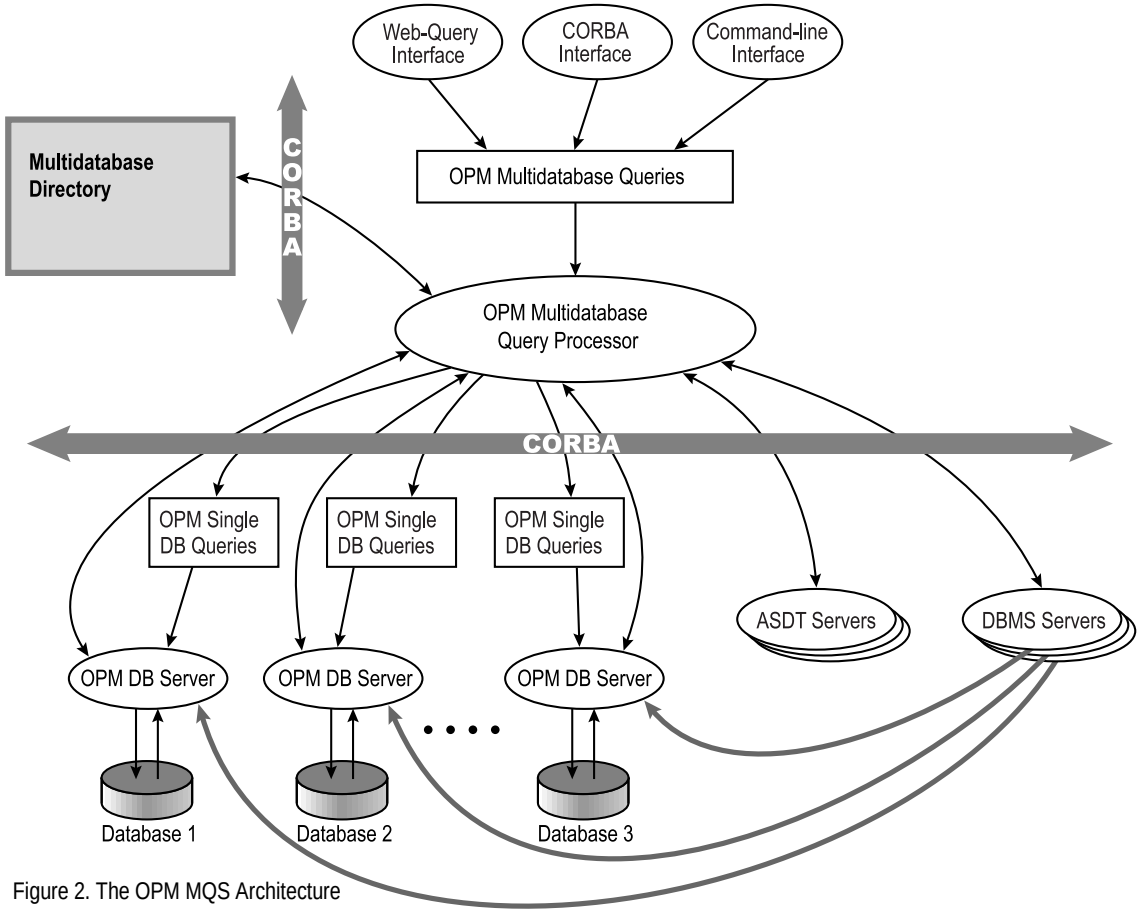


Figure 2. The OPM MQS Architecture

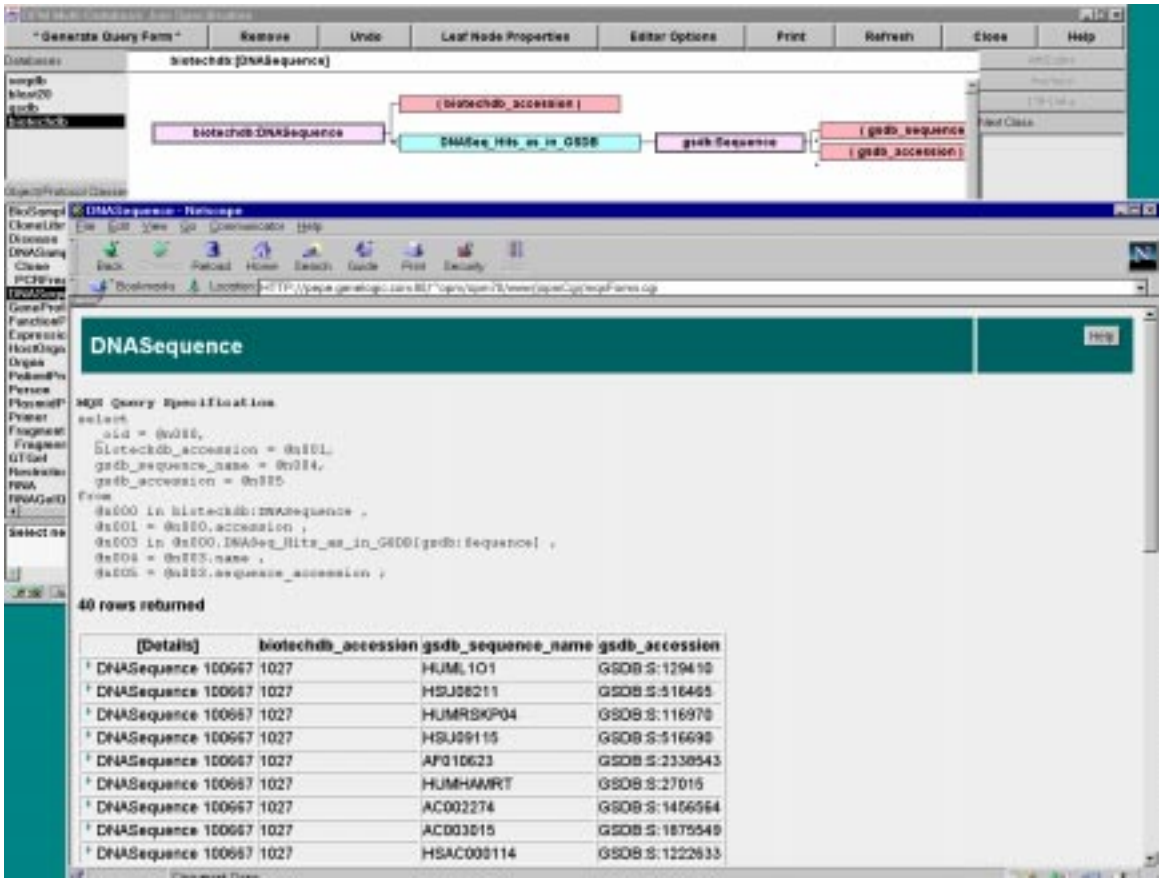
MQS uses a client-server architecture that supports multiple DBMSs and data sources, and allows a multidatabase system to be dynamically reconfigured with new databases, DBMSs and ASDTs. Servers for each DBMS in the system handle database-specific query optimizations and evaluate single-database queries.

The main components of an OPM Multidatabase Query System are shown in Figure 2.

- A central *Multidatabase Query Processor* takes OPM-MQL queries and translates them into expressions in a nested-relational algebra [Buneman et al, 1995b]. This expression involves embedded single database queries. The Multidatabase Query Processor evaluates the expression using a combination of a local query engine and OPM Database Servers.
- OPM *Database Servers* provide database-specific functions used in the translation and optimization of OPM-MQL queries, evaluate single-database queries, and return query results as OPM data structures.
- OPM *ASDT Servers* perform methods for OPM ASDTs and return method results as OPM data structures.
- OPM *DBMS Servers* create and manage the OPM Database Servers. The Multidatabase Query Processor sends requests to the appropriate DBMS server to provide access to an OPM database server associated with a given database. The DBMS Server then checks whether a server for the database is already running, starts up a database server process if necessary, and returns the address of the database server.
- The *Multidatabase Directory* stores information about the databases, DBMSs and ASDTs involved in a federation, and, also about the *inter-database links*, representing known connections between databases.

More details of the Multidatabase Query System can be found in [Kosky et al, 1998].

Figure 3 illustrates a multidatabase query. The query retrieves the name and the GSDB accession number for each of the top BLAST hits stored as annotation of a *DNASequence* in biotechdb. This query has been graphically composed using the Java-based multidatabase query constructor shown in the top section of the figure. The top left window lists the names of all the databases in the multidatabase system. The query is specified by first selecting the database and the class that eventually become the root of the query tree; then the query tree is built by selecting a series of retrieval attributes, abstract attributes or links. Abstract attributes and links specify "joins" within



[Details]	biotechdb_accession	gsdb_sequence_name	gsdb_accession
DNASequence 100667	1027	HUML1O1	GSDB:S:129410
DNASequence 100667	1027	HSJ38211	GSDB:S:516465
DNASequence 100667	1027	HUMRSKP04	GSDB:S:116970
DNASequence 100667	1027	HSJU09115	GSDB:S:516690
DNASequence 100667	1027	AF010623	GSDB:S:2338543
DNASequence 100667	1027	HUMHAMRT	GSDB:S:27015
DNASequence 100667	1027	AC002274	GSDB:S:1456564
DNASequence 100667	1027	AC003015	GSDB:S:1875549
DNASequence 100667	1027	HSAC000114	GSDB:S:1222633

Figure 3. A Multidatabase Query Example

the context of a single database or across databases, respectively.

Inter-database links are schema-level constructs representing known, meaningful connections between databases, while hiding the many low-level details required for connecting the databases. In this example, a link named *DNASeq_Hits_as_in_GSDB*, encodes some conditions under which biotechdb *DNASequence* objects would be related with GSDB *Sequence* objects. These conditions may be arbitrarily complex, incorporating joins, data manipulations, and even references to additional databases and classes. Each link is described by an entry in the database directory, including the link conditions. The entry for the example link might be:

```
LINK DNASeq_Hits_as_in_GSDB
  FROM DB biotechdb
  FROM CLASS DNASequence
  TO DB gsdb
  TO CLASS Sequence
  FROM VARIABLE @_bs
  TO VARIABLE @_ps
  CONDITION
    WHERE @_bs.blastHits.accession =
          @_ps.ic_accession ;
```

The bottom window in Figure 3, displays the OPM-MQL query generated by the graphical query tool and the results of evaluating the query.

Integration through Application Specific Data Types

The Application Specific Data Type (ASDT) concept in OPM is used to model complex, multimedia data types such as DNA sequences, maps or gel images. For example, the following OPM class represents gel objects in biotechdb. A gel object is qualified by a number of attributes, including: a sample set; the project to which it belongs; the plate it is loaded from; the processing date and the person who conducted the experiment; the name of the directory where the gel data is stored; and an image that holds its content. The attribute image takes as its value an instance of the ASDT *GelImageASDT*.

```
OBJECT CLASS Gel
  ID: gelId
  ATTRIBUTE gelId: INTEGER REQUIRED
  ATTRIBUTE samples: set-of Sample REQUIRED
  ATTRIBUTE project: Project REQUIRED
  ATTRIBUTE gelPlateId: INTEGER REQUIRED
  ATTRIBUTE preparedBy: Person REQUIRED
  ATTRIBUTE prepareTime: TIMESTAMP OPTIONAL
  ATTRIBUTE datadir: VARCHAR(128) OPTIONAL
  ATTRIBUTE image: GelImageASDT OPTIONAL
  ATTRIBUTE comments: VARCHAR(255) OPTIONAL
```

Primitive attributes are easily handled by databases because their data types and most of the standard operations on them are supplied directly by the underlying DBMS. Complex data types such as images are not handled adequately by pure relational database management systems because of the limitations of the data types provided and SQL's lack of support for user-defined, type-specific operations. The ASDT construct allows OPM databases to treat complex data types as primitive types; that is, to store, retrieve and perform content-based operations on complex data such as gel images. For example, one can apply complex image processing algorithms to a gel image with the same ease as adding two numbers.

An ASDT's methods model the application specific computations that are applicable to instances of the ASDT. Methods support type-specific or content-based manipulations of complex objects. For example, an image object could be cropped, displayed or searched for a specific pattern of pixel intensities at certain positions. In the OPM system, any ASDT instance is represented as an atomic value that can be explored through its methods. Method calls can be specified within an OPM query, and are recognized by the query processor, which will call the appropriate application server to execute them. Methods can appear in the `SELECT`, `FROM`, and `WHERE` clauses of an OPM query. For example, an OPM query that retrieves a set of gel images satisfying certain criteria, and cuts out and displays the first lane (perhaps the quality control lane) of the images, might be:

```
SELECT label = @g.gelId,
       displ = @g.image.get_lane(1).display()
FROM   @g IN Gel
WHERE  @g.preparedBy.initials = "TT" AND
       @g.preparedTime = "7/23/1998" AND
       @g.image.avg_intensity() < 500 ;
```

Without ASDTs, the above example would require us to store images in the database as binary objects, (or the references to them), and develop a client application program that queries the database to select a set of images and then applies proper algorithms to each member of the set. The pitfalls of this approach include network traffic overhead, since the entire image files need to be shipped from the server to the client, and client programs that do the bulk of the work and need to be maintained for a number of client sites.

Another alternative would be to preprocess all the images ahead of time, and extract explicit attributes for average intensity, lanes, etc., that are stored in the database. This alternative also has disadvantages: the schema and database need to reflect the types of data analysis employed at a given time, and frequent changes of the analysis will cause schema/data revisions and version

management overhead. The advantage of using ASDTs is that data processing is done under the supervision of the OPM query processor. This processing takes place on the server side.

ASDTs are comparable to data blades, cartridges and extenders supported by Object-Relational Database Management Systems (ORDBMSs), but are a middleware solution, which may utilize the facilities provided by an ORDBMS, or provide object-relational functionality on top of pure relational databases.

The design of an OPM ASDT consists of three stages:

1. Defining the ASDT using the OPM specification language. The description of an ASDT requires the identification of its methods, and specifying the signatures (interfaces) for each method.

2. Finding or developing code that implements the methods.

3. Building a CORBA server for the ASDT that functions as the method execution engine. The MQS query processor dispatches method calls for the ASDT to this server. This last step is supported by code templates provided by the OPM application development environment, where the application developer can plug-in custom code.

In the following example, we demonstrate the development and use of *SeqASDT*, an ASDT which provides sequence analysis utilities to the *Sequence* class of database seqdb (also part of the mutlidatabase system of Figure 3). The following statements illustrate the ASDT declaration in OPM, and the schema definition of class *Sequence*. The ASDT declaration includes the underlying data type for its instances, its storage mode and a complete enumeration of its methods, including their signatures. In this example sequence ASDT instances are stored in the database as long strings. Alternatively, they could be stored internally as text large objects (`OPM_BLOB/TEXT`), or outside of the database in which case the underlying data type would be a string storing a "reference" to the external object.

```
APPLICATION SPECIFIC DATA TYPE seqASDT
  DATATYPE: VARCHAR(2000)
  STORED: INTERNAL
  METHOD toFasta
    SIGNATURE: "string toFasta(string,string)"
    LANGUAGE:  "C++"
    DESCRIPTION: "translates sequences to
                  Fasta format"
  METHOD blastn1
    SIGNATURE: "string blastn1(string)"
    LANGUAGE:  "C++"
    DESCRIPTION: "blasts sequence against the
                  db specified by arg1"

OBJECT CLASS Sequence
  ID: seqID
  ATTRIBUTE seqID: VARCHAR(100) REQUIRED
  ATTRIBUTE sequence: seqASDT OPTIONAL
  ATTRIBUTE dateIn: DATETIME OPTIONAL
  ATTRIBUTE rawSeq: VARCHAR(2000) OPTIONAL
```

Methods *toFasta* and *blastn1* are implemented using a programming language of choice, and added to the *SequenceASDT* server. This server communicates with the MQS query processor via a CORBA interface. Subsequently, the server needs to be registered in the MQS database directory. This is done by adding an entry, such as the following, to the database directory file:

```
ASDT_SERVER seqASDT
  SERVER  "sequenceAsdtServ2"
  HOST    "pepe.genelogic.com"
  COMMENT "server implements seqASDT methods"
```

When a method call occurs inside an OPM query, the MQS query processor evaluates the method-free query using the appropriated database servers, and then calls the ASDT server to execute method calls based on the intermediate query results. Below, we illustrate a method call within a query to BLAST all the sequences in the database against a blast-able database. Extending queries with the capability to call user-defined methods allows us to integrate

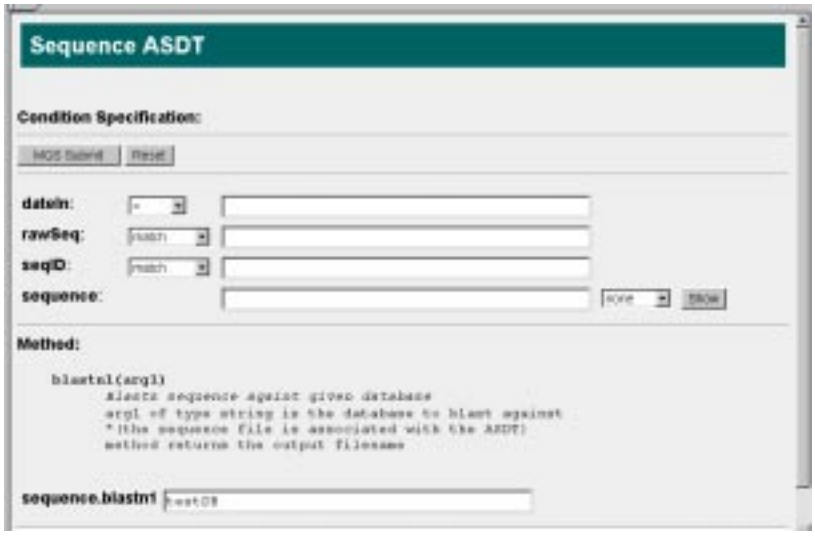


Figure 4: Query form containing a method call

application specific computations in queries, such as calling blast on a set of sequence objects directly as they are retrieved from the database. Figure 4 shows the single class query form generated automatically by the OPM query tools, extended with methods and fields to specify method arguments. The bottom part of the query form provides a field to specify the parameters for the method call.

ASDTs extend the OPM type system, and allow applications to be called from inside OPM queries. However there are cases where one would like to treat entire applications as query-able databases returning results that can be treated as OPM query objects. This approach to application integration is studied in the following section.

Application Servers

OPM provides support for "wrapping" biological applications of interest into application servers that can be included into an OPM multidatabase query system. In this section we describe an example of this mechanism as it applies to homology search engines.

An application server for homology search engines allows accesses to databases and homology searches to be integrated into a single OPM query. An example of such a query is: "retrieve a fragment from the biotechdb and find all its homologous sequences in dbEST that are longer than 300 base pairs and have a p-value less than .001". In order to process this query, we must first query biotechdb, then pass the query results through a homology search engine such as BLAST, and finally apply the remaining query conditions to the BLAST results. The OPM *BLAST server*, using BLAST as the back-end search engine, allows us to perform homology searches from within an OPM query, and to perform further processing (e.g., selecting or projecting) on BLAST output data. Note that, in addition to being part of a multidatabase system, OPM application servers can be directly accessed using the OPM query language. e.g., using the BLAST server, one may blast a specific sequence against a blast-able database and retrieve only accession ids and p-values of hits with length larger than 1000 base pairs.

The development of a server for an application like BLAST involves modeling the application as an OPM database. This includes modeling a BLAST call, as well as the output of the call, as OPM object classes. The difference between such a server and a database server is that the database server relies on a database engine to access data, while for the BLAST server, the data access is supported by one of the BLAST programs. A BLAST application needs to be "wrapped" in order to make an interface that supports the OPM query language. Building such an interface requires defining a mapping between the constructs of the BLAST OPM view and the retrieval operations of the underlying BLAST application. The

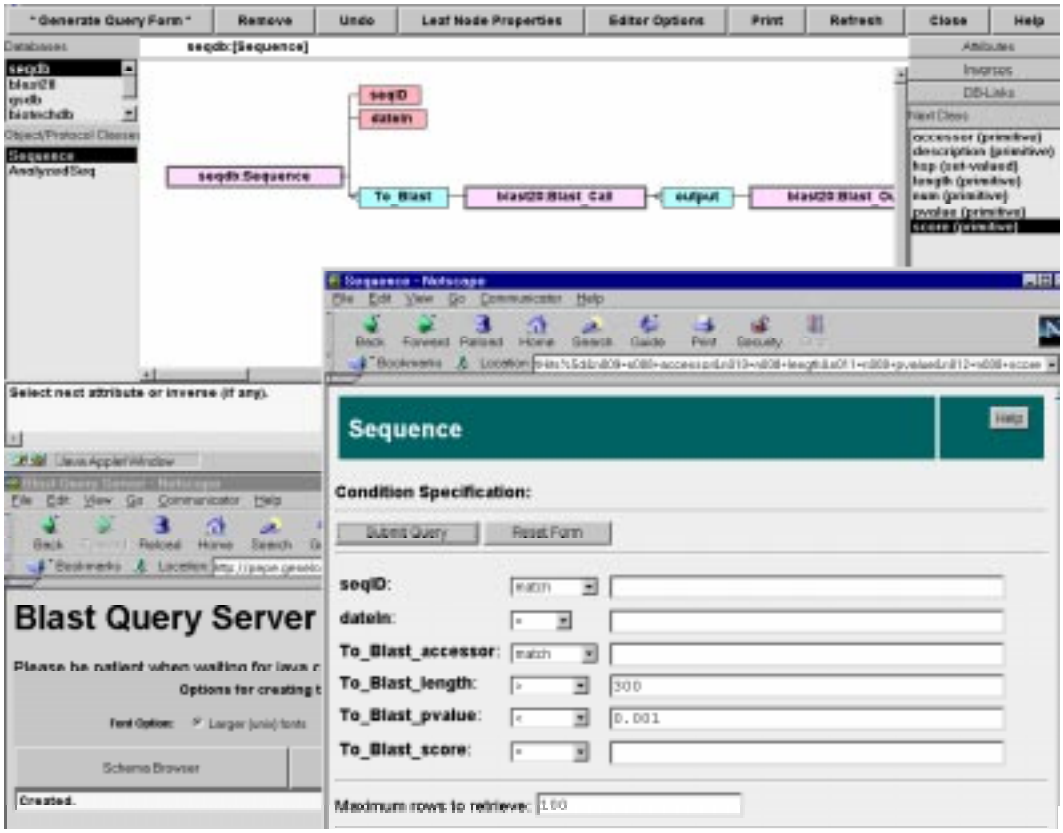


Figure 5: Multidatabase query addressing the OPM BLAST server

result is a server that takes OPM queries, generates a BLAST call, intercepts the results, fits them in an OPM schema, and applies further query processing operations to them.

The example query mentioned above, can be expressed by the following OPM-MQL query:

```
SELECT  l = @r.fragId,
        a = @bo.hits.accessor
FROM    @r in biotechdb:Fragments
        @bc in blast20:Blast_Call
        @bo in bc.output
WHERE   @r.finished = "today" AND
        @bc.querySeq = @r.sequence AND
        @bc.command = "blastn" AND
        @bc.dataSource = "dbEST" AND
        @bo.hits.length < 300;
```

This multidatabase query can be further refined. For example, several of the `WHERE` clause conditions dealing with parameters of the BLAST call can be grouped into an inter-database link, thus hiding the details of the BLAST call:

```
LINK toBlast
  FROM DB biotechdb
  FROM CLASS Fragment
  TO DB blast20
  TO CLASS Blast_Call
  FROM VARIABLE @f_local
  TO VARIABLE @c_blast
  COMMENT "Hide complexity of a blast call"
  CONDITION
    WHERE
      @f_local.sequence = @c_blast.query and
      @c_blast.command = "blastn" and
      @c_blast.dataSource = "dbEST";
```

Such a link can be used in a query like a conventional attribute, making the access to the BLAST server transparent to the user. For example, using inter-database links, the previous query could be written:

```
SELECT  l = @r.fragId,
        a = @bo.hits.accessor
FROM    @r in biotechdb:Fragments
        @bo in r.toBlast.output
        @h = @bo.summary.sequence
WHERE   @r.finished = "today" AND
        @bo.hits.length > 300;
```

The query evaluation steps are as follows. First, a single database query is evaluated by the OPM query server for biotechdb. Second, for each retrieved sequence, a BLAST call/query is created and submitted to the BLAST server. Third, the server invokes the BLAST program and creates an output that fits the OPM schema for BLAST results, which is subsequently passed back to the MQS query processor. Finally, post-processing takes place in MQS to evaluate predicates on the BLAST results.

Figure 5 illustrates how the same query can be expressed using the OPM Java query constructor tool and the multi-database query system introduced in Figure 3. Figure 6 presents sample results for the query, which clearly demonstrate the advantages of applying query processing to BLAST output. Instead of the long, incomprehensible BLAST reports for each fragment, the BLAST server query only displays qualified and relevant components of the output.

Sequence Help

18 rows returned

[Details]	seqID	dateIn	To_Blast_accessor	To_Blast_length	To_Blast_pvalue	To_Blast_score
Sequence 1	11	01/05/1999 00:00:00	AF044459	321	0.0001	195
Sequence 1	11	01/05/1999 00:00:00	AF044450	321	0.0003	186
Sequence 1	11	01/05/1999 00:00:00	AF044461	321	0.0003	186
Sequence 1	11	01/05/1999 00:00:00	AF044451	318	0.0003	186
Sequence 1	11	01/05/1999 00:00:00	AF044455	318	0.0003	186
Sequence 1	11	01/05/1999 00:00:00	AF044456	318	0.0003	186
Sequence 1	11	01/05/1999 00:00:00	AF044458	318	0.0003	186
Sequence 1	11	01/05/1999 00:00:00	AF044460	306	0.0003	186
Sequence 1	11	01/05/1999 00:00:00	AF044462	318	0.0008	177
Sequence 4	22	01/06/1999 00:00:00	AF044459	321	0.0001	195

Figure 6: Results of the multidatabase query from Figure 5

Conclusions

In this paper, we described tools for seamless integration of biological applications into a database framework. In particular, we described two alternatives: (a) a "light-weight" application integration based on Application

Specific Data Types (ASDTs) and (b) a "heavy-duty" integration of analytical tools based on mediators and wrappers. These methodologies were developed in the context of the OPM Multidatabase Query System, a middleware architecture that allows queries across multiple data sources and applications.

The two methodologies address quite different application integration needs. The "light-weight" approach extends the query language with user-defined functions that allow us to perform data-specific computations from within queries. The extension capability is built inside the query processor, therefore the amount of effort needed to develop a new ASDT, and hence perform application integration, is minimal.

The "heavy-duty" integration provides fine granularity control over input parameters and access to the output of certain applications. In this approach, the application programs are turned into "servers" able to respond to queries. In addition, application servers participate in the query planning and optimization performed by the MQS query processor, while ASDTs do not. The effort involved in developing an application server is substantially larger than that necessary to implement an ASDT.

Few systems attempt to support comprehensive access to multiple heterogeneous biological data sources. Some of them, including SRS [Etzold and Argos, 1993], emphasize data retrieval rather than application integration. while others, such as Kleisli [Buneman et al, 1995a], do not provide any schema-level support for query-construction or exploration of databases and applications. Another category of systems deals with application integration at the user interface level. The Wisconsin Package [GCG] is a system that brings together several analytical tools that can inter-operate in the very loose sense of the term, i.e., through files of a common format. Several public services such as GSDB at NCGR [GSDB], provide web-based interfaces that allow data to be passed to several applications. These interfaces allow processing data one object at a time, and are based on using pre-specified paths and interfaces. TAMBIS [Baker et al, 1998] is a noteworthy system that intends to provide transparent access to biological databases and analysis tools. TAMBIS adopts a mediator/wrapper architecture for data access and application integration similar to that described in this paper, but it relies on a knowledge-based interface that hides the data sources.

A similar approach to ours for providing access to legacy sources through database middleware is followed by the Garlic project [Roth and Schwarz, 1997]. Ideas originating in Object-Relational Database Management Systems [Stonebraker 1995], best represented today by products such as IBM's DB2, Oracle 8 and Informix's Universal Server, have been catalytic for the research and development of ASDTs in OPM. The key difference is that ASDTs can be defined on top of non Object-Relational DBMSs. This characteristic also differentiates ASDTs from the Enhanced-ADTs in PREDATOR [Seshadri et al, 1997] and objects with methods in Object-Oriented dababases.

Finally, a lot of optimism for integrated bioinformatics systems has been generated by the OMG life science committee (LSR), regarding how CORBA offers a coherent framework in which independent data sources and their service are easily accessible. Our work is an excellent case study to show how CORBA can be useful. We extensively used CORBA (a) for client-server communication, (b) to wrap programs/servers implemented using different technologies, and (c) to achieve a flexible, scalable, and reconfigurable system architecture. However we have not found CORBA/IDL to be suitable for specifying the semantics of database interfaces or biological applications. This is partly because IDL is intended only to specify syntactic interfaces of applications, the semantics of which are hidden in their implementation, and partly because of limitations on the data-types supported by IDL.

The contributions of this work include extensions to the OPM data model and query language, to support methods defined on ASDTs. In addition, ASDTs, as a mechanism to integrate databases with applications, achieves similar expressive power to that of Object Relational Data Management Systems (ORDBMS) in legacy and relational environments. Finally, we demonstrated novel ways for "wrapping" biological applications of interest into application servers that can be included in a multidatabase query processing system.

References

Baker, P.; Brass, A.; Bechhofer, S.; Goble, C.; Paton, N.; and Stevens, R. 1998. TAMBIS – Transparent Access to Multiple Bioinformatics Information Sources. In Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB'98).

Buneman, P.; Davinson, S.; Hart, K.; Overton, C.; and Wong, L. 1995a. A Data Transformation System for Biological Data Sources. In Proceedings of VLDB 1995.

Buneman, P.; Naqvi, S.; Tammen, V.; and Wong, L. 1995b. Principles of Programming with Complex Objects and Collection Types. *Theoretical Computer Science*, 149, pp. 3-48.

Chen, I.A., and Markowitz, V.M. 1995. An Overview of the Object-Protocol Model and OPM Data Management Tools. *Information Systems* Vol. 20, No. 5.

Chen, I.A.; Kosky, A.S.; Markowitz, V.M.; and Szeto, E. 1997. Constructing and Maintaining Scientific Database Views. In Proceedings of the 9th International Conference on Scientific and Statistical Database Management, Hansen, D. and Ioannidis, Y. (Eds), pp. 237-248.

Chen, I.A.; Kosky, A.S.; Markowitz, V.M.; Szeto, E.; and Topaloglou, T. 1998. Advanced Query Mechanisms for Biological Databases. In Proceedings of the 6th International Conference on Intelligent Systems for Molecular Biology (ISMB'98).

Data Logic. 1998. Data Management and Integration Tools for Bioinformatics. Technical White Paper, http://www.genelogic.com/opm.htm

Etzold, T., and Argos, P. 1993. SRS, An Indexing and Retrieval Tools for Flat File Data Libraries. *Computer Applications of Biosciences*, 9, 1, pp. 49-57. See also http://www.embl-heidelberg.de/srs/srsc.

The Wisconsin Package. Genetics Computer Group (GCG), Madison, Wisconsin.

Genome Sequence DataBase (GSDB). The National Center for Genome Resources. http://www.ncgr.org/gsdb

Object Management Group. Life Sciences Research. http://www.omg.org/homepages/lsr

Kosky, A.S.; Chen, I.A., Markowitz, V.M.; and Szeto, E. 1998. Exploring Heterogeneous Biological Databases: Tools and Applications. Proceedings of the 6th International Conference on Extending Database Technology (EDBT'98), Lecture Notes in Computer Science Vol. 1377, Springer-Verlag, 1998, pp. 499-513.

Cattell, R. G. G. (ed) 1996. The Object Database Standard: ODMG-2.0. Morgan Kaufmann.

Roth Tork, M., and Schwarz, P. 1997. "Don't Scrap it, Wrap it! A Wrapper Architecture for Legacy Data Sources. In Proceedings of the 23rd VLDB Conference.

Seshadri, P.; Linvy, M.; and Ramakrishnan, R. 1997. The Case for Enhanced Abstract Data Types. In Proceedings of the 23rd VLDB Conference.

Stonebraker, M. 1995. Object-Relational DBMS: The Next Wave. Morgan Kaufmann,.