Exploring Heterogeneous Biological Databases: Tools and Applications *

Anthony S. Kosky, I-Min A. Chen, Victor M. Markowitz, and Ernest Szeto

Bioinformatics Systems Division, Gene Logic Inc.
2001 Center Str, Suite 600, Berkeley, CA 94704
e-mail: [anthony,ichen,vmmarkowitz,szeto]@genelogic.com

Abstract. We present a tool-based strategy for exploring heterogeneous biological databases in the context of the Object-Protocol Model (OPM). Our strategy involves tools for examining the semantics of biological databases; constructing and maintaining OPM views for biological databases; assembling biological databases into an OPM-based multidatabase system, while documenting database schemas and known links between databases; supporting multidatabase queries via uniform OPM interfaces; and assisting scientists in specifying and interpreting multidatabase queries. We describe these tools and discuss some of their scientific applications.

1 Introduction

Exploring data across biological databases entails coping with their distribution, the heterogeneity of their underlying systems, and their semantic heterogeneity. These databases are defined using a variety of data models and notations, such as relational and object-oriented models, and the ASN.1 data exchange notation; they are implemented using different systems, including relational database management systems (DBMSs), and various indexed flat-file systems; they are based on different views of the biological domain; and they contain different and possibly conflicting data.

Strategies for exploring heterogeneous databases range from loose to tight integration of component databases, with distributed or centralized data access [13]. Tight integration of heterogeneous databases entails constructing a *global* schema representing a consistent integrated view of all component databases expressed in a common data model, while loose integration does not require constructing such a global schema.

Constructing a global schema over biological heterogeneous databases is hindered by numerous semantic conflicts between schemas of component databases.

* The work presented in this paper was carried out while the authors were affiliated with the Lawrence Berkeley National Laboratory, with support provided 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. This paper has been issued as technical report LBNL-40728.

Often the same concept (e.g., *gene*) is represented in different biological databases using synonyms, alternative terminology, or different data structures. Resolving semantic conflicts among biological databases has been attempted by systems such as IGD [12], but only to a limited extent: these systems detect and resolve only simple schema and data conflicts, such as name and object identification conflicts. A more systematic resolution of such conflicts would require the involvement of numerous groups worldwide,[1] and would be extremely difficult to achieve, if at all possible.

A more feasible approach involves employing multidatabase querying mechanisms in a loosely integrated framework. Such mechanisms support construction of queries over component databases, where a query explicitly refers to the elements of each database involved. Component databases of multidatabase systems can be queried by constructing functions that access the databases in their native format, as done in Kleisli [1], or can be accessed through views in a common data model, as entailed by the strategy we propose. Kleisli is a query system based on the nested relational data model, which is a purely "value-based" model and, as such, does not support any concept of schemas or integrity constraints. This approach makes it relatively simple to add new databases to a multidatabase system, since it is not necessary to convert schemas or construct new views of a database, but does not provide the support or documentation necessary to formulate queries or to interpret their results. Consequently, in order to construct ad hoc multidatabase queries, a programmer must have expert knowledge of each database involved in the query, its semantics and its data model, as well as of CPL, the query language employed by Kleisli.

Providing an efficient, extensible mechanism for evaluating multidatabase queries is not in itself sufficient: in addition it is necessary to provide support for exploring and documenting multiple heterogeneous databases via a uniform model and interface, and to aid non-expert users in formulating meaningful multidatabase queries. Such an approach requires an initial investment of time and effort in developing database views in a common notation and in constructing a directory of information (metadata) needed to assist users in constructing and interpreting multidatabase queries. This initial effort is particularly important for the exploration of biological databases which are not documented in a consistent and comprehensive way [9]: biological database documentation is often incomplete, sometimes hard to access (e.g., hidden in large files), and requires knowledge of the notation of the underlying data management system (e.g., relational, object-oriented, ASN.1 data exchange format).

Our strategy for exploring heterogeneous databases uses the Object-Protocol Model (OPM) [3] as the common data model; takes advantage of a suite of existing OPM tools; and involves developing several additional tools. Each of these tools can be used independently and therefore represents a valuable resource in its own right.

[1] According to a recent issue of Nucleic Acids Research (vol 24, no. 1, 1996), there are over 50 important molecular biology databases.

Each database in an OPM-based multidatabase system is associated with a native OPM schema or a retrofitted OPM view. Databases with native OPM schemas are developed using the OPM Database Development tools, that provide facilities for translating OPM schemas into complete database definitions for a variety of commercial DBMSs. Databases developed without the OPM tools can be retrofitted with OPM views using the OPM Retrofitting tools [6].

The OPM Multidatabase tools provide additional facilities for assembling heterogeneous databases into a multidatabase system, together with information on known links between databases; for browsing schemas and metadata for the multidatabase system via Web-based interfaces; and for specifying and evaluating multidatabase queries via uniform OPM query interfaces.

The remainder of this paper is organized as follows. OPM is briefly reviewed in section 2. The construction of OPM views and the structure of the Multidatabase Directory are described in section 3. The multidatabase query system is described in section 4. In section 5 we present the Web interfaces for exploring multidatabase systems. Section 6 presents an application of the OPM Multidatabase tools to a molecular biology multidatabase system. Section 7 contains concluding remarks.

2 The Object Protocol Model

We use the Object Protocol Model (OPM) to provide an abstract and uniform representation of heterogeneous databases. OPM provides extensive schema documentation facilities in the form of descriptions, examples and user-specified properties that can be associated with schema elements. Existing OPM tools provide facilities for developing and accessing databases defined using OPM, for constructing OPM views for existing relational databases and structured files, for representing database schemas using alternative notations, and for querying databases through uniform OPM views. We briefly review below the main features of OPM; details can be found in [3].

OPM is a data model whose object part closely resembles the ODMG standard for object-oriented data models [10]. Objects in OPM are uniquely identified by object identifiers (oids), are qualified by attributes, and are classified into classes. Classes can be organized in subclass-superclass hierarchies and can be classified into *clusters*. In addition to object classes, OPM supports a protocol class construct for modeling scientific experiments. Protocol classes are not discussed in this paper.

Attributes can be *simple* or consist of a *tuple* of simple attributes. An attribute can have a single value, a set of values, or a list of values. If the value class (or domain) 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 displays examples of classes involved in the OPM schemas for the Genome Sequence Database (GSDB)[2] and the Genome Database (GDB),[3] representing nucleic acid sequences in GSDB and GDB respectively. For example, attribute `confidences` of class `Sequence` is set-valued, while attribute `release` is single valued; attribute `confidences` is a tuple attribute consisting of five component attributes, `loc_start`, `loc_end`, `confidence`, `multiple_read`, and `double_stranded`; component attribute `sp_id` of tuple attribute `sequence_pieces` of class `Sequence` is an abstract attribute taking values from class `Sequence`, while attribute `seq_order` of the same tuple attribute is a primitive attribute.

OPM supports the specification of *derived attributes* using derivation rules involving arithmetic expressions, aggregate functions (`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.

3 Constructing Biological Multidatabase Systems

Our strategy for exploring heterogeneous biological databases involves (1) constructing OPM views for databases developed with or without OPM; (2) assembling databases within an OPM-based multidatabase system, while documenting the databases and known links between them; and (3) expressing, processing, and interpreting queries over multidatabase systems. In this section we describe the first two stages.

3.1 Constructing OPM Views for Existing Databases

A database designed using OPM can be implemented with a commercial relational DBMS, such as Sybase or Oracle, using the OPM Schema Translator [4]. This tool automatically generates complete definitions for the underlying relational DBMS, including rules and constraints required for maintaining data integrity, while creating a *mapping dictionary* that records the correspondences between the classes and attributes of an OPM schema and the underlying relational tables.

Existing databases implemented as flat files (defined using a parsable notation such as ASN.1) or as relational databases (developed with or without OPM), can be associated with one or more *OPM views* constructed using the OPM Retrofitting tools [6]. The procedure for constructing OPM views for existing databases consists of first automatically generating a *canonical* OPM view from the database definition, where no additional assumptions are made about

[2] `http://www.ncgr.org/gsdb/`
[3] `http://gdbwww.gdb.org/gdb/schema.html`

the structure of the underlying database, and then iteratively changing or refining the OPM view using schema restructuring operations, such as renaming and/or removing classes and attributes, merging and splitting classes, adding or removing subclass relationships, defining derived classes and attributes, and so on. The mapping between OPM constructs and the native database constructs is recorded in a mapping dictionary. Refining an OPM view results in changes to the OPM view definition and corresponding changes in the mapping dictionary, but has no effect on the underlying (relational or flat file) database.

3.2 The Multidatabase Directory

The information core of an OPM-based multidatabase system is a *Multidatabase Directory* which contains metadata on component databases and systems needed by the Multidatabase Query System as well as supporting documentation for exploring and using a multidatabase system. Information in a Multidatabase Directory includes database names, descriptions of the purpose of the databases, information on connections to other related databases in the same multidatabase system, and database schemas or views specified in OPM, as well as information specifically required by the multidatabase query system, discussed in section 4.4. Of particular importance is the information about known links between classes in different databases, including a description of the semantics of the links. From the perspective of a user exploring an OPM based multidatabase system, inter-database links connect classes in distinct databases in the same way that OPM abstract attributes connect classes within a single database.

The (native or retrofitted) OPM schema or view of a database is used as its reference (common) representation in a Multidatabase Directory; in addition, the Directory contains alternative representations for the database schemas using various notations and data models, including the native data model of the database, an Extended Entity-Relationship (EER) model, an abstract, DBMS-independent relational model, and the ASN.1 data exchange notation [11]. A Multidatabase Directory is stored and maintained using structured flat files, with automatically generated HTML versions supporting exploration using Web browsers.

The OPM views of databases in a multidatabase system can be represented using a diagrammatic notation and browsed with a Java based OPM Multidatabase Schema Browser. Figure 1 shows an example of using the OPM Multidatabase Schema Browser for browsing the schemas of GDB and GSDB, where the inter-database links are displayed as abstract OPM attributes and are used for crossing database boundaries (e.g., `GSDB_to_GDB_sequence`).

4 The Multidatabase Query System

The OPM Multidatabase Query System provides support for retrieving and combining data from multiple heterogeneous databases via queries expressed across the OPM views of the component databases. The main components of the system are shown in figure 2:

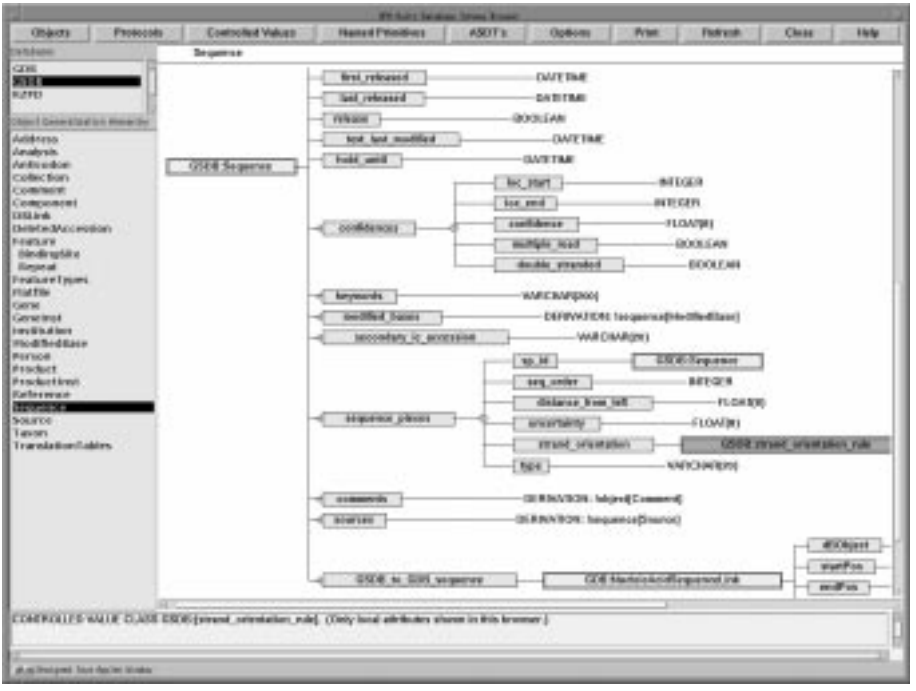

Fig. 1. Browsing Multidatabase Views with the OPM Multidatabase Schema Browser

1. OPM Database Servers provide database specific query translation facilities and a common interface for executing queries for each database involved in the multidatabase system.
2. A central OPM Multidatabase Query Processor interprets OPM multidatabase queries, generates queries for individual databases, performs local data manipulations necessary to combine the data retrieved from individual databases, and provides functionality that is not supported by the underlying DBMSs or file systems.
3. Information on component databases, DBMSs, and inter-database links is recorded in a Multidatabase Directory which is used by the OPM Multidatabase Query Processor in order to translate and plan the execution strategies for queries.
4. Queries can be specified using a Multidatabase Web Query Interface or as textual queries written in the OPM multidatabase query language, OPM*QL.

The Web-based query interfaces do not offer the full expressive power or flexibility of OPM*QL (e.g., for handling aggregates or nested sets), but are considerably easier and more intuitive for non-expert users, are convenient for rapid data exploration, and are sufficiently powerful to express a large class of common queries. OPM*QL provides expert users with the ability to formulate

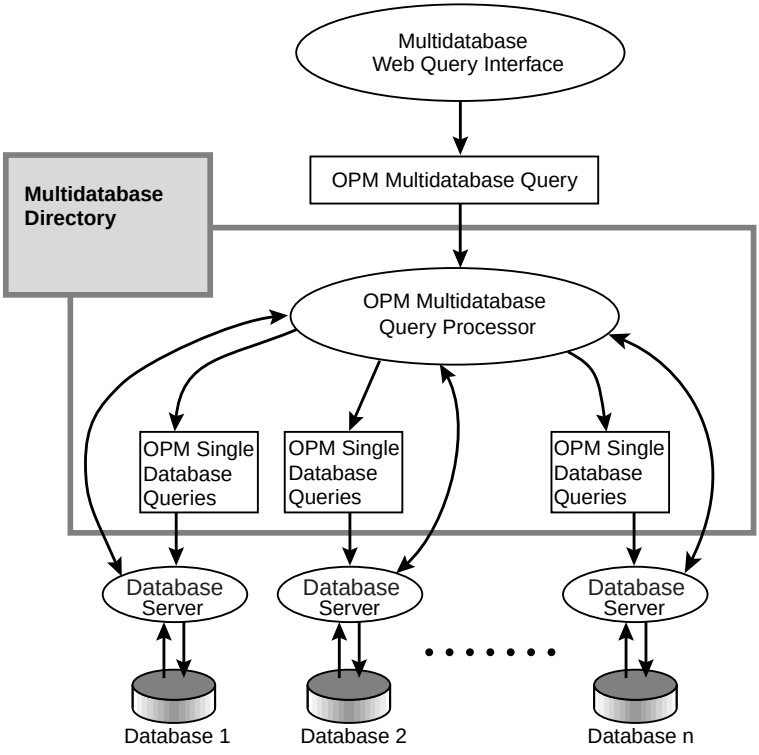

Fig. 2. The Architecture of the OPM Multidatabase Query System

very general queries across multiple heterogeneous databases using a single data model and language. The Multidatabase Web Query Interface will be described in section 5.

4.1 The OPM Multidatabase Query Language

The OPM Multidatabase Query language, OPM*QL, is an object-oriented query language similar to OQL, the ODMG standard for object-oriented query languages [10]. OPM*QL extends the OPM single database query language, OPM-QL [5], with constructs needed for accessing multiple databases and for taking advantage of the additional power offered by the query processing capabilities of the OPM Multidatabase Query Processor. In addition, OPM*QL allows using inter-database links defined in the Multidatabase Directory, in a similar manner to regular abstract attributes. These links hide many of the low-level details involved in formulating multidatabase queries, and simplify the task of finding meaningful connections between objects in distinct databases.

For example, the following OPM*QL query, for finding *protein kinase genes on chromosome 4* is expressed across two databases, GDB and GSDB:

```
SELECT gene = z.displayName
FROM   x in GSDB:Product,
       y in x.!object[DBLink],
       z in y.GSDB_to_GDB_gene
WHERE  x.authorative_name MATCH "%protein kinase%"
  AND  z.!segment[MapElement]map.chromosome.searchAlias = "4";
```

In the query above, "`GSDB:Product`" refers to class `Product` of `GSDB`, while "`GSDB_to_GDB_Gene`" refers to a link from GSDB to GDB recorded in the Multidatabase Directory.

OPM*QL can also be used to express single database queries. However, unlike the single-database query language, OPM-QL, it is not restricted by the need to translate queries into equivalent queries expressed in the query language of the underlying DBMS, for example into DBMS-specific versions of SQL. Consequently, OPM*QL provides uniform and powerful query facilities for any database developed or retrofitted with OPM, irrespective of the query facilities of the underlying DBMS or file system.

4.2 Query Processing Strategy

The evaluation of a query using the OPM Multidatabase Query Processor involves two stages:

1. A query (input) is analyzed in order to form an execution plan. This process involves determining the single-database *subqueries* to be evaluated by the individual OPM Database Servers, and generating a nested-relational algebra expression that incorporates the single database queries together with those parts of the query that need to be evaluated locally. The nested relational algebra used is similar to that proposed in [2] but with certain modifications to make it better suited to evaluating OPM queries.
2. The nested-relational expression is evaluated, with the embedded single-database OPM queries being passed to the appropriate OPM Database Servers for execution.

Query evaluation follows a "semi-join" approach: remote queries are evaluated in sequence, with the results of each query being used to constrain the next queries. Currently the Multidatabase Query Processor uses the structure of the OPM*QL query in order to determine the order in which subqueries are evaluated. In the future, we plan to develop heuristics for estimating the cost of individual subqueries and the size of their results in determining an optimal evaluation order.

In general, as many of the query conditions as possible are incorporated into the subqueries that are executed remotely. However the evaluation of these subqueries depends both on the underlying DBMS and on the metadata for a particular database. For example, various relational DBMSs support different dialects of SQL, so that the subsets of OPM-QL that can be translated into SQL may differ. Further, for an indexed flat file database, only attributes that

have been indexed can be involved in a condition that is used in a subquery. Consequently, each OPM Database Server provides the functions necessary to determine which parts of a subquery must be evaluated locally and which can be passed to a remote database server, and to interpret the results of the remotely executed query.

The individual OPM Database Servers provide results in the format best suited to their underlying DBMS. For example relational DBMSs provide results as flat tables, while object oriented systems can use objects or nested data-structures. Nested data-structures are more efficient in terms of the size of the data-structures that are returned, but interpreting them requires more local computation.

The Multidatabase Query Processor has the ability to compute aggregate and arithmetic functions, as well as general manipulations of nested relational data, and consequently is able to process the same queries over DBMSs with both limited and powerful query facilities.

4.3 The OPM Database and DBMS Servers

An OPM DBMS Server is required for each DBMS or file access system involved in a multidatabase system. An OPM DBMS Server manages all the OPM Database Servers for databases implemented using that DBMS, including creating database servers as they are needed, keeping track of which databases have servers currently in use, and closing down Database Servers that are no longer needed.

The architecture of the Multidatabase System allows new DBMS Servers to be added to or removed from the system dynamically simply by registering the change in the Multidatabase Directory. This approach has a number of advantages, including making it easy to add new DBMSs to a system or update a server to accommodate new versions of a DBMS, and allowing users to tailor the system to their needs by only including servers for DBMSs of interest.

OPM Database Servers handle several tasks, including implementing functions needed to determine what part of an OPM*QL query can be evaluated by the underlying DBMS, and evaluating single database OPM-QL queries issued by the multidatabase query processor.

The OPM Database Servers evaluate queries using the OPM Query Translators [5]. An OPM Query Translator takes queries specified in OPM-QL, and generates equivalent queries expressed in the query language supported by the underlying DBMS or file system. In the case of a relational DBMS, these query translators generate a single SQL query corresponding to the OPM-QL query. For flat file systems such as SRS [7], it may be necessary to generate several queries. The results of single database queries are subsequently structured and returned using OPM specific data structures.

The OPM query translators are driven by an OPM *mapping dictionary* that records the mapping between the elements of an OPM schema or view, such as classes and attributes, and the corresponding elements of the underlying database or file. The content of the mapping dictionary and OPM schema is

compiled into a *metadata* file that is dynamically linked into the OPM Database Server for efficiency.

Currently, OPM Database Servers are available for commercial relational DBMSs, such as Sybase and Oracle, and for structured flat-file systems. The relational OPM query translators generate queries in the SQL dialect supported by the underlying DBMS, and employ DBMS-specific C/C++ APIs for database access. Query results are formatted as sets of tuples.

The flat-file OPM Database Servers employ SRS (Sequence Retrieval System), developed at the European Molecular Biology Laboratory (EMBL) [7]. SRS is a system which uses grammars defined in a declarative language to parse and index structured flat files. Query conditions are limited to simple comparisons between indexed attributes and constants, while non-indexed attributes can be retrieved but not used in conditions. Consequently the parts of an OPM*QL query that can be evaluated using SRS are smaller than those that can be evaluated for a relational database, and local processing is needed. Query results from an SRS/flat-file database server are sets of objects which map directly into OPM data-structures.

The interfaces between the OPM Multidatabase Query Processor and the OPM Database Servers are implemented using CORBA based products, which provide a convenient tool for implementing client-server architectures. However CORBA's limitations regarding the data structures that can be passed between applications require extraneous conversions between data structures used in applications and those that can be passed.

4.4 Multidatabase Directory Information for Query Processing

The OPM Multidatabase Directory described in section 3.2 is used to provide the OPM Multidatabase System with information on the DBMSs, databases and inter-database links involved in the system. New DBMSs, databases and inter-database links can be added to a Multidatabase System simply by adding entries to the OPM Multidatabase Directory file, making the system easy to extend and adapt to changing requirements.

For each DBMS, the information required by the Multidatabase System includes the name and location of the OPM DBMS Server, and possibly a description of the DBMS and server.

For each database, the information required by the Multidatabase System includes the name of the database, information necessary for accessing the database (e.g., login name, password and server), and a pointer to the metadata file for the database. The metadata files include schema and mapping file information for the databases.

Inter-database *links* represented in the Multidatabase Directory represent known, meaningful connections between the databases, and are extremely useful in formulating multidatabase queries, while hiding many of the low level details needed for connecting databases. Such links connect classes in (usually) distinct databases. Each link has a name which can be used in OPM multidatabase queries in a similar manner to a regular abstract attribute, except that

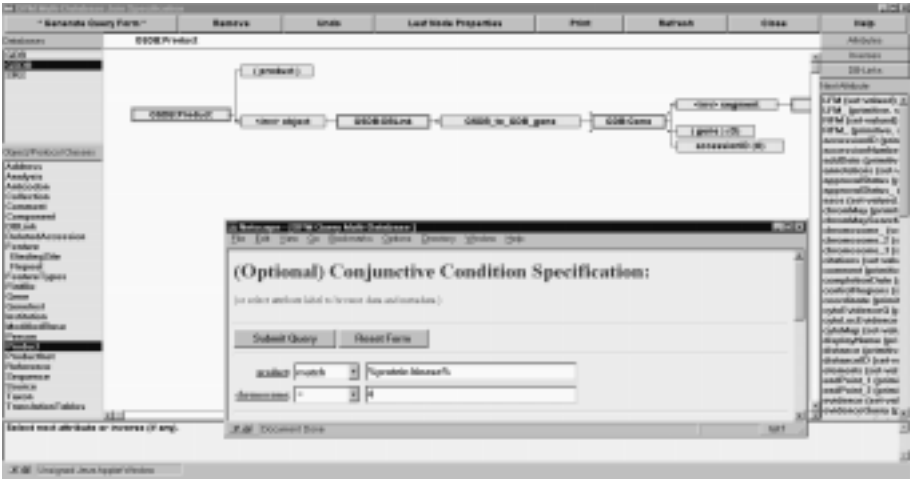

Fig. 3. Query Specification with the OPM Multidatabase Web Query Interface

the domain of the link may be a class in a different database. In addition to the link name and the class names and database names for the two classes it connects, the Multidatabase Directory contains the OPM*QL code necessary to implement a link. This code will often involve several other classes or additional databases, a number of additional conditions, and possibly some reformatting and coercions of various values. For documentation and interpretation purposes, the Multidatabase Directory contains comments and information explaining the semantics and the significance of links. Examples of such links are given in section 6 .

Links are interpreted in the first stage of query analysis by the OPM Multidatabase Query Processor, when they are simply replaced by the corresponding OPM*QL code stored in the Multidatabase Directory.

5 The Multidatabase Web Based Query Interfaces

An OPM Multidatabase Web Query interface provides support for constructing multidatabase queries by selecting databases, classes, and attributes of interest using a graphical user interface, and then dynamically generating HTML query forms containing the selected attributes, possibly from different classes and databases. Further query condition specification can be carried out by filling in these forms. Query results are organized in HTML pages with hyperlinks to related objects and metadata definitions in order to support Web browsing.

In order to specify a query, first a database involved in the multidatabase system is selected, and the classes of this database are displayed in a Classes list-box. A query tree is then constructed by first selecting a *root* class in the list-box and then expanding this tree by recursively selecting classes and attributes that

are involved in the conditions and/or output of the query. Attributes, inverse attributes and inter-database links associated with a selected class, are added to the tree by selecting one of the buttons Attribute, Inverses or DB-Links, and then selecting the attribute or link from the Next Attribute list-box.

In the example shown in Figure 3, GSDB is selected from the list of component databases, and class `Product` of GSDB is selected as the root of the query tree. Attributes, such as `product` and `gene` are then added to the tree by selecting (clicking on) a class in the diagrammatic representation of the tree and then selecting attributes from the Next Attribute list-box on the right-hand side. Primitive attributes, such as `product`, form the leaves of the tree and can be renamed. The selection process is repeated until all attributes and links of interest have been included in the query tree.

Once the query tree is completed, an HTML query form is generated (see the lower part of Figure 3). This form is used for specifying conditions on attributes in the familiar Query-by-Example mode (e.g., `chromosome_number` = "4"). Menu buttons help selecting the operator appropriate for the type of a given attribute. For controlled value classes, a list-box displays the set of valid values that can be used in expressing conditions. The query can then be passed to the OPM Multidatabase Query Processor, or the form can be saved as an HTML file that can be customized for subsequent use or inclusion in Web pages.

The two-stage query construction described above leads to the specification of a query in OPM*QL. For example, the query constructed in figure 3, for finding protein kinase genes on chromosome 4, is expressed by the OPM*QL query given in section 4.

The OPM Multidatabase Web Query tool is implemented using a combination of the Java programming language and HTML forms. The tool is schema driven in the sense that the OPM schemas and mapping information for all component databases are loaded once the interface is started. Communication between the Java tools and HTML forms and the Multidatabase Query Processor are implemented using a Common Gateway Interface (CGI) script.

6 An Application

We have employed the OPM Multidatabase tools for constructing several prototype multidatabase systems that involve heterogeneous databases accessed either remotely over the net or via copies located on the same server as the Multidatabase tools. In this section we discuss one of these prototypes, namely a federation of molecular biology databases that includes the Genome Database (GDB), the Genome Sequence Database (GSDB), and GenBank:

GDB is an archival database maintained at Johns Hopkins School of Medicine, in Baltimore [8]. GDB contains information about genomic data, literature references, people and organizations. Genomic data include information on genomic maps (e.g. genes associated with maps), biological functions, and experimental data about maps. The current version, GDB 6, is a native

OPM database: it was designed and implemented using the OPM data model and Database Development tools, and Sybase 11 is used as its underlying DBMS.[4]

GSDB is maintained at the National Center for Genome Resources, in Santa Fe. GSDB contains information mainly about nucleotide sequences and their features, which represent biologically interesting regions on these sequences.[5] GSDB is implemented as a relational database using Sybase 11, and the OPM retrofitting tools have been used to construct an OPM view on top of the relational views provided by GSDB for public access.

GenBank is one of the main international archival genome data repositories, maintained at the National Centre for Biotechnology Information (NCBI). GenBank contains information about nucleotide and protein sequences and supporting bibliographic and biological data.[6] GenBank is implemented as an indexed flat file. The OPM view of GenBank has been built using the OPM flat-file retrofitting tools and is based on the SRS (Sequence Retrieval System) interface to GenBank.

A Multidatabase Directory for this federation has been set up[7] and includes information regarding the component databases and their inter-database links. For example:

Sequences are the main entities recorded in GSDB and GenBank, and are represented by instances of class `Sequence` in both databases. Sequence data include the actual sequence, sequence length, information on the source (origin) of the sequence, and so on. Sequence information in GDB is represented by objects of class `NucleicAcidSequenceLink` that contain annotations representing links from primary GDB objects to external sequence databases such as GSDB and GenBank, as well as information regarding the beginning and end points of sequences. Nucleotide sequence references from GDB to GSDB and GenBank are represented by the inter-database links `GDB_to_GSDB_sequence` and `GDB_to_GENBANK_sequence` respectively. Evaluating the link `GDB_to_GSDB_sequence` involves (a) determining which external database is referenced, using attribute `externalDB` of class `NucleicAcidSequenceLink` and attribute `searchName` of class `ExternalDB`, in the OPM condition

```
NucleicAcidSequenceLink.externalDB[ExternalDB]searchName
        = "gsdb"
```

and (b) finding sequence objects in GSDB related to sequences in GDB using attribute `accessionID` of class `NucleicAcidSequenceLink` in a join condition with attribute `ic_accession` of class `Sequence` of GSDB:

[4] See `http://gdbwww.gdb.org/gdb/gdbDataModel.html`.
[5] See `http://www.ncgr.org/gsdb/schema.html`.
[6] See `http://www.ncbi.nlm.nih.gov/Web/GenBank/index.html`.
[7] `http://gizmo.lbl.gov/DM_TOOLS/OPM/MBD/MBD.html`

```
GSDB:Sequence.ic_accession
        = GDB:NucleicAcidSequenceLink.accessionID
```

The link from GDB to GenBank sequences is implemented in a similar way.

Genes are among the main entities recorded in GDB, where they are represented by objects of class `Gene` and are characterized by information that includes the evidence that a genomic region is considered a gene, and links to gene families the gene belongs to. GSDB also has a class `Gene`, but not necessarily representing the same real-word concepts as the GDB `Gene` class; GDB genes are in most cases represented by instances of other GSDB classes, such as `Product` and `Feature`. Gene references between GSDB and GDB are represented by inter-database link `GSDB_to_GDB_gene` which involves (a) determining which external database is referenced, using attribute `extdb` of class `DBLink` in GSDB in condition `DBLink.extdb = "GDB"`; and (b) finding genes in GDB related to genes in GSDB using attribute `extid` of class `DBLink` in a join condition with attribute `accessionID` of class `Gene` of GDB:

```
GSDB:DBLink.extid = GDB:Gene.accessionID
```

Some reformatting of accession numbers is also involved in defining this link.

In this prototype multidatabase system, GDB and GSDB are accessed remotely via the net, while GenBank is accessed via a local copy. Examples of queries specified over this system, expressed in OPM*QL or using the OPM Multidatabase Web Query Interface, were given in sections 4 and 5 respectively.[8] Figures 1 and 3 contain examples of using the OPM Multidatabase tools for browsing and querying this system.

7 Concluding Remarks

The existing version of the OPM Multidatabase tools has demonstrated the feasibility and advantage of our tool-based incremental strategy in dealing with heterogeneous databases. Work is under way to improve the Multidatabase Query Processor by developing increasingly efficient query processing strategies, including concurrency and lazy evaluation techniques for evaluating distributed queries, and employing heuristics for estimating the cost of individual subqueries and determining an optimal evaluation order. We also plan to experiment with using secondary storage, rather than main memory, for intermediate results, in order to handle queries involving large amounts of intermediate data, and to allow the results of multidatbase queries to be stored for further querying.

The most difficult problems of querying multiple heterogeneous databases include (i) formulating a query, which involves determining which databases contain relevant data, understanding how data are represented in each of these

[8] This prototype multidatabase system is available on the Web at `http://gizmo.lbl.gov/jopmDemo/gdbs_mqs.html`

databases, and how data in these databases relate to one another; and (ii) interpreting the result of a multidatabase query. The facilities provided by the Multidatabase Directory and the Web based interfaces address the first problem. The second problem has been only partly addressed. We plan to use information in the Multidatabase Directory together with the semantics of the operations underlying multidatabase query processing for interpreting results of queries expressed across multiple databases. For example, information on the semantics of objects in a given class can be used for annotating query results, while information about inconsistent interdatabase links can be used for explaining null query results.

References

1. Buneman, P., Davidson, S., Hart, K., Overton, C., and Wong, L., A Data Transformation System for Biological Data Sources. In *Proc. of the 21st Int. Conference on Very Large Data Bases*, pp. 158-169, 1995.
2. Buneman, P., Naqvi, S., Tannen, V., Wong, L., Principles of Programming with Complex Objects and Collection Types. In *Theoretical Computer Science 149*, pp 3–48, 1995.
3. Chen, I.A., and Markowitz, V.M., An Overview of the Object-Protocol Model (OPM) and OPM Data Management Tools. *Information Systems*, 20(5), pp. 393-418, 1995.
4. Chen, I.A., and Markowitz, V.M., OPM Schema Translator 4, Reference Manual, Technical Report LBL-35582, 1996.
5. Chen, I.A., Kosky, A., Markowitz, V.M., and Szeto, E., The OPM Query Translator, Technical Report LBL-33706, 1996.
6. Chen, I.A., Kosky, A.S., Markowitz, V.M., and Szeto, E., Constructing and Maintaining Scientific Database Views, Proc. of the 9th Conference on Scientific and Statistical Database Management, IEEE Computer Society, pp. 237- 248, 1997.
7. Etzold, T., and Argos, P., SRS, An Indexing and Retrieval Tools for Flat File Data Libraries. *Computer Applications of Biosciences*, **9**, 1, pp. 49-57, 1993. See also http://www.embl-heidelberg.de/srs/srsc.
8. Fasman, K.H., Letovsky, S.I., Cottingham, R.W., and Kingsbury, D.T., Improvements to the GDB Human Genome Data Base. Nucleic Acids Research, Vol. 24, No. 1, pp. 57-63, 1996. See also http://wwwtest.gdb.org/gdb/about.html.
9. Markowitz, V.M., Chen, I.A., Kosky, A.S., and Szeto, E., Facilities for Exploring Molecular Biology Databases on the Web: A Comparative Study. Pacific Symposium on Biocomputing'97, Altman, R.B. & al (eds) World Scientific , 1997.
10. *The Object Database Standard: ODMG-93*, Cattell, R. G. G. (ed), Morgan Kaufmann, 1996.
11. Programmer's Reference. National Center for Biotechnology Information, Bethesda, Maryland, November 1991. See also the ASN.1 homepage at `http://www.inria.fr:80/rodeo/personnel/hoschka/asn1.html`.
12. Ritter, O. The Integrated Genomic Database. In *Computational Methods in Genome Research* (S. Suhai, ed.), pp. 57-73, Plenum, 1994.
13. Sheth, A.P., and Larson, J.A. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), pp. 183-236, 1990.